IMPLEMENTATION AND VERIFICATION OF ORDERED MESH NETWORK INTERCONNECT

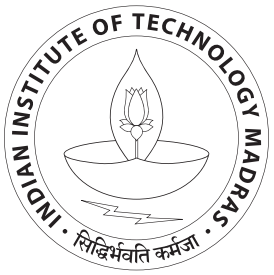
A Project Report

submitted by

M.KRISHNA CHAITANYA

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2018

THESIS CERTIFICATE

This is to certify that the thesis entitled **IMPLEMENTATION AND VERIFICATION OF ORDERED MESH NETWORK INTERCONNECT**, submitted by **M.KRISHNA CHAITANYA**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. V. Kamakoti
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 15 MAY 2018

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my guide Dr. V. Kamakoti for all the valuable advice and motivation he gave from time to time. Despite a very busy schedule, he always gave me a patient hearing. His knowledge and an extraordinary ability to lighten the students with his positive approach towards things always infused me with great energy and positivity. The invaluable inputs and suggestions from him enabled me to achieve the desired goals during the project work.

I would like to extend a special thanks to my faculty advisor Prof. Saurabh Saxena who continuously supported me throughout my course with his knowledge and advise.I would sincerely like to thank Rahul Bodduna who has been of immense help and very supportive through out the project. I would like to extend special thanks and deepest gratitude to Dr. Neel Gala and Arjun Menon who have been very supportive during the course of this project. They have enriched the project experience with their knowledge of the subject matter, active participation, deep understanding and invaluable suggestions. I would like to thank Roberta for her constant support and inspiring me to complete this project.I would like to thank all my friends for wonderful memories.

Last but not the least, I would like to thank my parents and all of my friends for their constant encouragement and support.

ABSTRACT

KEYWORDS: Multi Core Processor; Virtual Channel; Flits; Router Architecture; Network Topology; Mesh Interconnect.

It is seen that in the two last decades that Moore's Law continues to hold true and more and more computing elements are being packed into the same area of silicon, the performance per unit area has been increasing as shown in Moore's Law. Increase in performance has given rise to further challenges like heat dissipation. The problem of heat dissipation gave birth to the concept of Multi Core Processors wherein multiple processing elements are mounted on a single chip. The SHAKTI processor project, which is being undertaken at RISE Lab in Computer Science Department at Indian Instituite of Technology Madras, is based on RISC - V Instruction set architecture from UC Berkeley. As part of SHAKTI processor project, this project aims to verify the implementation of ordered mesh network so that it can be used to implement Snoopy Cache Coherence for Network on Chip.

In a shared memory architecture there are are number of shared caches because there are number of independent cores. The overall performance of such architecture depends on the On-chip latency experienced by packets while traversing across multiple cores using interconnection network. An interconnection network for mesh topology, which has been implemented in this project, provides significant advantages in terms of performance, power and timing as compared to ring or bus topology. Any node can initiate cache request and the same will be broad-

casted throughout the entire mesh network and whenever a hit occurs at any of the node in the network, a response will be sent back from that node to the requesting node. Requests from different nodes are ordered by notification network so that all nodes maintain the same order, for the purpose of processing a request.Such network is implemented and verified using design verification methods.

The HDL that has been used to implement the mesh network and verify it is Bluespec System Verilog (BSV).The proper functioning of the mesh fabric is verified by giving test cases to the fabric.Also attempts are made to optimize the logic of node used in mesh network.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
BSV	Bluespec System Verilog
RISE	Reconfigurable and Intelligent Systems Engineering
HDL	Hardware Description Language
NoC	Network on Chip
VC	Virtual Channel
RISC	Reduced Instruction Set Computing
CMOS	Complimentary Metal Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
CMP	Chip Multiprocessors
NIC	Network Interface Controller
vcid	Virtual Channel Identifier

CHAPTER 1

INTRODUCTION

Gordon Moore in 1965 has said that the number of transistors per square inch of an Integrated Circuit (IC) will roughly double every two years. This is also commonly called as Moore's law. Thus, every few years there has been a drastic increase in the computing performance of silicon chips of unit area, as more and more computing elements could be packed on the same size of silicon chip. Advent of CMOS technology made it further possible to run Silicon chips at a higher clock frequency and thus increasing performance.

As per MOSFET scaling theorem, as the transistor becomes smaller in size, their power densities still remain constant. Thus power consumed by the circuit and area of chip are proportional to each other. Thus, a functionally equivalent circuit would consume less power for same frequency and if we increase the frequency we can enhance the performance without increasing power. As a result of above processor manufacturers from 1985 to early 2000, were able to improve performance of processor by approx 50 percent every year. But after 2005, MOSFET scaling almost stopped and proportional performance could not be increased only by increasing clock frequency. As size started shrinking the problem of leakage current started becoming prominent and started consuming significant part of power supply. Also, if one tried to increase the frequency without scaling down voltage, led to heating up of the chips and in some cases even resulted in a thermal runaway.

To overcome all the above mentioned problems, industry moved to the concept

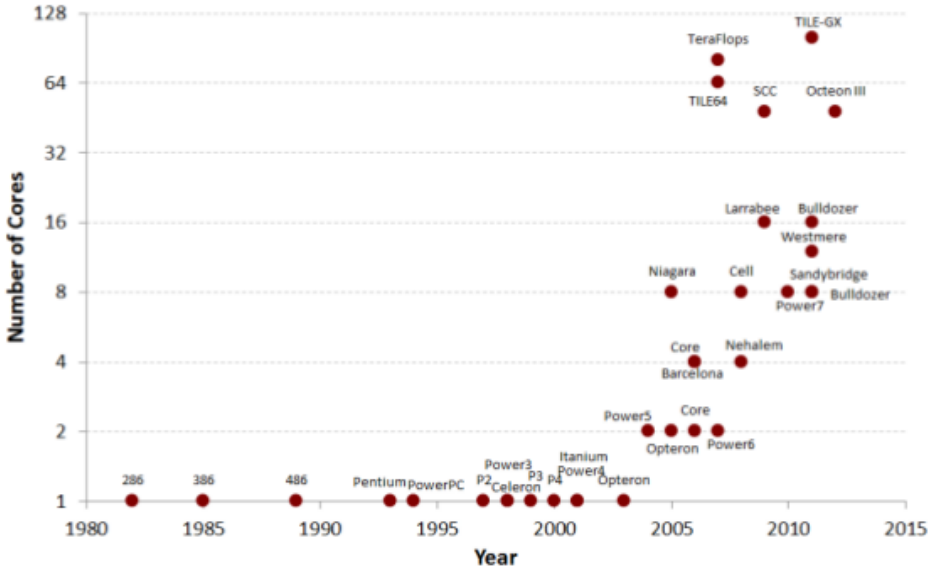


Figure 1.1: **Shift to multicore**

of Multi Core Processor.The fig 1.1**Shift to multicore**figure.1.1 shows that industries have been moving towards multi core Processors. The objective was to pack identical cores on a common substrate and as the cores were identical they could be easily replicated. All multiple cores would run on same and highest clock frequency. The objective was to run all those applications that were compatible with parallel processing on these multi core processors. It can also be stated that a larger problem being solved by applications could be fragmented down into smaller and independent problems that would be handled by threads of application and each thread would run on a specific core that has been assigned to it. The maximum benefit of scaling down of transistors can be derived if one packs as many cores per chip as possible, thus significantly increasing the performance per unit area.

In order to connect all the cores together what is very important is to develop a scalable and high bandwidth communication fabric. A small number of cores can be connected using conventional methods like bus interconnect, however, such conventional interconnect can't be scaled down beyond a limit. Bus interconnect

does not support higher bandwidths as number of cores increases, though it is excellent for cache coherence traffic using snoopy protocol. As more and more cores are added the throughput decreases considerably. Another alternative to bus interconnect is crossbar and it does not suffer from low throughput, however, as the number of cores increases, area and power requirement of the crossbar interconnect increases considerably. Number of ideas have been proposed to overcome above mentioned difficulties but Network on Chips (NoC) have emerged as one of the most appropriate media as they occupy less chip area, consume less power and can be scaled.

A good interconnect for communication between the cores is essential for optimization of the architecture. As interconnect keeps on getting complicated, its cost keeps on increasing and it will take more area and thus consume more power.

Optimization of Interconnects in a multi-core processor have become one of the key area for researchers. The primary aim of optimization is to improve performance and decrease the cost incurred on exchange of data between the cores. The architecture and logic selected for the design of interconnect plays a key role in achieving desired functionality in the network. Routing algorithm also affects the performance of interconnects. All the optimization have a direct impact on power consumption, therefore, while designing an interconnect it is important to keep in mind the overall power consumed by the processor.

The thesis is organised as follows:Chapter 2 discusses the work done on multi core processors and how different coherency mechanisms are proposed.Chapter 3 talks about interconnects and how they are implemented.Chapter 4 discusses about various aspects in designing ordered mesh network interconnect.Chapter 5 talks how the implementation of the mesh fabric and verification of the mesh

fabric.Chapter 6 gives the results and chapter 7 gives conclusion.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 MEMORY CONSISTENCY AND CACHE COHERENCE

2.1.1 MEMORY CONSISTENCY OVERVIEW

Single threaded serial programs present a simple and intuitive model to the programmer. All instructions appear to execute in a serial order2 , and a thread transforms a given input state into a single well-defined output state. The consistency model for a single threaded program on a uniprocessor, therefore, maintains the invariant, that a load returns the value of the last store to the corresponding memory location.

Shared memory consistency models are concerned with the loads and stores of multiple threads running on multiple processors. Unlike uniprocessors, where a load is expected to return the value from the last store, in multiprocessors the most recent store may have occurred on a different processor core. With the prevalence of out-of-order cores, and multiple performance optimizations such as write buffers, prefetching etc., reads and writes by different processors may be ordered in a multitude of ways. To write correct and efficient shared memory programs, programmers need a precise notion of how memory behaves with respect to reads and writes from multiple processors. Memory consistency models define this behavior, by specifying how a processor core can observe memory references made

from other processors in the system. Unlike the uniprocessor single threaded consistency model which specifies a single correct execution, shared memory consistency models often allow multiple correct executions, while disallowing many more incorrect executions.

While there are many memory consistency models for multiprocessor systems, arguably the most intuitive model is sequential consistency. Lamport was the first to formalize the notion of sequential consistency. A single processor core is said to be sequential if the result of an execution is the same as if the operations had been executed in the order specified by the program. A multiprocessor system is said to be sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program. This total order of memory operations is referred to as memory order. Figure 2.1**Example for sequential consistency**figure.2.1 depicts a code segment involving two processors P 1 and P 2. The critical section could be a portion of code that attempts to access a shared resource that must not be accessed concurrently by more than one thread. When P 1 attempts to enter the critical section, it sets the flag1 to 1, and checks the value of flag2. If flag2 is 0, then it implies P 2 has not tried to enter the critical section, and therefore it is safe for P 1 to enter. The assumption here is that if flag2 is read to be 0, then it means P 2 has not written flag2 yet, and consequently not read flag1 either. However, since processor cores apply several optimization techniques for improving performance, such as out-of-order execution, this ordering may not hold true. Sequential consistency ensures this ordering by requiring that the program order among operations by P 1 and P 2 be maintained. Sequential consistency conforms to the typical thought process when programming sequential code, and thus presents a very intuitive model to

programmers to reason about their programs.

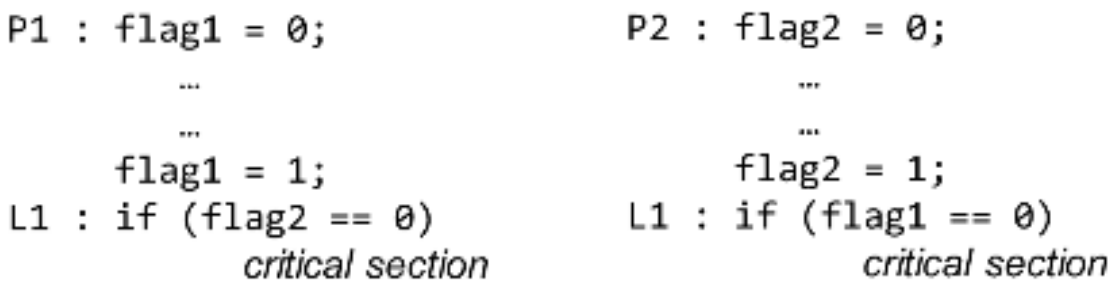


Figure 2.1: **Example for sequential consistency**

Multiple cores in a multi-core processor while working simultaneously, share various resources like memory and memory devices. This arrangement has many advantages as compared to the single core processor.

- As resources are shared, cost per processor is reduced significantly.
- It increases redundancy of the chip with multi processor and thus increasing reliability, as failure of one or more processor will only affect performance and not functionality.
- There is an increase in performance/throughput as more number of processors will complete the task in less time as compared to single processor.

Frequently used instructions and data are stored in a Cache memory, which is nothing but a computer memory and has a very short access time. Since cache memory is generally built within Central Processing Unit, it is very fast. The instructions or data that are frequently required by CPU to run program are stored in Cache memory. Whenever the processor wants to fetch some data or instruction from memory, it is first checked in cache memory and if it is available there then it is fetched, otherwise it will be fetched from main memory. Cache has number of advantages and some of them are as follows:

- As cache is co-located with processor it reduces the average data access time, thus improving the overall speed of the chip.

- Bandwidth demand on the interconnect is reduced.
- Being a volatile memory, it can be used again and again.

Cache memory has its own problems of Coherence and Consistency when used with multi-core processor. Coherence problem will come when more than one processor will cache an address at the same time. Consistency problem will come when a processor updates a data item and does not inform the other processor, inconsistency may lead to incorrect execution.

2.2 COHERENCY MECHANISMS AND PROTOCOLS

Permissions are attached to each block which are stored in a processor's cache by the cache coherence protocols. This permission helps in taking appropriate action for the block, when access is sent to the local cache for every load or store operation. These permissions are called as the state of the cache line. Three of the commonly used states in a cache are enumerated below:

- **Invalid (I)**: The block stored in the processor's cache is not valid. Either it is an old copy that can't be read or written or cache does not contain the block.
- **Shared (S)**: The block stored in the processor's cache is valid, but the block can only be read. This also signifies that other processor caches in the system may be sharing this block with only read permission.
- **Modified (M)**: The block is held in the respective processor's cache with access to both read and write. This also signifies that the only valid copy of the block is held in the cache and the same has to respond in case of a request for this block.

Other than the above mentioned states, there are two more states, one is Exclusive (E) state if there are no copies of the block in other caches and it allows for implicit write access. Second is Owned (O) state which signifies that the cache will source a remote coherent request.

Two main approaches to Cache Coherence protocols are:

- **Broadcast based snoopy protocols:** All coherence requests are observed by all the coherence controllers and controllers take appropriate action to maintain coherence. In this protocol all requests to a block arrive in an order. This system ensures that the caches block state which are represented by finite state machine are correctly updated by coherence controllers. This protocol is based on ordered interconnects for example bus or tree, to ensure correct ordering. Block ordering becomes a subset of total ordering and thus coherence is maintained. Also, this system makes it easier to implement memory consistency models that needs total ordering of all memory transactions.
- **Directory based protocols:** The main concept of this protocol is that a global view of the coherent state of each block is maintained in a directory. The status of each block along with its corresponding state can be tracked by this directory. Appropriate action is determined by looking at the state of the block in directory. In this there is no requirement of broadcasting messages to all nodes and messages are sent to the relevant nodes by virtue of sharer list for each block. Ordering of coherent transactions with respect to other transactions needs to be clearly defined in this protocol, as was done in snoopy protocol also. Transactions in most of the directory protocol are ordered at the directory. However, sequential consistency can only be achieved by additional mechanism.

