NOTTINGHAM TRENT UNIVERSITY

School of Science and