One of the advantage of directory protocol is its low communication bandwidth requirement and thus can easily be used with large number of cores. However, directory indirection causes higher latencies. Also, considerable area and power is required for maintaining a directory of large number of cores. Lastly, directory protocols are difficult to implement and require considerable effort for verification to ensure memory consistency.

Multi-core processors have been mainly dominated by snoopy protocols as snoopy protocols are simple to design, have low overheads as compared to directory based protocols. Also, sequential consistency is easy to implement in snoopy coherence due to total ordering of requests. Dependence on ordered interconnect and broadcast overhead are two main hurdle in scaling snoopy coherence protocol to many core chips.

2.3 RELATED WORKS

2.3.1 SNOOPY COHERENCE ON UNORDERED INTERCONNECTS

Uncorq is an embedded ring coherence protocol in which snoop requests are broadcast to all nodes, using any network path. However along with the snoop request broadcast, a response message is initiated by the requester. This response message traverses a logical ring, collecting responses from all the nodes and enforcing correct serialization of requests. However, this method requires embedding of a logical ring onto the network. In addition, there is a waiting time for the response to traverse the logical ring. Further, Uncorq only handles serialization of requests to the same cache line which is sufficient to enforce snoopy coherence, but does not enforce sequential consistency. Multicast snooping uses totally ordered fat-tree networks to create a global ordering for requests. Delta coherence protocols also implement global ordering by using isotach networks. However, these proposals are restricted to particular network topologies.

Time-stamp snooping (TS) is a technique that assigns logical time stamps to requests and re-orders packets at the end points. It defines an ordering time (OT) for every request injected into the network. Further each node maintains a guaranteed time (GT), which is defined as the logical time that is guaranteed to be less than the OTs of any requests that may be received later by a node. Once a network node has received all packets with a particular OT, it increments its GT by 1. Nodes exchange tokens to communicate when it is safe to update their GT. Since multiple packets may have the same OT, each destination employs the same ordering rule to order such packets. While TS allows snoopy coherence to

be implemented on unordered interconnects, it has a few drawbacks. Each node needs to wait for all packets with a particular OT to arrive before it can process them and update its GT. This can increase the latency of packets. In addition, it also requires large number of buffers at the end-points which is not practical. TS also requires updating of slack values of messages buffered in the router, resulting in additional ports in the router buffers.

In-Network Snoop Ordering (INSO) maps snoopy coherence onto any unordered interconnect, by ordering requests in a distributed manner within the network. All requests are tagged with distinct snoop orders which are assigned based on the originating node of the request. Nodes process messages in increasing sequence of snoop orders. However, INSO requires unused snoop orders to be expired periodically, through explicit messages. If the expiration period is not sufficiently small, then it can lead to degradation of performance. On the other hand, small expiration periods also leads to increased number of expiry messages, especially from nodes that do not have any active requests. This consumes power and network bandwidth, which is not desirable for practical realizations. While the INSO proposal suggests using a separate network for these expiry messages, it can lead to subtle issues where expiry messages overtake network messages, which could void the global ordering guarantee.

Intel's QPI and AMD's Hammer protocol are industrial protocols that extend snoopy coherence for many-core processors. AMD Hammer is similar to snoopy coherence protocols in that it does not store any state on blocks stored in private caches. It relies on broadcasting to all tiles to solve snoop requests. To allow for implementation on unordered interconnects, the Hammer protocol introduces a home tile that serves as an ordering point for all requests. On a cache miss, a tile

sends its request to the home tile which then broadcasts the same to all nodes. Tiles respond to requests by sending either an acknowledgement or the data to the requester. A requester needs to wait for responses from all tiles, after which it sends an unblock message to the home tile, thereby preventing race conditions. While this protocol allows for snoopy coherence on unordered interconnects, the ordering point indirection latency can become prohibitively large as the core count increases. Intel's QPI implements a point-to-point interconnect that supports a variant of snoopy coherence, called source snooping. QPI also introduces a home node for ordering coherence requests. However, they add a new state Forward to their coherence protocol to allow for fast transfer of shared data. QPI is well suited for a small number of nodes, and scales well for hierarchical interconnects.

2.3.2 NoC PROTOTYPES and SYSTEMS

MIT's RAW is a tiled multicore architecture that uses a 4x4 mesh interconnection network to transmit scalar operands. However, this network is also used to carry memory traffic between the pins and the processor for cache refills. The TRIPS processor uses an on-chip operand network to transmit operands among the ALU units within a single processor core. It also uses a 4x10 wormhole routed mesh network to connect the processor cores with the L2 cache banks and I/O controllers.

IBM's Cell is a commercial processor that uses ring interconnects to connect its processing elements, external I/O and DRAM. It uses four ring networks and uses separate communication paths for commands and data. The interconnect supports snoopy coherence, and uses a central address concentrator that receives and orders all broadcast requests.

Tilera's TILE64 is a multiprocessor consisting of 64 tiles connected together by

five 2D mesh networks. The five mesh network support distinct functions and are used to route different types of traffic. Four of the five networks are dynamically routed and implement wormhole routing, while the static network is software scheduled.

Intel's Teraflops research chip demonstrated the possibility of building a high speed mesh interconnection network in silicon. The prototype implements 80 simple RISC-like processors, each containing two floating point engines. The mesh interconnect operates at 5 GHz frequency, achieving performance in excess of a teraflop.

Intel's Single Chip Cloud computer (SCC) connects 24 tiles each containing 2 Pentium processors through a 4x6 mesh interconnect. The interconnect uses XY routing and carries memory, I/O and message passing traffic. The NoC is clocked at 2 GHz and consumes 6W of power. The hardware is not cache-coherent, and instead supports the message passing programming model.

CHAPTER 3

INTERCONNECTION NETWORKS

Interconnection networks are composed of switching elements. An interconnection net- work in a parallel machine transfers information from any source node to any desired destination node. This task should be completed with as small latency as possible and it should also allow a large number of such transfers to take place concurrently. Packet- switched Networks-on-chip (NoC) provide scalable bandwidth for Multi-core processors and they also have relatively low latency. An NoC usually comprises of a collection of routers and links that connect various processor nodes. Routers take care of the communication between cores by multiplexing the traffic flowing in from different input links onto the output links. A message transmission in a NoC occurs by breaking the message into packets before injection into the network. A packet comprises one or more flow-control-units or flits which are the smallest units of the message on which flow-control is performed. A flit, in turn, is composed of one or more physical-digits or phits which is the number of bits that can be transmitted over a physical link in a single cycle. The primary features that characterize a NoC are:

- Topology
- Routing algorithm
- Flow control mechanism
- Router micro-architecture

3.1 TOPOLOGY

It refers to the physical layout and connection of the nodes and links in the network. The performance and cost of the network is affected by the topology that is being used. It determines the number of hops that a message takes to reach its destination, as well as the wire length of each hop. Thus topology significantly influences the latency of the network. The power consumption also depends on the number of hops since it involves storing and forwarding of the packets. The topology also affects the path diversity available in the network, which refers to the number of unique paths that exist between a pair of source and destination nodes. A higher path diversity leads to greater robustness from failures, while also providing greater opportunities to balance traffic across multiple paths. The three major interconnect topologies in use are ring, mesh and ring-mesh hybrid interconnect or torus.

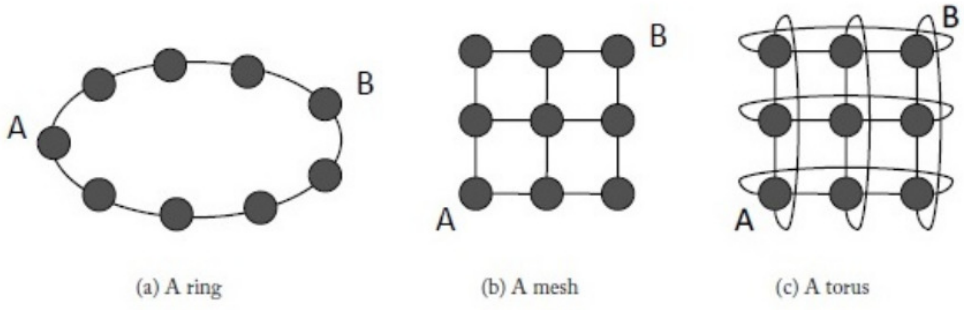


Figure 3.1: **Popular Network Topologies**

Figure 3.1**Popular Network Topologies**figure.3.1 shows these three main interconnect topologies. The degree of a node refers to the number of links connected at the node. The greater the number of links the lower will be the latency but at the cost of increased silicon area and wire lengths. Also, it can be seen that for equal number of nodes, hop count will be maximum for ring topology and minimum for torus.

3.1.1 OTHER RELEVANT TOPOLOGIES

Apart from the basic topologies discussed above, there are other relevant topologies in use which are derived from the basic topologies. One among them is the butterfly interconnect as shown in figure 3.2**The Butterfly Interconnect**figure.3.2. There are a number of switches in between the the source and destination nodes which performs the routing. Unlike mesh or torus the hop count here between any two pairs of nodes is exactly the same as the packets pass through the same number of stages. The hop count seems to have reduced but there is the disadvantage of an increased number of wires used in the design.

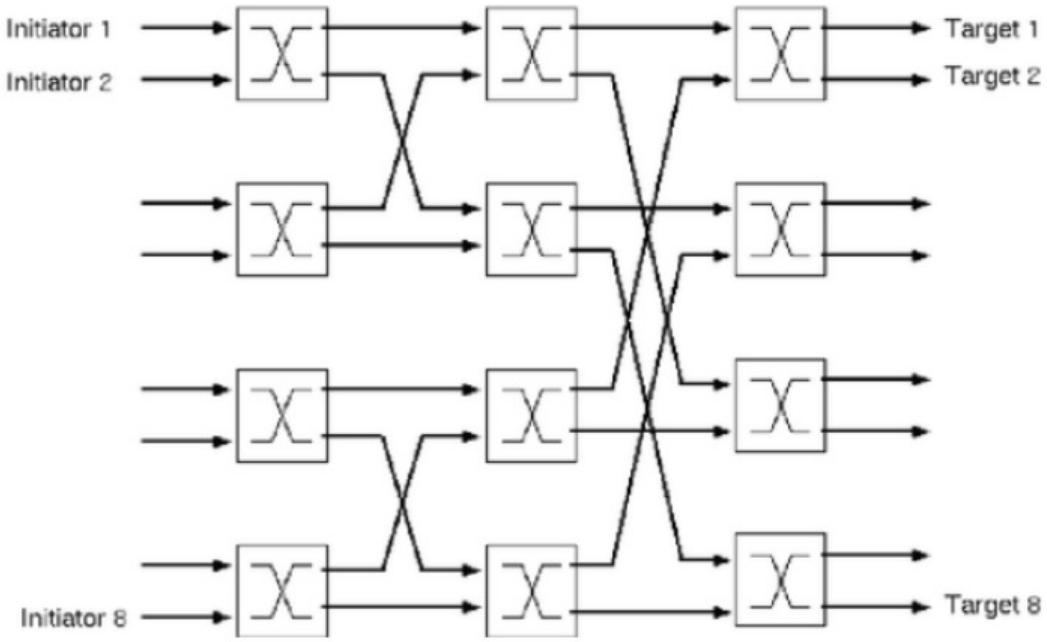


Figure 3.2: **The Butterfly Interconnect**

Another popular topology is the crossbar. It makes use of an arbitration unit which performs the control logic required for routing the packets. All the source and destination nodes are directly connected to the crossbar. Then there is the fat tree topology which is in the form of a binary tree. Here the source and destination nodes are the leaf nodes and the internal nodes are the switches. The same is shown in figure 3.3**The Fat Tree Topology**figure.3.3.

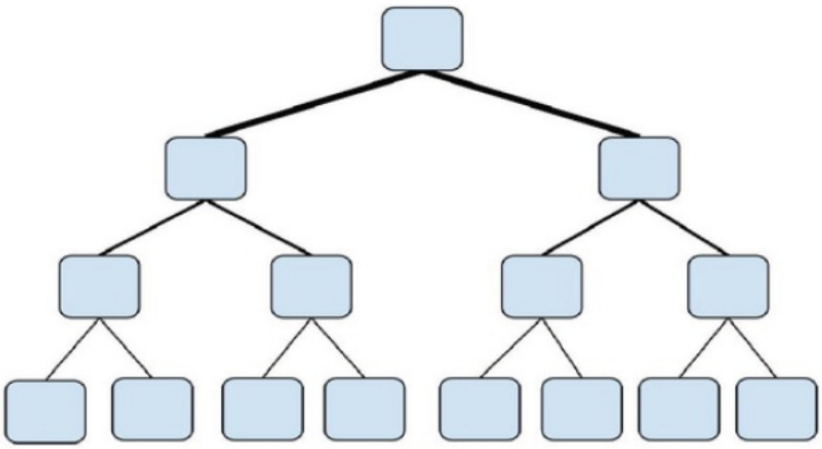


Figure 3.3: **The Fat Tree Topology**

In this project the implementation has been done on the mesh network. Mesh network provides scalable bandwidth. It has several advantages over the other popular network topologies. The problem with ring is delay, with crossbars it is area and with bus it is bandwidth. Also mesh can handle high traffic and it is robust. Mesh interconnect is suitable for higher dimension NoCs and is extensively used in the research of low power and low latency routers.

3.2 ROUTING ALGORITHM

For a given network topology routing algorithm determines the path taken by the packet from the source node to the destination node through the network. A

good routing algorithm should aim to distribute the network traffic evenly, thereby avoiding network congestion and it should also provide good latency.

Routing algorithms are basically classified into three, which are deterministic, oblivious and adaptive routing. In deterministic routing, the packets always follow the same path for a given pair of source and destination nodes. In oblivious and adaptive routing, for the given source and destination pairs, the packet may follow different paths at different times. This is because, in deterministic algorithms the path is chosen without taking into consideration the state of the network but in the other algorithms network congestion and other factors are considered while choosing the path.

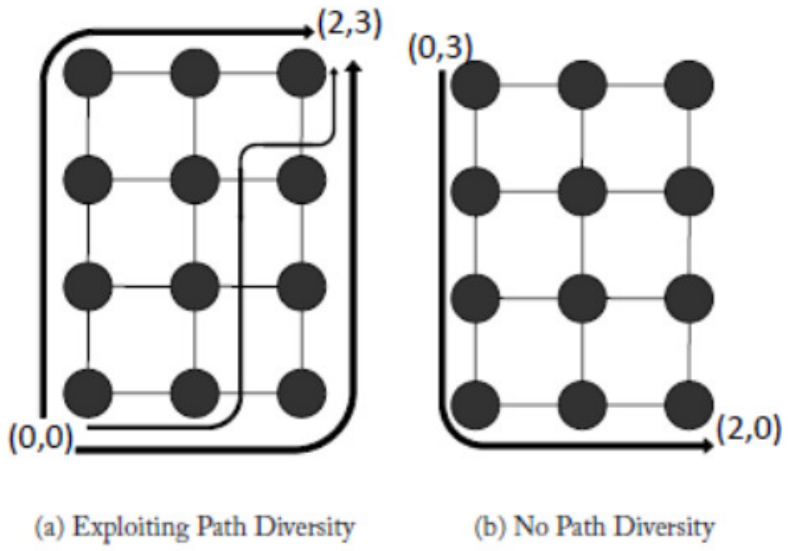


Figure 3.4: **Routing Algorithm**

A routing algorithm is chosen by considering its effect on the network latency, throughput, complexity, cost, delay etc. Oblivious and adaptive routing algorithm suffer from the problems of deadlock, live lock and starvation. A deadlock usually occurs when two or more packets are waiting for each other to be routed forward. These packets may have reserved some resources and are waiting for each other

to release the resources. Live lock occurs when a packet keeps spinning around its destination without ever reaching it. It is a common problem that exists in non-minimal type of routing algorithms. Complex networking algorithm and different set priorities can often lead to circumstances where lower priority packets never reach their destinations. This is referred to as starvation. This mainly happens when the packets with higher priorities reserve the resources all the time.

3.3 FLOW CONTROL

Network resources usually include buffers, links, channels etc. Flow control determines how these resources are allocated to the packets. A good flow control mechanism should allocate resources evenly so that congestion is reduced,bandwidth is increased and latency is also reduced.

The most basic flow control technique is circuit switching. Here the message as a whole is transmitted from the source to destination in one go without waiting at any of the intermediate nodes. In order to enable this, a probe packet is send first in order to reserve the resources required for the message to pass. The actual transmission of message is initiated only after an acknowledgement signal is received after resource allocation. This provides power and latency savings. But the links remain idle until acknowledgement of resource allocation is received and it cannot be used by any other packet. Therefore this results in reduced throughput.

Another method that overcomes several of the disadvantages of circuit switching is packet switching. Here the messages are broken down into packets and are buffered. Switching decision is made at every intermediate node. Packets from several sources can share links and therefore there is better link utilization in this

method. There are two variants of packet switching. In store and forward method of packet switching, a packet is transmitted further only if it has been received in its entirety at the current node. This requires large buffer spaces and can also introduce delays in the intermediate nodes. The other one is the virtual cut through method where a part of the packet is transmitted further even though some portion of it has not yet been received at the current node. This reduces latency but still requires buffer storage.

A flit based flow control technique has been found to be suitable for NoCs. Due to the granularity of the flits buffer space is reduced which provides area and power savings. Also flits could be transmitted forward independently as in virtual cut through technique. Therefore there is improvement in latency.

3.4 ROUTER MICROARCHITECTURE

The basic architecture of the router should be such that latency and throughput are im- proved. The flits while traveling through the network spend most of their time at the routers waiting for the switching decision. Therefore delay of the router affects the delay between hops and thus the overall delay suffered by the packets from source to destination. Thus it also affects the maximum frequency at which the system can operate. Typical routers are usually pipelined. Area and power considerations should also be taken into account while designing the router as it has a direct impact on those factors.

CHAPTER 4

DESIGN ASPECTS OF OMNI NETWORK

4.1 OVERVIEW

Snoopy protocols depend on the logical order and not the physical time at which requests are processed, i.e. the physical time at which requests are received at the nodes are unimportant so long as the global order in which all nodes observe requests remains the same. Traditionally, global ordering on interconnects have relied on a centralized arbiter or a global ordering point. In such a scheme, all messages are first sent to the ordering point, which then broadcasts the messages to the rest of the system. The interconnection network needs to ensure that the order in which messages leave the ordering point is the same as the order in which the nodes observe the requests. However, as the number of nodes increases, such a mechanism introduces greater indirection latency. Forwarding to a single ordering point also creates congestion at the ordering point degrading network performance.

In OMNI, the requirement of a central ordering point has been removed, and instead, this responsibility has been entrusted to each individual node. Fundamentally, a decision on ordering essentially involves deciding which node to service next. All nodes in the system are allowed to take this decision locally, but guarantee that the decision is consistent across all the nodes. At synchronized time intervals referred to as time window, all nodes in the system perform a local decision on which nodes to service next, and in what order to service these nodes. The

mechanism for the same is as follows. Consider a system with N nodes numbered from 1 through N referred to as source ID (SID). The nodes are connected by an interconnection network of a particular topology, This network is referred as the main network. All messages from a particular node are tagged with the SID. Now, for every message injected into the main network a corresponding notification message is constructed. This message essentially contains the SID, and indicates to any node that receives this notification message, that it should expect a message from the corresponding source.

The notification message is sent to all nodes through a separate 'fast' network, this fast network is referred as the notification network. At the end of each synchronized time interval, it is guaranteed that all nodes have received the same set of notification messages. For every node, this establishes the set of sources they will service next. Every node performs a local decision on the order in which it will service these sources, for example, the decision rule could be 'increasing order of source ID'. Consequently, this fixes the global order for the actual messages in the system. This global order is captured through a counter maintained at each node called the expected source ID (ESID). Messages travel through the main network and are delivered to all nodes in the system. However, they are processed by the network interface (NIC) at every node in accordance to the global order.

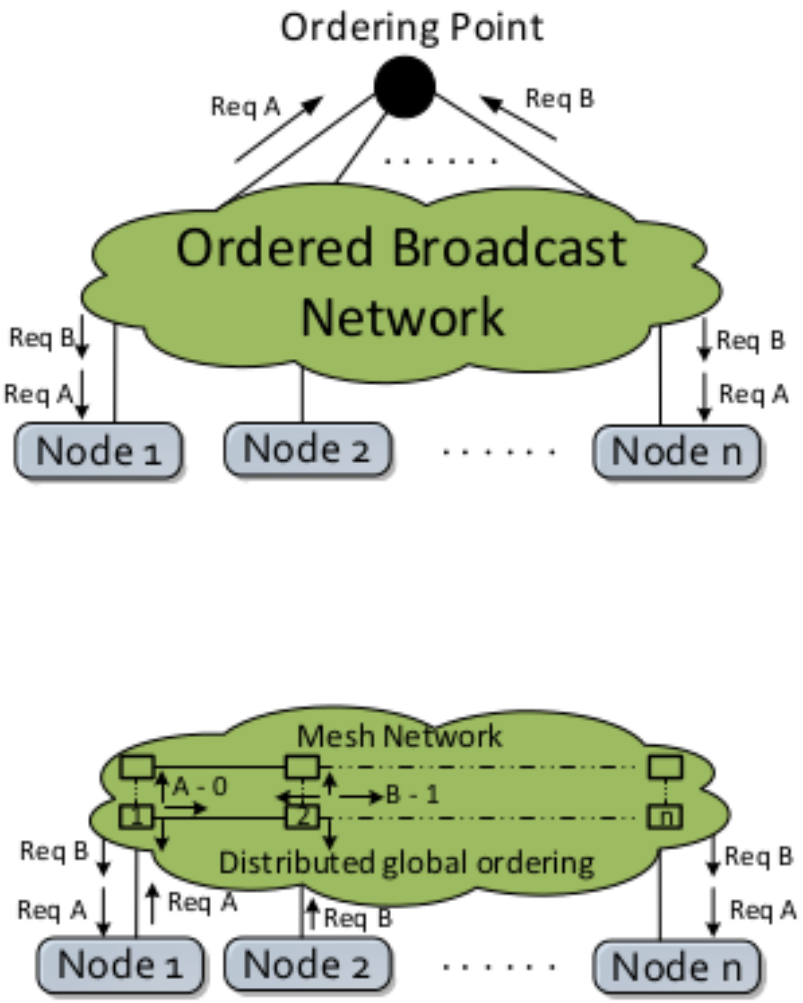


Figure 4.1: **Ordering Point vs OMNI**

4.1.1 WALKTHROUGH EXAMPLE

A detailed walkthrough of OMNI is presented here. For simplicity, the walkthrough example considers a 16-tile CMP system. Each tile comprises a processor core with a private L1 cache, and a private L2 cache attached to a router. Two memory controllers on two sides of the chip are connected to the routers as well. Cache coherence is maintained between the L2 caches and the main memory. The routers are connected in a 4x4 packet switched mesh network, i.e. the main network is a 4x4 mesh. The 'fast' notification network, in this example, is a contention-free light-weight 4x4 mesh network. We walkthrough how two requests M 1 and M 2 are handled and ordered by OMNI.

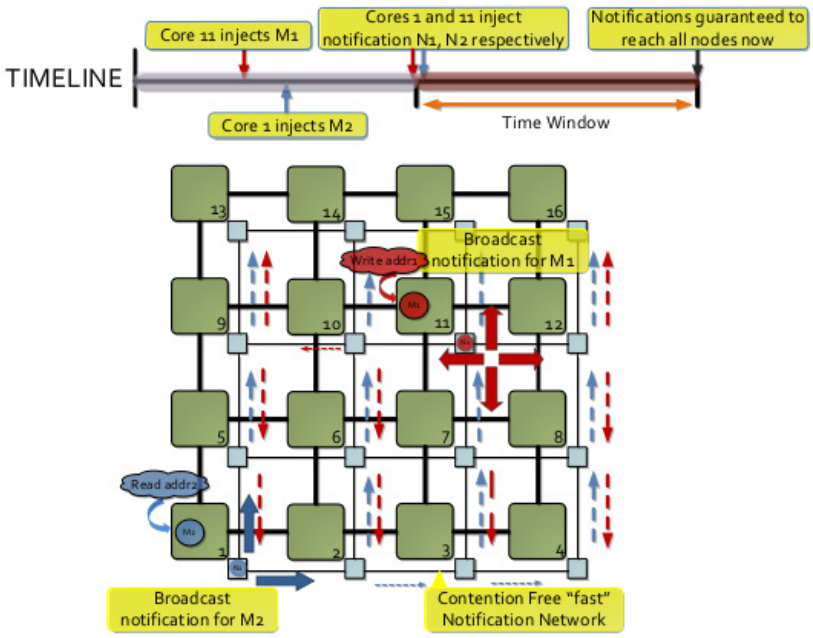


Figure 4.2: **Cores 11 and 1 inject M 1 and M 2 immediately into the main network. The corresponding notifications N 1 and N 2 are injected into the notification network at the start of the next time window**

- Core 11 miss triggers a request Write addr1 to be sent to its cache controller, which leads to message M 1 being injected into the network through the NIC. The NIC takes a request from the cache controller and encapsulates it into a single-flit packet and injects the packet into the attached router in the

main network. Corresponding to M 1, notification message N 1 is generated. However notification messages are injected into the notification network only at the beginning of every time window. Therefore notification message N 1 is sent out at the start of the next time window.

- Similarly, Core 1 miss triggers request Read addr2 to be sent to its cache controller. This leads to message M 2, being injected by the NIC into the main network. The corresponding notification message N 2 is generated, and injected at the start of the next time window as shown in figure 4.2**Cores 11 and 1 inject M 1 and M 2 immediately into the main network. The corresponding notifications N 1 and N 2 are injected into the notification network at the start of the next time window**figure.4.2.

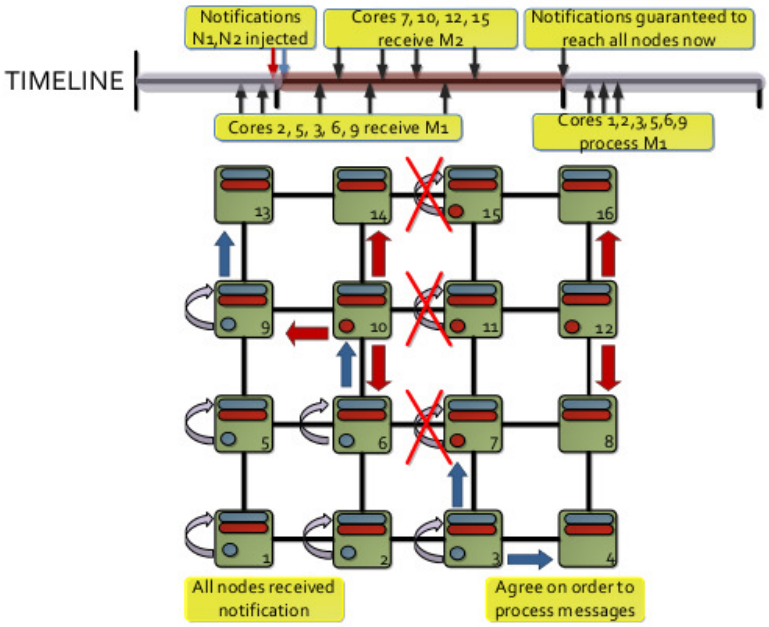
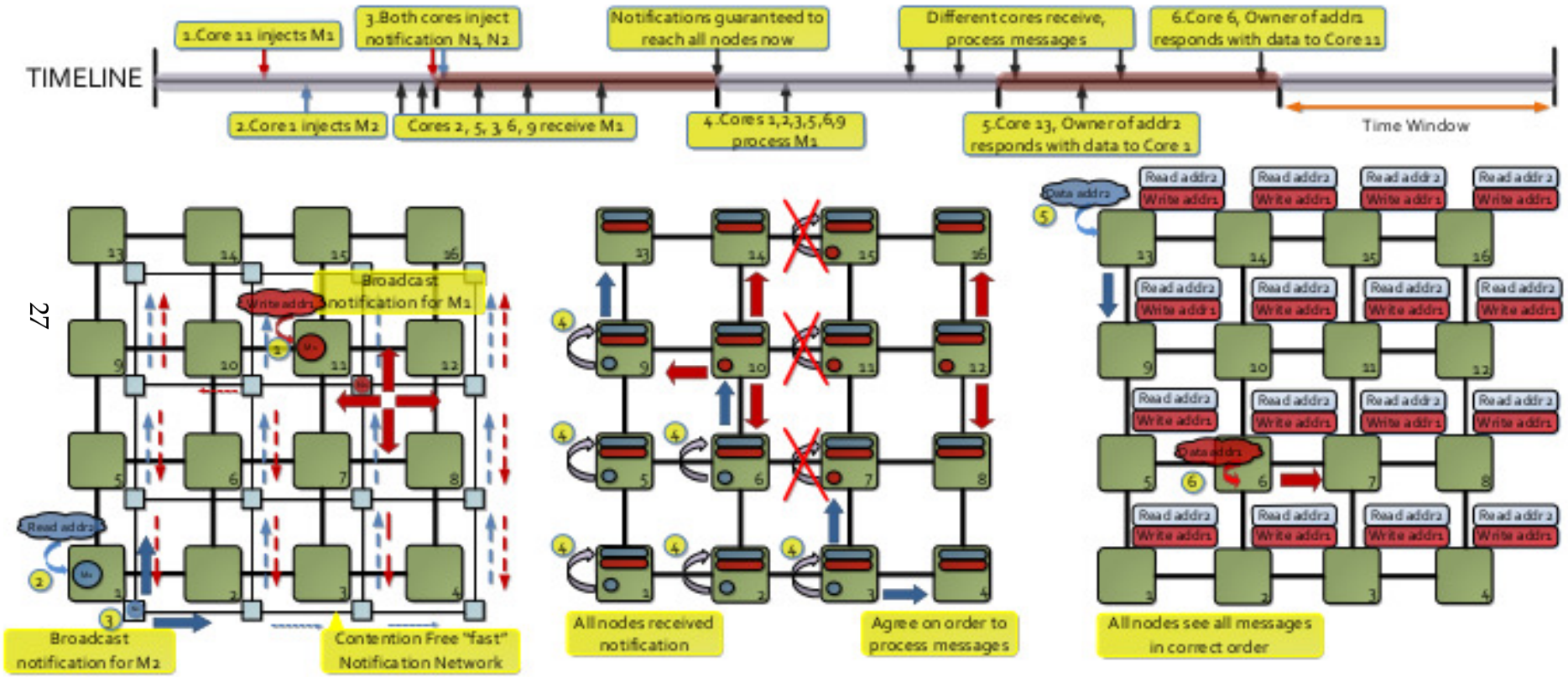


Figure 4.3: **All nodes agree on the order of the messages viz. M 2 followed by M 1. Nodes that have received M 2 promptly process it, others wait for M 2 to arrive and hold M 1**


TIMELINE
1.Core 11 injects M1
3.Both cores inject notification N1, N2
Notifications guaranteed to reach all nodes now
Different cores receive, process messages
6.Core 6, Owner of addr1 responds with data to Core 11
2.Core1 injects M2
Cores 2, 5, 3, 6, 9 receive M1
4.Cores 1,2,3,5,6,9 process M1
5.Core 13, Owner of addr2 responds with data to Core 1
Time Window
Broadcast notification for M1
Write addr1
Read addr2
Broadcast notification for M2
Contention Free "fast" Notification Network
All nodes received notification
Agree on order to process messages
Data addr2
Read addr2
Write addr1
All nodes see all messages in correct order

- Messages M 1 and M 2 make their way through the main network, and are delivered to all nodes in the network. However, until the corresponding notifications are received and processed by the nodes, these messages have not been assigned a global order. Hence, they are held in the NIC or routers of the destination nodes.
- At the same time, notifications N 1 and N 2 make their way through the 'fast' notification network, and are delivered to all the nodes in the network.
- At the end of the time-window, we guarantee that all nodes have received any notifications sent during this time window from any node; in this case N 1 and N 2 have been received by all nodes in the system. At this instant, all nodes know that they need to process messages from nodes 1 and 11. They take a local decision on how to order these messages. In this case, the rule is 'increasing order of SID(Source ID)'. Thus all nodes agree to process M 2 (from node 1) before M 1 (from node 11).
- If a node has already received M 2 then it may process the message M 2 immediately, and subsequently if it has received M 1 it may process that too. If a node has received neither M 1 nor M 2, then it waits for M 2. If a node has received only M 1, then it holds the same in the NIC (or the router, depending on availability of buffers) and waits for M 2 to arrive. The mechanism for the same is as follows. Each node that represents the source of the next message it must service. When a message arrives on the main network, the NIC performs a check on its SID field. If it matches the ESID, then the message is processed; else it is held in the buffers. Once the message with the SID equal to ESID is processed, the ESID is updated to the next value.
- Eventually, all the nodes receive M 1 and M 2 and process them in the agreed order, namely, M 2 followed by M 1.

  Figure 4.4**Walkthrough example with full timeline of events**figure.4.4shows the complete timeline of events in the system.

4.2 IMPLEMENTATION DETAILS

4.2.1 ROUTER MICROARCHITECTURE

Fig 4.5**Router Microarchitecture**figure.4.5 shows the micro architecture of the three stage main network router. During the first pipeline stage, the incoming flit is buffered (BW) and in parallel arbitrates with the other virtual channels (VC's) at

that input port for access to the crossbar's input port (SA-I). In the second stage, the winners of SA-I from each input port arbitrate for the crossbar's output ports (SA-O), and in parallel select a VC from a queue of free VC's. In the final stage, the winners of SA-O traverse the crossbar (ST). Next, the flit traverse the link to the adjacent router in the following cycle.

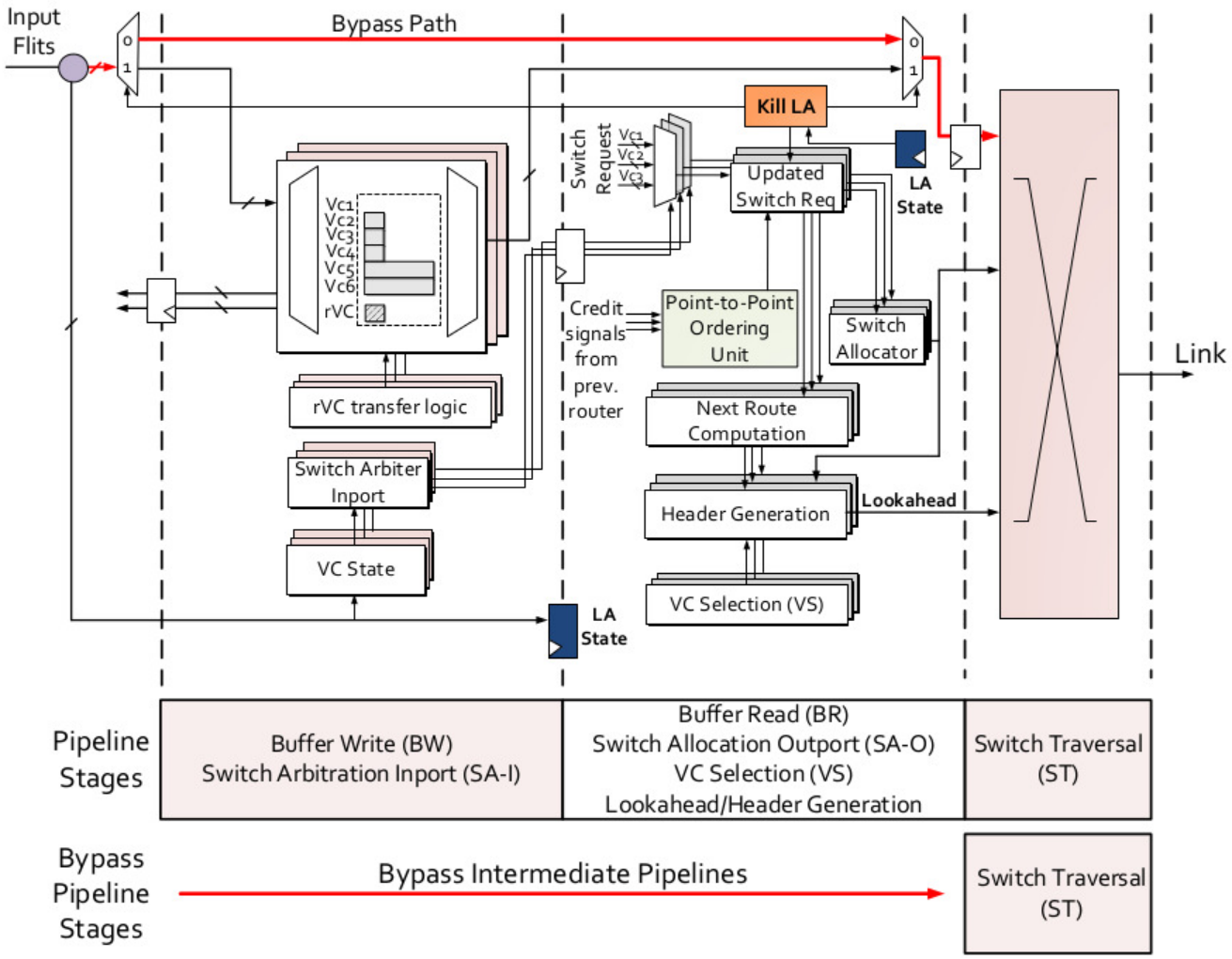


Figure 4.5: **Router Microarchitecture**

To reduce the network latency and buffer read/write power, Lookahead (LA) bypassing is implemented. A lookahead containing control information for a flit is sent to the next router during that flit's ST stage. At the next router, the lookahead performs route computation and tries to pre allocate the cross bar for

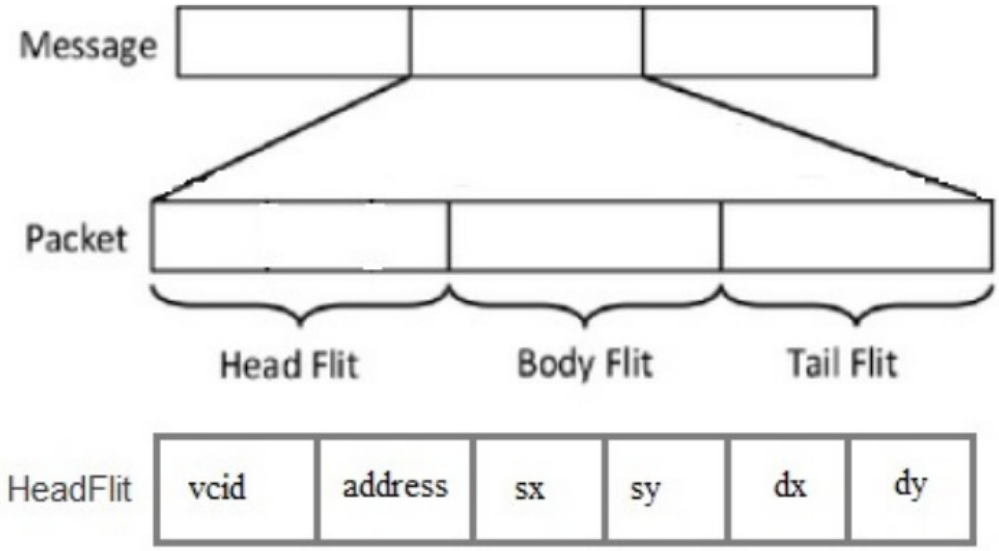


Figure 4.6: **Breaking Down the Message into Flits**

the approaching flit. Lookaheads are prioritized over buffered flits - they attempt to win SA-I and SA-O, obtain a free VC at the next router, and setup the cross bar for the approaching flits, which then bypass the first two stages and move to ST stage directly. Conflicts between lookaheads from different input ports are resolved using a static, rotating priority scheme. If a lookahead is unable to setup the crossbar, or obtain a free VC at the next router, the incoming flit is buffered and goes through all three stages. The control information carried by lookaheads is already included in the header field of conventional NoCs - destination coordinates, VC ID and the outport ID - and hence does not impose any wiring overhead.

Since we are using mesh topology, the router has five input and output ports corresponding to the four neighbouring directions North, South, East and West and a local port. The messages are broken down into packets which are further broken down into flits. This is schematically shown in Fig 4.6**Breaking Down the Message into Flits**figure.4.6 In this implementation we have a head flit, four body flits and a tail flit which indicates the end of the packet. Head flit contains

the virtual channel number, address required for matching and the source and destination information. Body and tail flits contains data.

The various stages that the flits pass through are shown in the figure 4.5**Router Microarchitecture**figure.4.5 and they are as follows:

- **Buffer Write:** The incoming flits are first written into the input buffers.
- **Route Computation:** Based on the source and destination information available on the head flits, route com- putation is performed in order to determine the output port for the flit. We have used the classic XY routing algorithm which is a distributed deterministic routing algorithm. It is a minimal algorithm suitable for mesh and torus topologies. XY routing algorithm routes packets first in x-direction (or horizontal direction) and then in Y- direction (or vertical direction) to the destination. The addresses of the routers are their xy- coordinates. It has the advantage of never running into a deadlock or live lock.
- **VC Allocation:** In this stage the flit arbitrates for a virtual-channel corresponding to its output port. In order to perform VC allocation we make use of VC credits that indicate whether a VC is free or empty. If the credit is set to 1 it shows that the VC corresponding to it is free and if it is set to 0 then it indicates that the VC is filled. As flits travel downstream, credits flow upstream to grant access to the buffers that are vacated.
- **Switch Allocation:** In this stages the flit arbitrates for access to the crossbar switch. Here we make use of a direction arbiter in order to allocate the resources.
- **Switch Traversal:** On successfully winning the required output port, the flit traverses the crossbar.
- **Link Traversal:** The flit finally traverses the link to the next node.

Each flit arrives at the input of the router. The input unit contains the buffers in which the flits are held. The head flit includes a field called vcid which is the virtual channel number. Depending on the vcid the flit is stored in the corresponding VC. The status of each of the virtual channels are maintained using two state fields which are Global (G) and Route (R). Global field can take on any of the values Idle(I), Rout- ing(R), waiting for next VC (V) or Active (A). Route indicates the

output port for the flit as obtained from route computation and can take any of the values North (N), South (S), East (E) or West (W).

Only the head flit passes through route computation and VC allocation. Body flits and tail flit follow the route setup by the head flit.

4.2.2 NOTIFICATION NETWORK MICRO ARCHITECTURE

The notification network is an ultra light weight bufferless mesh network consisting of 5N bit wise OR gates and 5N bit latches at each router as well as N bit links connecting these routers as shown in Fig 4.7**Notification Router Microarchitecture**figure.4.7, where N is the number of cores. A notification message is encoded as a N bit vector where each bit indicates whether a core has sent a coherence request that needs to be ordered. With this encoding, the notification router can merge two notification messages via a bit wise OR of two messages and forward the merged message to the next router. At the beginning of time window, a core that wants to send notification message asserts its associated bit in the bit vector and sends the bit vector to its notification router. Every cycle, each notification router merges received notification messages and forwards the updated message to all its neighbor routers in the same cycle. Since messages are merged upon contention, messages can always proceed through the network without being stopped, and hence, no buffer is required and network latency is bounded. At the end of that time window, it is guaranteed that all nodes in the network receive the same merged message, and this message is sent to the NIC for processing to determine the global order of the corresponding coherence requests in the main network.

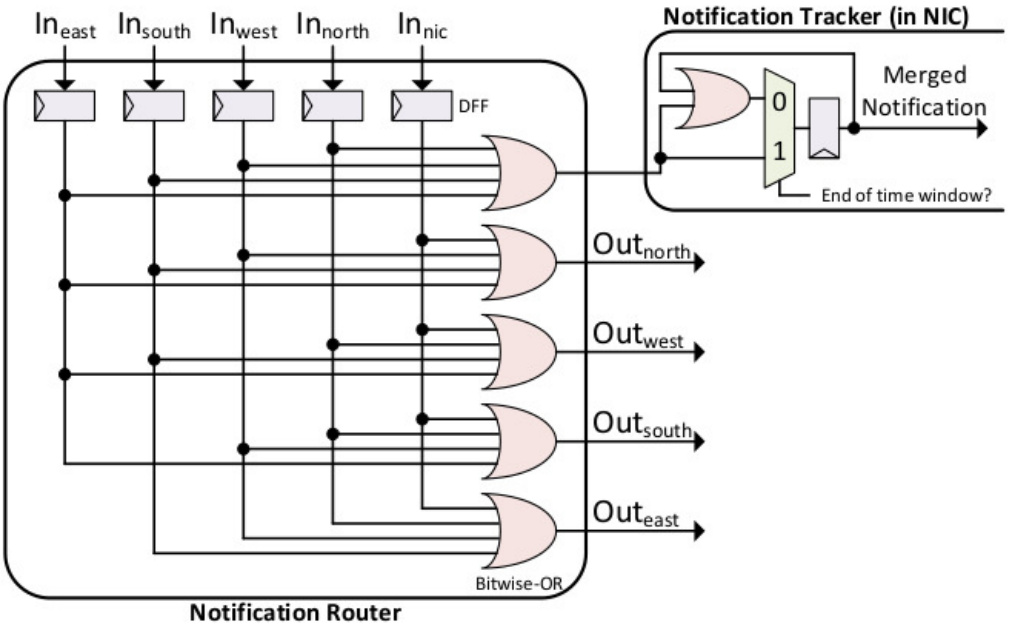


Figure 4.7: **Notification Router Microarchitecture**

4.3 THE ROUTER PIPELINE

Each head flit should go through route computation, virtual-channel allocation, switch allocation, and switch traversal. Each of these steps takes one clock cycle. Suppose the head flit of a packet arrives at the router during cycle 0. The packet is directed to a particular virtual channel of the input port, as indicated by the vcid field of the head flit. At this point, the global state (G) of the target virtual channel is in the idle state(G = I). The arrival of the head flit causes the virtual channel to transition to the routing state (G = R) at the start of cycle 1.

During cycle 1, information from the head flit is used in order to perform route compu- tation and to select one of the output port. The result of this computation updates the route field (R) of the virtual channel state vector and advances the virtual channel to the waiting for output virtual channel state(G = V) , both at the start of cycle 2. In parallel with this, the first body flit arrives at the router.

During cycle 2, the head flit enters the VC allocation stage and also the second body flit arrives at the router. During this stage, the result of route computation, held in the R field of the virtual channel state vector, is the input to the virtual channel allocator. If successful, the allocator assigns a virtual channel on the next router specified by the R field. The result of VC allocation updates the virtual channel id (vcid) of the flit and transitions the virtual channel to the active state(G = A).

At the start of cycle 3 switch allocation begins. The head flit starts this process, but is handled no differently than any other flit. In this stage, any active virtual channel (G = A) that contains buffered flits and has downstream buffers available (C greater than 0) bids for a single flit time slot through the switch from its input

virtual channel to the output unit. Depending on the configuration of the switch, this allocation may involve competition not only for the output port and the switch, but also competition with other virtual channels in the same input unit for the input port of the switch.

If successful switch allocation occurs during cycle 3 then the head flit traverses the switch during cycle 4. The head flit can start traversing the channel to the next router in cycle 5 without competition. While the head flit is being allocated a virtual channel, the first body flit is in the route computation stage, and while the head flit is in the switch allocation stage, the first body flit is in the VC allocation stage and the second body flit is in the route computation stage. However, since routing and virtual -channel allocation are per -packet functions, there is nothing for a body flit to do in these stages. Body flits cannot bypass these stages and advance directly to the switch allocation stage because they must remain in order and behind the head flit. Thus, body flits are simply stored in the virtual channel's input buffer from the time they arrive to the time they reach the SA stage.

As each body flit reaches the switch allocation stage, it bids for the switch and output channel in the same manner as the head flit. The processing here is per flit and the body flits are treated no differently than the head flit. As a switch time slot is allocated to each flit, the credits are updated to reflect its departure.

Finally, during cycle 6, the tail flit reaches the switch allocation stage. Allocation for the tail flit is performed in the same manner as that for head and body flits. When the tail flit is successfully scheduled in cycle 6, the packet releases the virtual channel at the start of cycle 7 by setting the virtual channel state for the output virtual channel to idle (G=I) and the input virtual channel state to idle(G=I) if the input is empty. If the input buffer is not empty, the head flit for the next packet is

already waiting in the buffer. In this case, the state transitions directly to routing (G=R).

Each virtual-channel contains a buffer for head flit alone and a separate buffer for body flits and tail flit.

Vcstate represents the state of each VC and includes the G and R fields. To see if a packet is processed completely from the head flit to the tail flit a variable vacancy is used for virtual channels in each direction and it is updated with values 'Empty', 'Filling' or 'Filled'. When the VC is empty it holds the value 'Empty'. As head flit arrives the value is updated to 'Filling' and it remains so as the body flits get filled in. Once the tail flit arrives the value is updated to 'Filled' indicating that the entire packet has arrived.

CHAPTER 5

IMPLEMENTATION AND VERIFICATION OF MESH FABRIC

5.1 CHOICE OF LANGUAGE

C++, System C and System Verilog are some of the platforms that have been used for the design and synthesis of hardware for some years now. With the emergence of Bluespec System Verilog (BSV), more and more designers are opting Bluespec as their preferred platform for hardware design. The reasons for this are as follows:

- BSV semantics like interfaces and atomic rules are at high level of abstraction for concurrency and hence it is much preferred to describe the parallelism found in hardware systems.
- BSV's powerful types and static elaboration provides ease to describe the hardware architecture. The static elaboration is deterministic which helps to do a high level synthesis from BSV and arrive at a high quality of design much faster than the other platforms.
- BSV provides strong parameterization due to which all modules and functions can be parameterized by other modules and functions. This gives a higher level of expressive power to designer.
- Stronger type checking allows greater expansiveness with greater safety.
- The code written using all the features is still synthesizable which really sets BSV apart from other languages. The constructs and features as described above are either not available with other languages or are provided as add-ons and hence are not equally powerful. Hence the design has been implemented in BSV.

5.2 COHERENCE REQUESTS AND RESPONSES

We need to implement an interconnect fabric that can be used to employ snoopy coherence in a mesh network. Therefore in the network we deal with two types of messages which are request messages and response messages. Each is treated differently and separate algorithms are used for both the types of messages. Requests need to be broadcasted throughout the network so that all the nodes can see it. But responses have to be routed correctly from the source node towards the destination node. As such, it is to be noted that in the case of request messages, VC allocation is performed directly after the buffer write stage thereby skipping the route computation stage in between.

Usually, in an interconnect network, global message ordering is realized by making use of a centralized ordering point. But such a centralized ordering point introduces two types of latency in the network:

- **Indirection:** This is the latency incurred by the message as it travels from the source node to the ordering point.
- **Serialization Latency:** This is the latency of the message that is waiting at the ordering point before it is correctly ordered and forwarded to the other nodes.

In order to eliminate this dependence on a centralized ordering point, we make use of two separate networks in our implementation.

- **The Main Network:** The main network is responsible for broadcasting coherence requests to all the nodes and delivering responses to the requesting node. The network is unordered and the requests from several source nodes may arrive at a particular node in any order. At every node in the main network we have 16 FIFOs which store the requests coming from different source nodes. For example, the requests coming from node 0 will always go to FIFO 0, requests coming from node 2 will be stored in FIFO 2 etc. The routers are the basic elements of the main network. The router ports are connected together using mkConnection in order to create a mesh type

network as shown. The main interface Ifcrouter mesh has flit input and output methods and the interface Ifc credit has the upstream and downstream availability of credits for all the directions. Consider a 4 by 4 mesh network consisting of 16 routers. The routers are connected together as shown in figure 5.1**Broadcasting of request from the requesting node**figure.5.1 The routers are addressed by their x and y coordinates. Every router has input ports namely north in, south in, east in, west in and local in. Similarly the output ports are denoted by north out, south out, east out, west out and local out. Whenever a request originates at any of the nodes, it is send to the local out port from where it will be broadcasted to all the four directions. The broadcasting of requests is done throughout the network by following the given rules:

- If a request is received at the local port, it will be sent to all the four directions, namely north, south, east and west ports.
- If a request is received at the north port, it will be sent to the south port.
- If a request is received at the south port, it will be sent to the north port.
- If a request is received at the east port, it will be sent to north, south and west ports.
- If a request is received at the west port, it will be sent to north, south and east ports.

Thus all the nodes are able to receive the requests. In figure, broadcasting is shown by the black arrows. Here, node (1,1) is the requesting node that sends the request.

- **The Notification Network:** Ordering of the requests from several source nodes is achieved by making use of the notification network. It is a fast, buffer less network. Whenever a request is send from the main network, a notification message encoding the source node's id is broadcasted in the notification network. The notification message is essentially a bit vector, where each bit corresponds to request from a particular source. Therefore requests from several sources are merged together by OR-ing the bit vectors in a contention less manner. The notification network consists of 5 N-bit bitwise OR gates where N is the number of cores. From the notification message every node knows for which source's request it is waiting for. In an interconnect of 16 nodes, the highest priority is given to node 0 and the lowest priority to node 16.

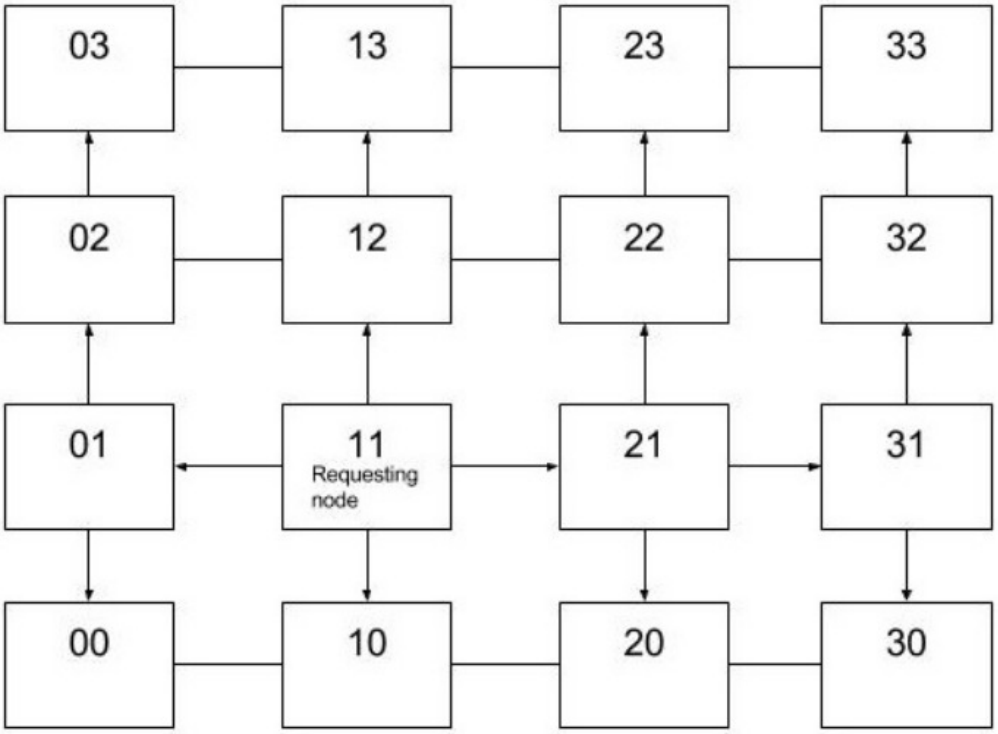


Figure 5.1: **Broadcasting of request from the requesting node**

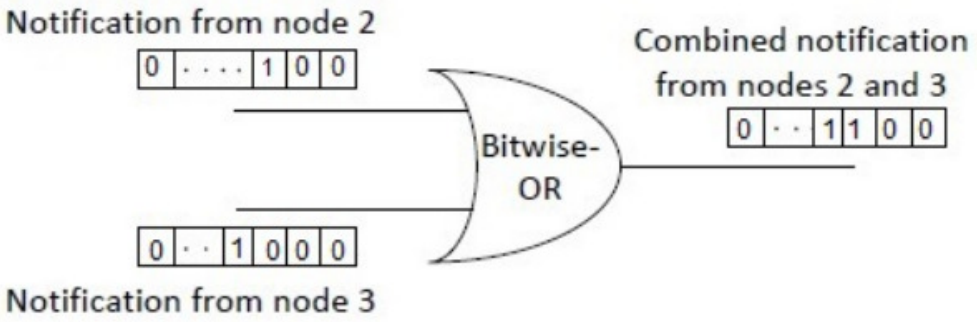


Figure 5.2: **OR-ing of notification messages from different sources**

When the request arrives at the node, the notification message is checked in order to determine the node whose request needs to be serviced first. Depending on this, the request message is dequeued from the corresponding FIFO in the main network. Thus even though requests from different source nodes arrive at a particular node in any order, it will be serviced at all the nodes in a consistent global order. Thus, all the nodes locally determine the global order for servicing the requests.

If address matching is successful and hit occurs, then a response is send from that particular node to the requesting node. The response follows a path along the network as determined by the XY routing algorithm. The flit is first routed along the x direction and then along the y direction till it reaches the destination node.

In order to perform the action of address matching, a tag register was introduced at every node. In one of the nodes the tag value was kept equal to the value of address that we are passing in the request flit. Everywhere else the tag value is equal to zero. Therefore when the request reaches that particular node with the corresponding tag value, hit occurs and response can be send back.

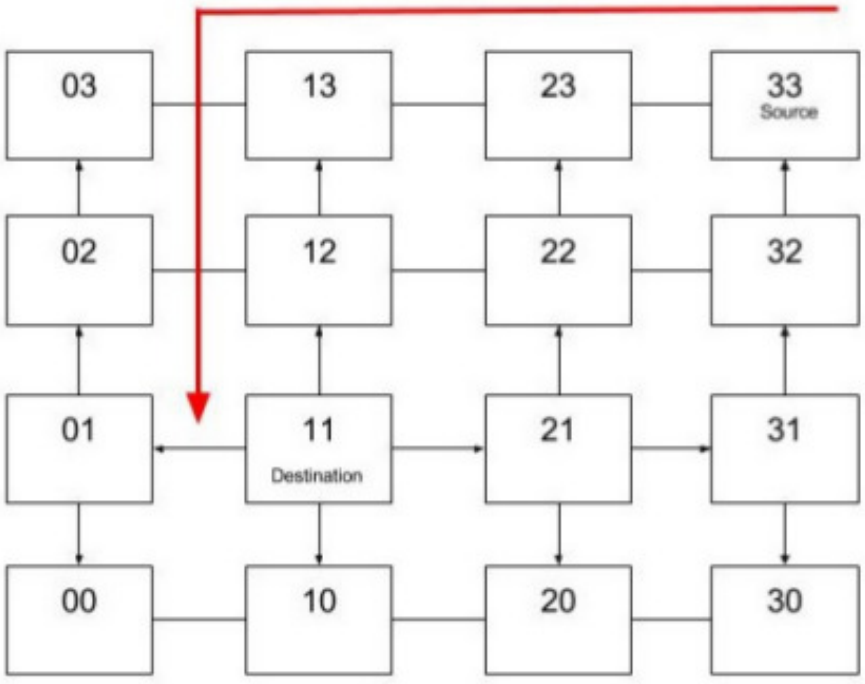


Figure 5.3: **Routing the request back to the requesting node**

5.3 VERIFICATION OF MESH FABRIC

Verification is the process of checking for the correctness of the fabric that is designed.It's the important step in design flow and is performed after every step to ensure the correctness of the design.To ensure if the fabric is performing its intended task we perform verification.The verification step is highly important during the design.Verification of the design helps to find if the fabric works in the correct way.It helps to identify if there's a bug in the design.There are three verification types namely:

- Design Verification
- Manufacture Verification
- Implementation Verification

Design verification is the verification done to check if the designed circuit is consistent with its intended functionality and the original specifications.Implementation verification checks if the implementation is consistent with the design. Manufacture verification is performed to check if the circuit that's fabricated is consistent with what the designer intends it to do.This is also known as testing which checks for the functionality of the fabricated integrated circuit and finds the fault of the fabricated ic.

In this project we perform design verification to check if the mesh fabric functions as it is expected to.In design verification the inputs are given to the fabric through the test bench and its checked if the corresponding response is generated or not.Two types of test benches are written to verify the functionality and to check if the fabric works fine with the core working on top of it.

The verification of mesh fabric is done by using two test benches.The first type

of test bench uses just the mesh fabric and the requests and the corresponding request flits are generated in the test bench.In the second test bench the fabric is connected to SHAKTI C-class core and the request flits are introduced in the main network when the core requests data from the memory.

5.3.1 TestBench without Core

This test bench verifies the proper functionality of the mesh fabric for two requests generated at consecutive clock cycles. In this part we first initialize each router with a tag value.Then we instantiate the network routers through the interface declared in the router module.Next we make the mesh network by connecting the in and out ports of the different routers.In this the east out port of router(i,j) is connected to west in port of the router(i+1,j).Then the north out port of router(i,j) is connected to south in port of router(i,j+1).A 4X4 mesh network is formed by connecting the routers as above.Then the notification network is instantiated through the interface and the mesh network is made by connecting the notification network routers in the same way as main network routers. Now we have the main network and notification network ready.

After making the connections we start a clock in the test bench which is a counter that counts 200 clock cycles.For verifying the network we generate a packet with single flit which is the head flit.This is a request type of head flit and it contains the information about vcid,address,x and y coordinates of the router which is sending out the request.Then the flit is introduced into the local out port of the corresponding router.Also a notification message is generated which is introduced in the notification network later.After the first request flit is sent to the local out port we generate another request flit at another router in the next clock cycle.This

request is tagged as a valid request and sent to the output port of the router and corresponding notification message is generated.Then the notification network is given a start signal which makes the notification message sent through the network.The request packets traverse through the main network and each router enqueues the request packet in a FIFO .

Once the notification message is received by every router,the request that's to be served first is decided based on the notification message. The priority is given for router(0,0) as highest and router(3,3) as the lowest.The notification message gives the id of the router whose request is being serviced.Each router will identify the request to which they have to respond to and they dequeue the corresponding request flit.The request flit is collected in the test bench through the local input port.Now each router checks if the flit is a valid head request. Then the address the packet holds is compared to the tag value assigned to each router. If there is a match a response flit is generated.First the head flit of the response packet is generated and then it's assigned a vcid and the destination is set as the router that has generated the request.The flit is sent out in the local out port.Then in the next clock cycles the four body flits are sent through the local port to the destination.Finally the tail flit is sent to the destination. After the response to the request if highest priority is served the routers check for the next request to be served.The routers again check the notification network and they get the id of the router whose request should be served.The routers again check the address with their corresponding tag value and if there's a match then the corresponding router will send the response packet to the destination.The body response flits and the tail response flits follow the head response flits.

In this way the test bench is written to test the mesh fabric and the response is

observed and presented in the results chapter.

5.3.2 Test Bench with Core

This test bench connects the core with mesh fabric that is a core is placed on top of the mesh fabric.The core used is the SHAKTI C-class core.It has the following variables set:

- RV64=True
- muldiv=True
- sequential=True
- atomic=True
- spfpu=True
- dpfpu=True
- bpu=True
- verbose=True
- MMU=True
- perf=True
- prefetch=True
- UART0=True
- UART1=True
- BOOTROM=True
- PLIC=True
- CLINT=True
- simulate=True

Also the following variables are set for the core for simulation purposes and also to get pre-processor macros:

- ISA=RV64IMAFD
- MMU=enable
- AXI=enable
- BPU=enable
- MUL=sequential
- PERF=enable
- VERBOSE=enable
- PREFETCH=enable
- DEBUG=disable
- OPENOCD=disable
- QSPI0=disable
- QSPI1=disable
- SDRAM=disable
- UART0=enable
- UART1=enable
- PLIC=enable
- BOOTROM=enable
- I2C0=disable
- I2C1=disable
- DMA=disable
- AXIEXP=disable
- TCM=disable
- CLINT=enable
- VME=disable
- SYNTH=SIM
- FLASHMODEL=cypress

Initially the core is instantiated through its interface.Then the router and the notification network routers are instantiated correspondingly.In this only two main network routers and two notification routers are used. The two routers are connected making a connection between the corresponding in and out ports. Also the notification network is made by making the connections.The core provides a transactor interface through which requests generated by core are received at the routers.The transactor interface which is of master type provides a write data channel, write response channel,read data channel and a read address channel.There's two transactor interfaces at the core where one has access to imem and one goes to the dmem.

Once the connections are made the core starts running instructions which are fetched from the boot files of the core.These instructions are run and when there is a request that needs to be sent to the dmem or imem the corresponding bits are set valid.If there's a valid request from the core it's identified by checking the valid bits in the read address channel or write address channel of the transactor.Also the type of memory request sent is noted that is if its a dmem request or an imem request .When the output of the port is read as valid all the parameters sent out by the core like the address,burst,length e.t.c are read and stored in registers.

Once the valid bit is received we generate a head flit request which sends out the address that the core sent to the mesh fabric.In this the flit holds the address and also the coordinates of the source router that received a request from the core.This head flit is broadcast and since we have only two routers here the flit is just forwarded to the other router.

Now the second node checks its address range for which it can get the data from and if the address sent by the core requesting for data is a match it will give

out a hit.The node then starts sending the response flits to the node that requested it.The body flits enqueue the data from the required memory location and they are sent back to the core.The test bench is stopped once the packet is sent back to the core.

The first test bench has helped to identify a bug in the virtual channel allocator module.

The results of both the test benches are presented in the next chapter.

CHAPTER 6

SIMULATION RESULTS

6.1 HARDWARE DESIGN FLOW

Once the design has been coded in high level language, it needs to be realised into actual physical hardware. The VLSI design flow as depicted in Figure 6.1**Flow of Synthesis**figure.6.1 outlines the major steps in converting the design towards physical realization.

The design of any product starts with an idea. The idea is born out of a client require- ment. This idea is put down as a higher level behavioural model of the final product using the high level languages like BSV. The behavioural model is then compiled into a RTL using a suitable compiler. RTL is generally the description of the circuit at the module level where input output interfaces, clock and other signals are visible. Any design can be described in RTL using the Huffman's model.

Once the RTL is arrived at, the next step in the design flow is the logic synthesis. Using commercial EDA tools, the designer converts the RTL into a netlist which is nothing but a list of gates and wires whose inputs and outputs are specified. The EDA tools gives a lot of options like types of gates to be used, constraints for the design with respect to the power, area and timing, thus a highly optimized netlist is achieved after logic synthesis.

On getting the netlist, more EDA tools are used to do place and route of

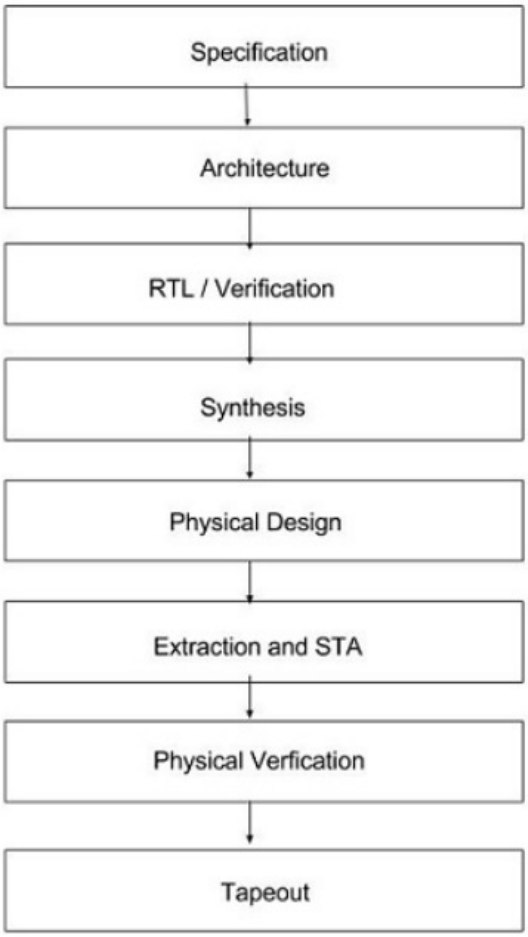


Figure 6.1: **Flow of Synthesis**

gates and wires or floor planning as it is popularly called. The result of place and route is the mask that could be handed over to the foundry for carrying out the fabrication of the chip. Two most important part of the design flow are the testing and verification. Testing is done to ensure final chip does not suffer from manufacturing defects and verification is done at each stage of the flow to ensure the design meets the requirements as were originally projected. In our case however, we limit the scope to design, implementation of the design in BSV, simulations and logic synthesis is done to verify performance.

6.2 SIMULATION RESULTS FOR TEST BENCH ONE

Once the BSV coding is done, the project is compiled with BSV compiler, which gives options to compile for BSV simulator or to generate verilog files for further processing. We simulated the design using the Bluesim simulator. The results of the simulations are observed on the BSV GUI and recorded for analysis. The simulation results for the first test bench for some of the test cases are shown below.

A test case core 0 send out a request with address value equal to 2 and core 5 send out a request with address value equal to 3 . The tag value in core 14, i.e., (2,3) is kept equal to 2 and tag value in core 10, i.e., (2,2) is kept equal to 3 so that when address matching is done hit occurs at these nodes and a response signal is sent back to core 0, i.e., core (0,0) and core 5, i.e., core (1,1) respectively.

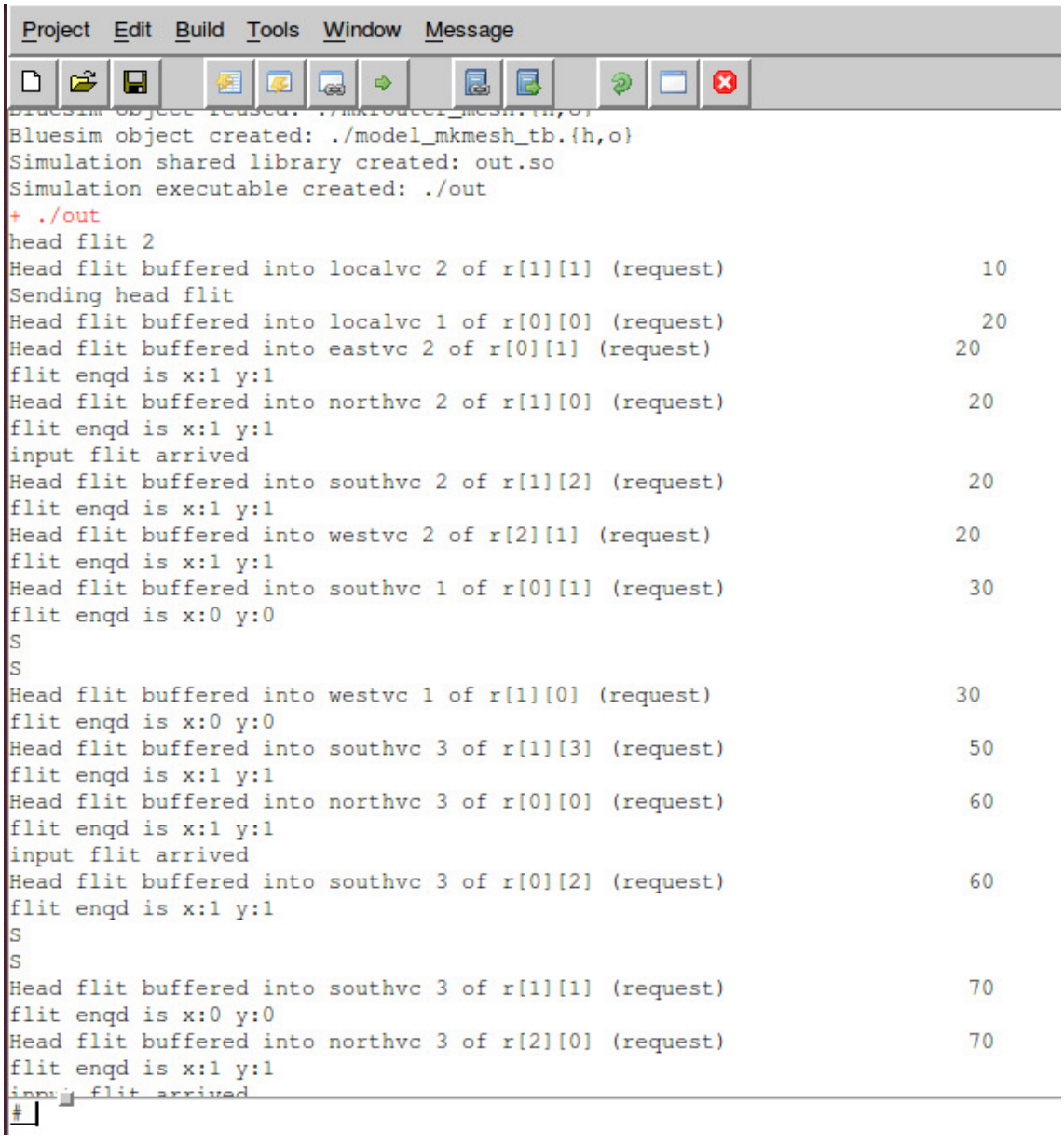


Figure 6.2: **BSV Simulation result for Test Case 1**

In figure 6.2**BSV Simulation result for Test Case 1**figure.6.2 we can see that the head flit, four body flits and tail flit are sent on to the local out port of node (0,0) and (1,1) and from there it passes on to the adjoining nodes by following the rules that were laid down for broadcasting.

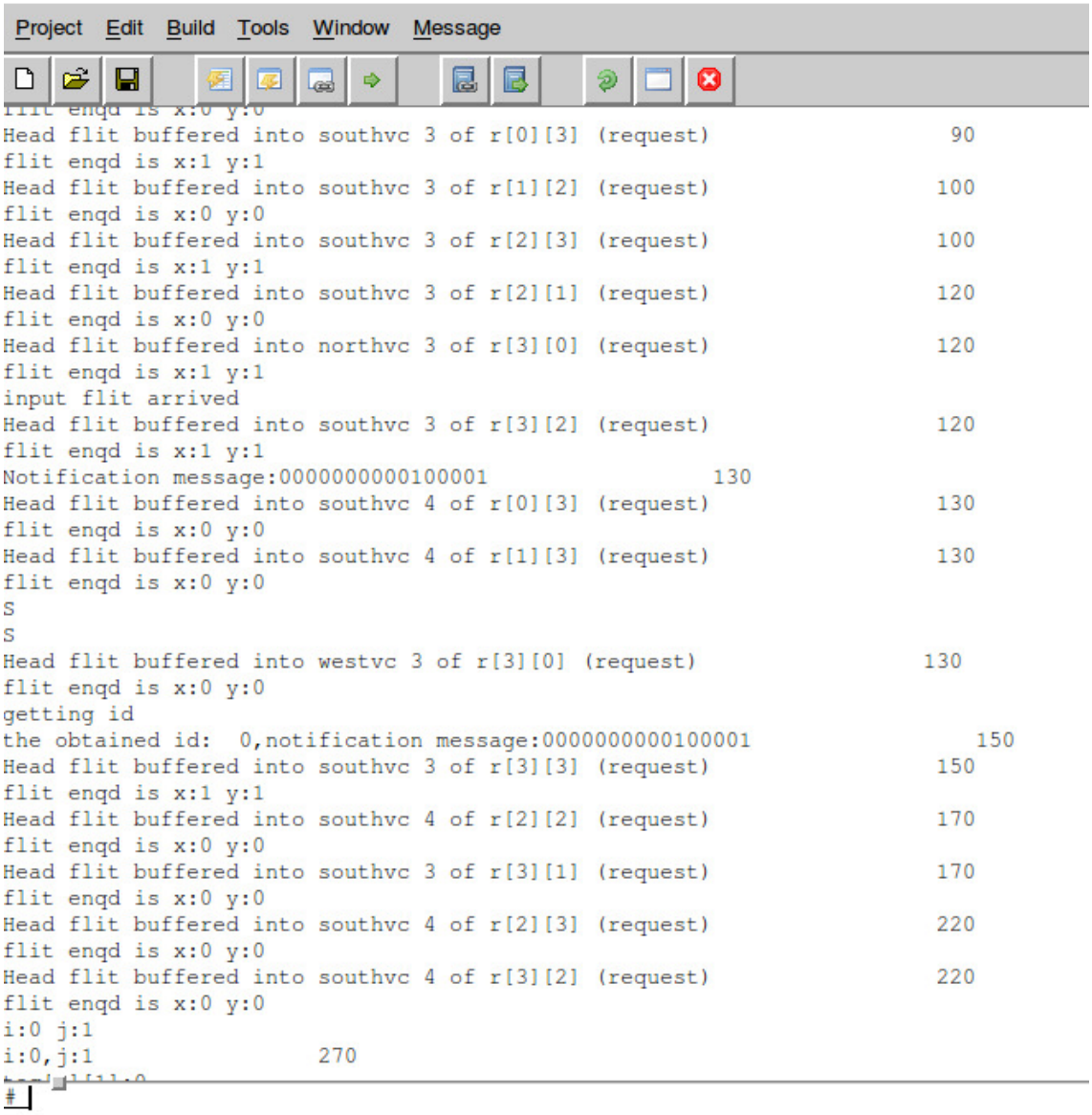


Figure 6.3: **BSV Simulation result for Test Case 1 Contd'**

In figure 6.3**BSV Simulation result for Test Case 1 Contd'**figure.6.3 it can be seen that by this time the notification message is obtained from the notification network. The notification message is 0000000000100001 showing that a request

from node (0,0) and (1,1) is in flight and all the nodes are waiting for it.

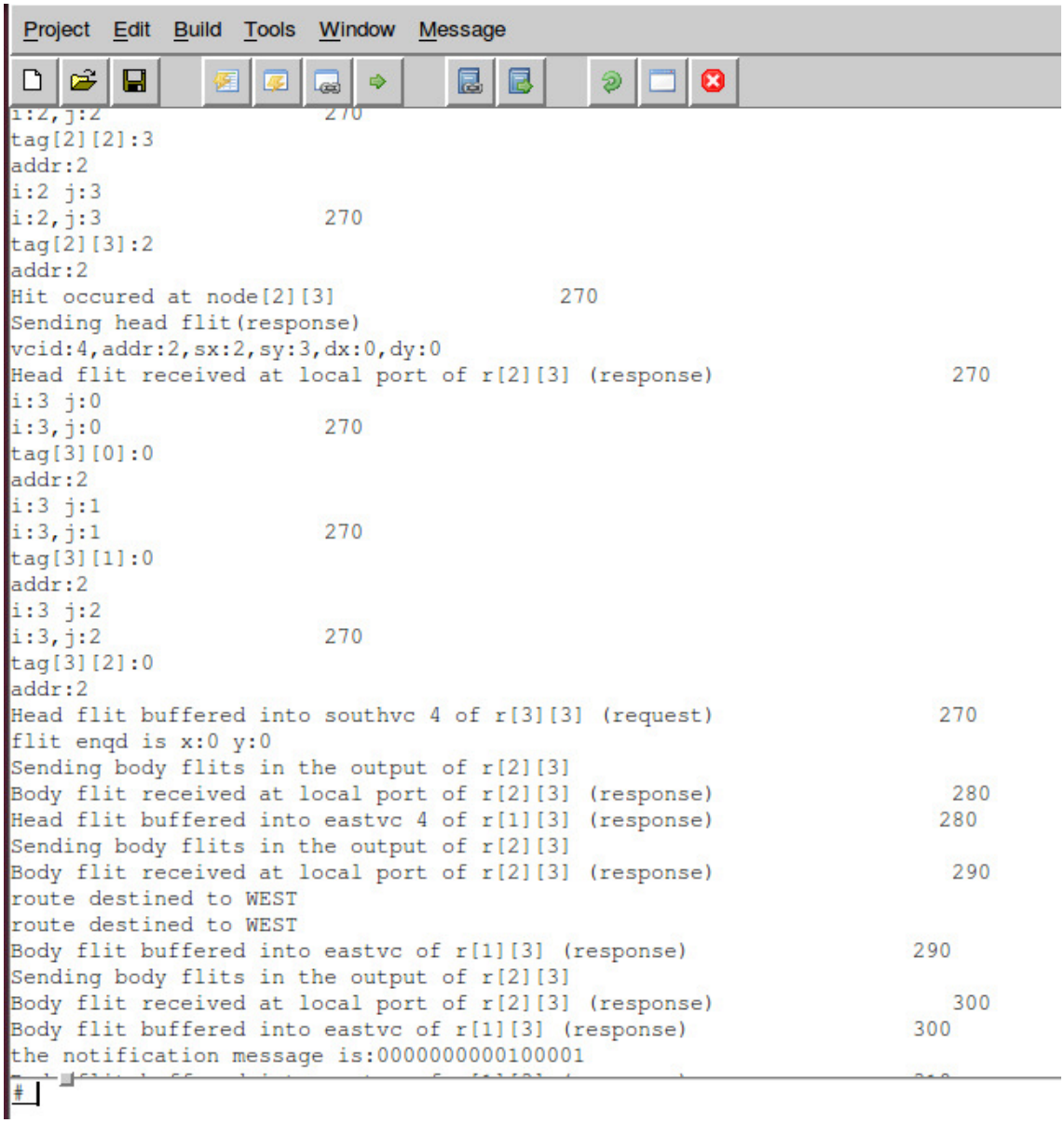


Figure 6.4: **BSV Simulation result for Test Case 1 Contd'**

In figure 6.4**BSV Simulation result for Test Case 1 Contd'**figure.6.4 it can be seen that the tag value and the address matches at node (2,3). Therefore, response is sent back from node (2,3).

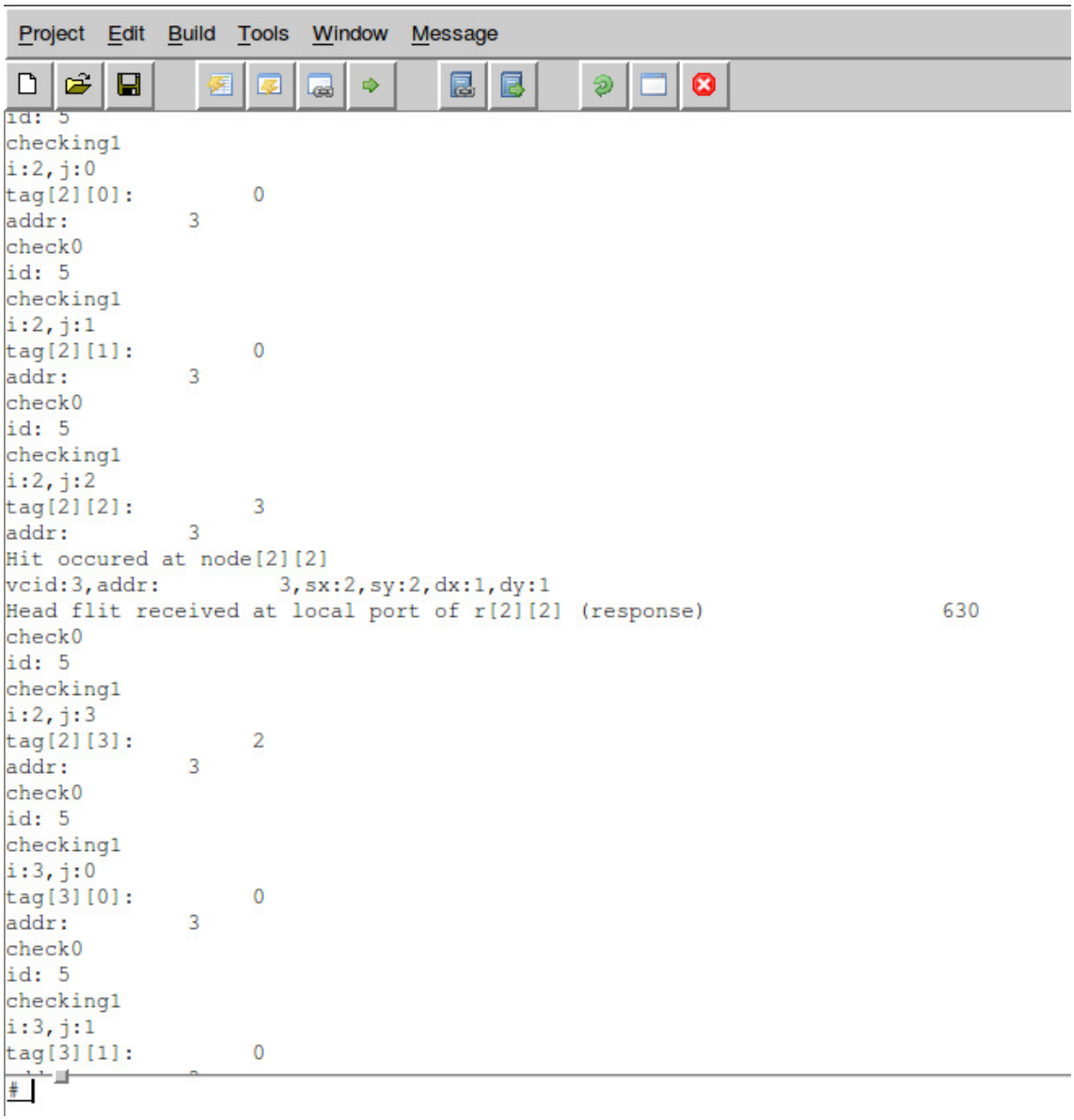


Figure 6.5: **BSV Simulation result for Test Case 1 Contd'**

In figure 6.5**BSV Simulation result for Test Case 1 Contd'**figure.6.5 it can be seen that the tag value and the address matches at node (2,2). Therefore, response is sent back from node (2,2).

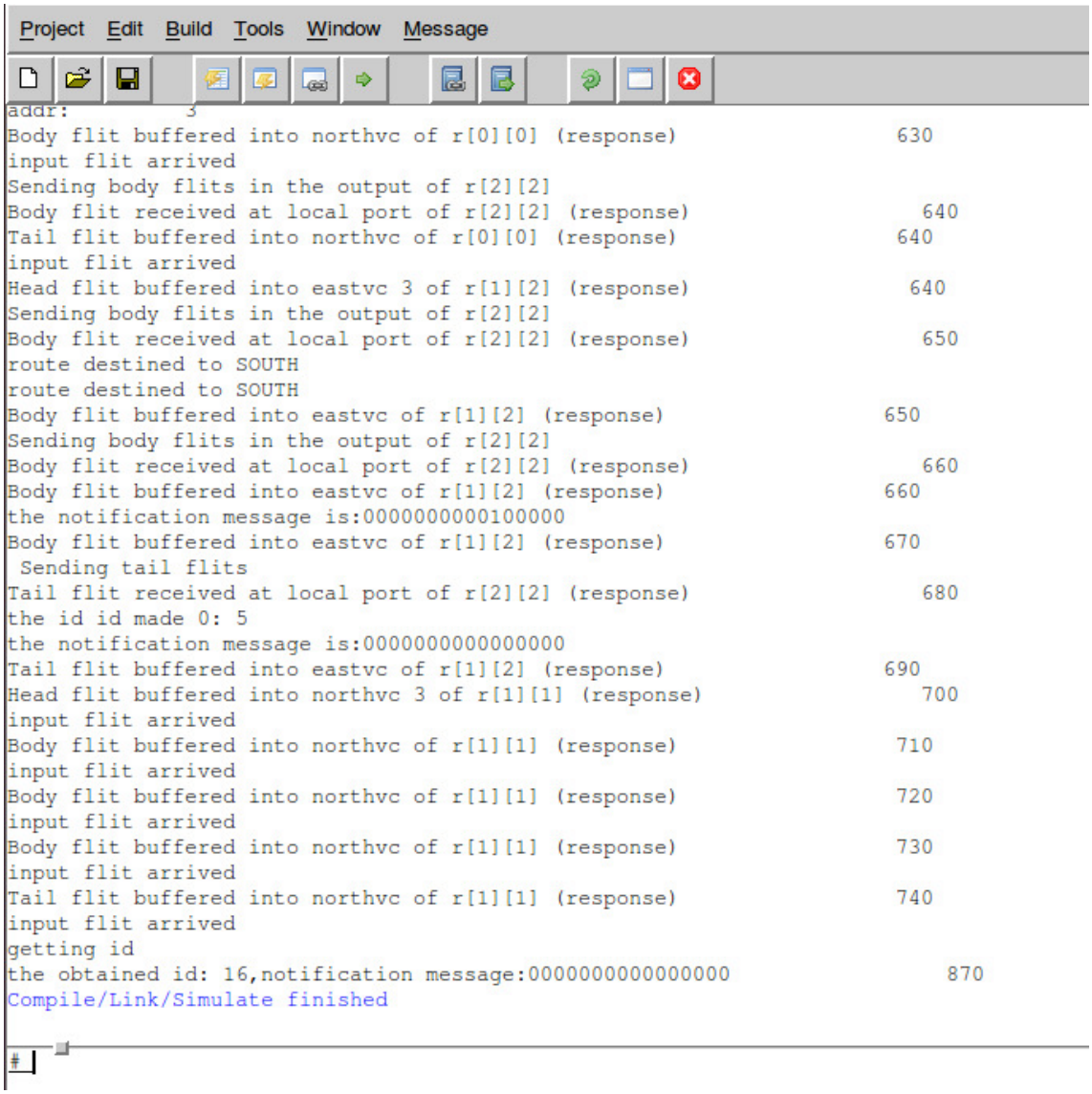


Figure 6.6: **BSV Simulation result for Test Case 1 Contd'**

The response is routed along the network according to the path given out by the XY routing algorithm. It is first routed along X direction and then along Y direction to finally reach the destination as shown in figure 6.6**BSV Simulation result for Test Case 1 Contd'**figure.6.6

6.3 SIMULATION RESULTS FOR TEST BENCH 2

In the second test bench we send out the request packet to router(0,0) from the core.The router(1,0) is set to hold the address the core requests for.So we send out the flits to the router(0,0) which broadcasts it to the other router.

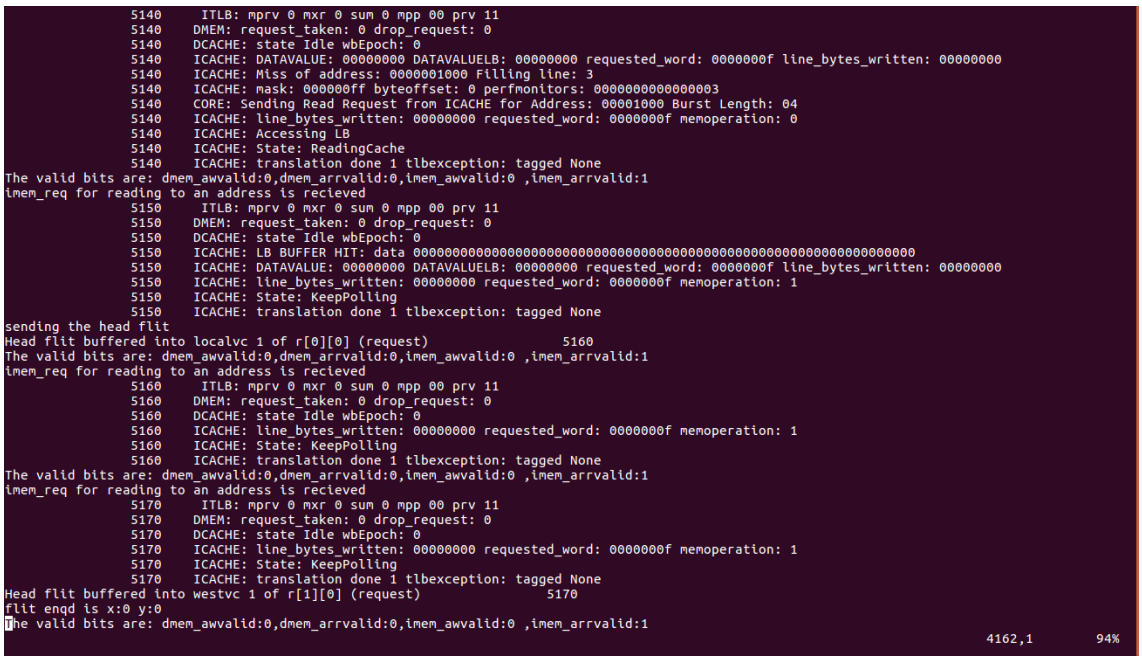


Figure 6.7: **BSV Simulation for Test bench 2**

In the above figure6.7**BSV Simulation for Test bench 2**figure.6.7 we can see the core executing instructions and it sends out a request to read from a memory location and the read channel valid bit is made True.Then in the next clock the head flit is sent out into the node(0,0) and its buffered into the next node. The following images show the sequence of events after the packet is introduced.

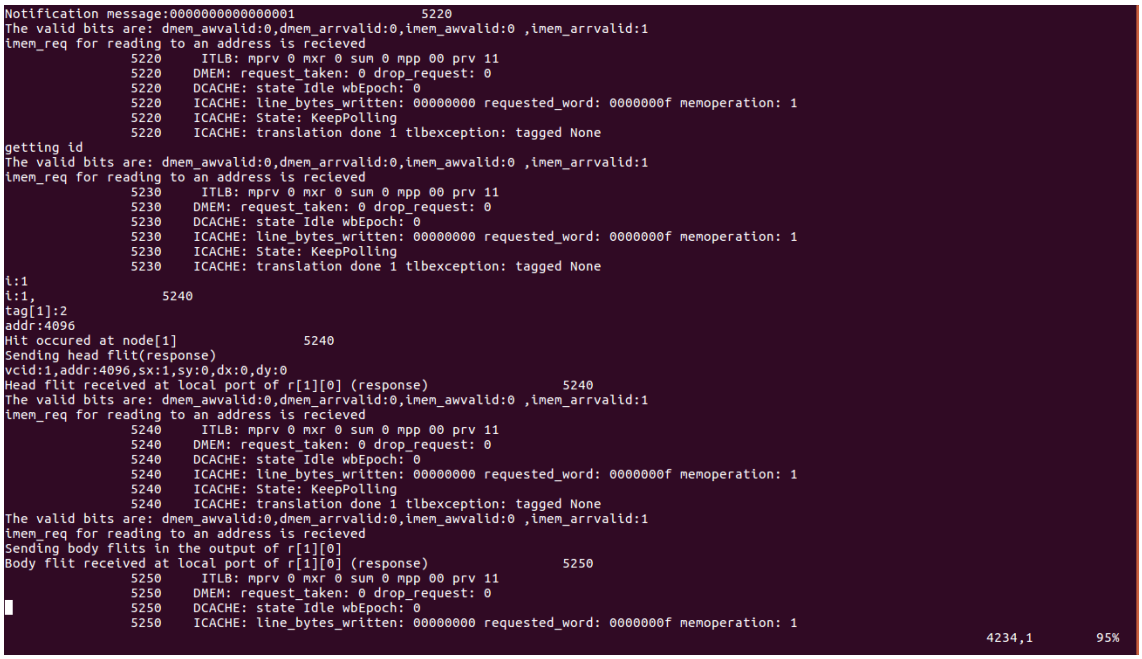


Figure 6.8: **BSV Simulation for Test bench 2**

In figure6.8**BSV Simulation for Test bench 2**figure.6.8 its seen that the notification message is generated and sent to the notification network and the id of the request to be generated is obtained. Then the nodes check for the address sent out by the core and there is a hit shown in the above figure6.8**BSV Simulation for Test bench 2**figure.6.8 and the head flit response and body flits are sent back.

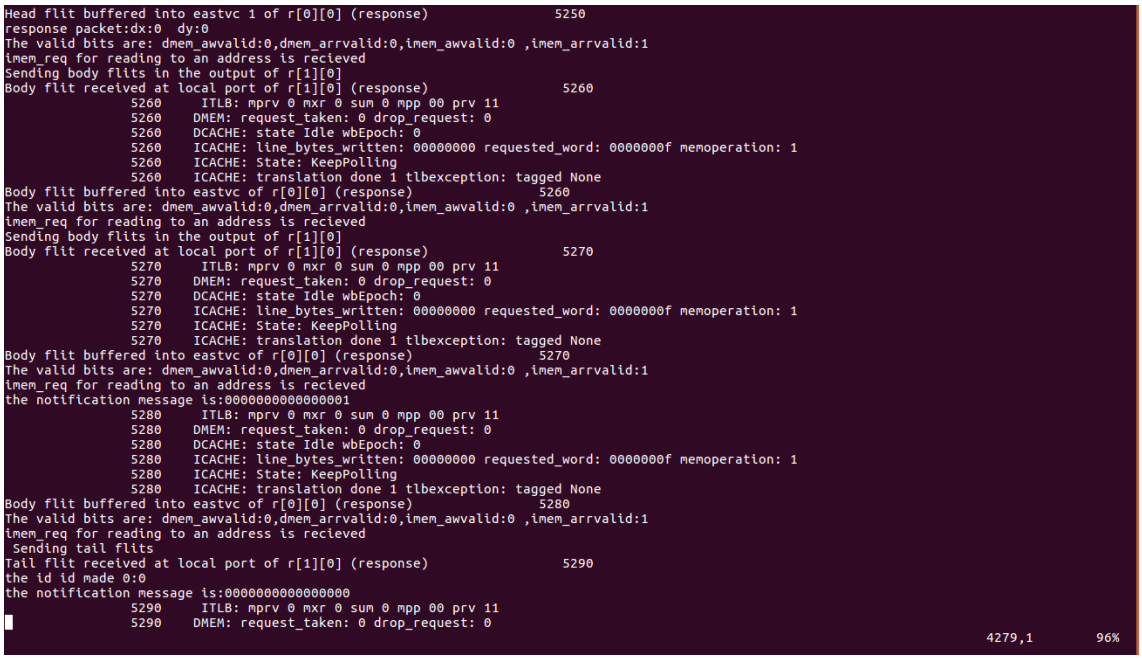


Figure 6.9: **BSV Simulation for Test bench 2**

In figure6.9**BSV Simulation for Test bench 2**figure.6.9 its seen that head flit and body flits are buffered into node(0,0).

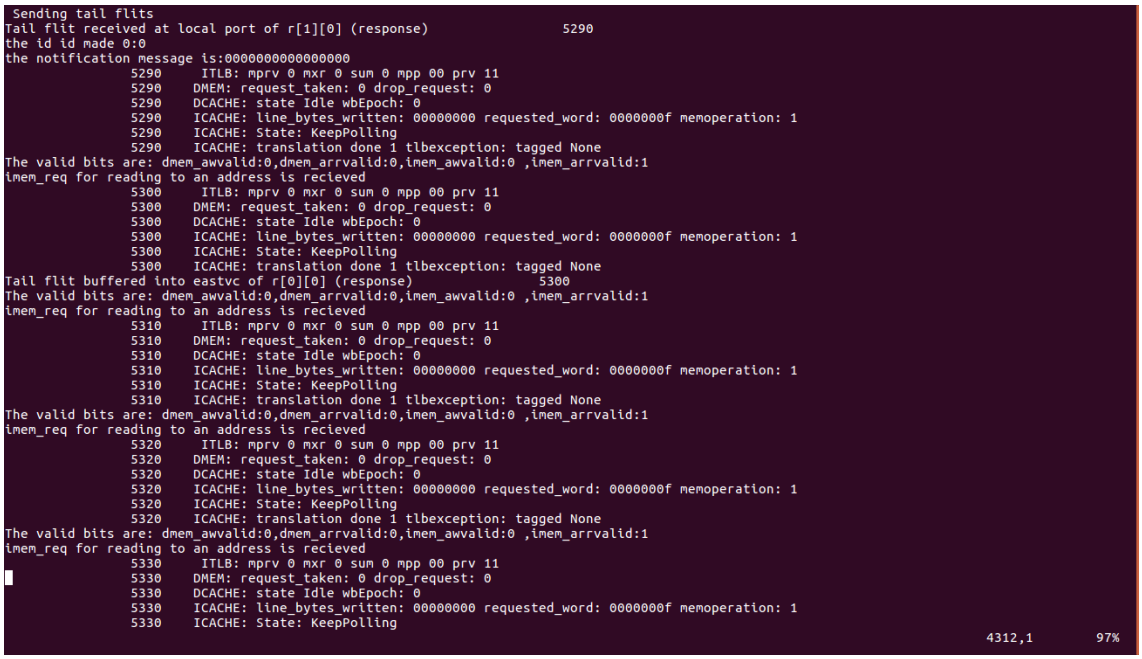


Figure 6.10: **BSV Simulation for Test bench 2**

In figure6.10**BSV Simulation for Test bench 2**figure.6.10 its seen that tail flit is buffered into node(0,0) .

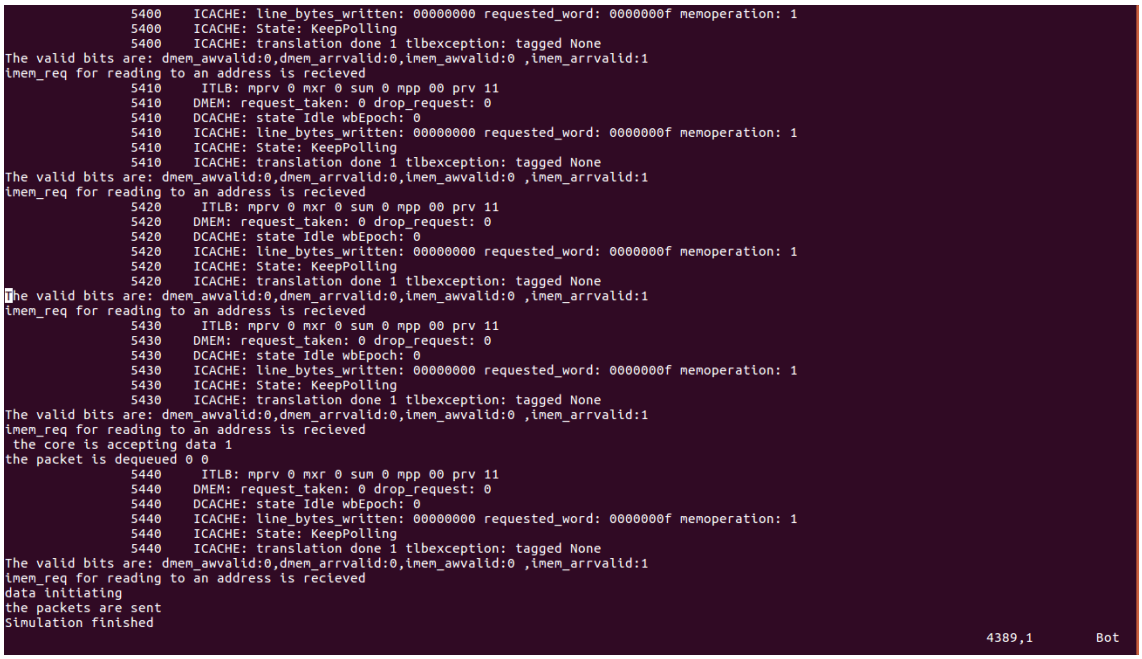


Figure 6.11: **BSV Simulation for Test bench 2**

In figure6.11**BSV Simulation for Test bench 2**figure.6.11its seen that the response is sent back to the core.

6.4 OBSERVATIONS

The following observations are made regarding the number of clock cycles taken for various tasks when a single flit is introduced:

- Time taken for the request to reach all nodes from the time the head flit is introduced = 23 clock cycles.
- Maximum latency for hit to occur from the time the head flit request is introduced = 27 clock cycles.
- Maximum latency for response to reach destination node since the hit occurs = 46 clock cycles.
- Minimum latency for response to reach destination node after the hit occurs = 6 clock cycles.
- Maximum latency for request to be served when the single request is broadcast =72 clock cycles.

The following observations are made regarding the number of clock cycles taken for various tasks when two flits are introduced:

- Time taken for both request flits to reach all the nodes after the head flits are introduced =30 clock cycles.
- Maximum latency for the first hit to occur from the time the head flit request is introduced =31 clock cycles.
- Maximum latency for the second hit to occur from the time the second head flit is introduced = 50 clock cycles.
- Maximum latency for the first response to reach destination node after the hit occurs=37 clock cycles.
- Maximum latency for the second response to reach the destination node after the second hit occurs=52 clock cycles.
- Maximum latency for both the requests to be served after the head flits are sent out=102 clock cycles.

CHAPTER 7

CONCLUSION AND FUTURE WORK

An ordered mesh network interconnect has been implemented by making use of a virtual channel router architecture so that unordered requests could be processed in a global order by making use of the notification network and it's been verified by integrating with SHAKTI C-class core. This has provided a solution to the problem of ordering requests without any complexity.

A single-cycle pipeline optimization can be achieved to reduce the network latency and buffer read/write power, by implementing lookahead bypassing, wherein, a lookahead containing control information for a flit is sent to the next router during that flit's switch traversal stage. At the next router, the lookahead performs route-computation and tries to pre-allocate the crossbar for the approaching flit. Lookaheads are prioritized over buffered flits.

The received requests are processed in the order determined by the notification net- work. Therefore, there is a possibility of the network running into a deadlock, as the request that the node is awaiting might not be able to enter the node because the buffers in the node and routers enroute are all occupied by other requests. To prevent this scenario, one reserved virtual channel (rVC) can be added to each router, reserved for the coherence request with source id equal to the expected source id of the node.

Also, we can enforce a mechanism where requests from the same source also can be ordered with respect to each other. Since requests are identified by source

id alone, the main network must ensure that a later request does not overtake an earlier request from the same source. To enforce this two requests at a particular input port of a router cannot have the same source id. At each output port, a source id tracker table can be used which keeps track of the source id of the request in each virtual channel at the next router.

REFERENCES

1.Bhavya K.Dal et al "SCORPIO:A 36-coreResearch chip demonstratingSnoopy coherence on a scalable mesh NoCwith in network ordering"ISCA,2014

2.Mark Hill,"On-Chip networks",University of Wisconsin Madison.

3.Shakti series Processors.ppt Rise lab.

4.William Dally,"Virtual channel flow control",ISCA,1990.

5.William James Dally and Brian Towels,"Principles and Practices of Interconection Networks",Morgan Kaufmann Publishers Inc.San Francisco,CA,USA,2003

6.John L.Hennesy and David A.Patterson,"Computer Architecture:A Quantitative Approach",5th edition,2011.

7.Shubhangi D Chawade, Mahendra A Gaikwad and Rajendra M Patrikar"Review of XY Routing Algorithm for Network-On-Chip Architecture",International journal of Computer Applications(0975-8887) Volume 43-No.21,April 2012.

8.Bluespec-Inc, Bluespec System Verilog BSV by Examples. Bluespec Inc, 2010.

9.Bluespec-Inc, Bluespec System Verilog Refeerence Guide. Bluespec Inc, 2014.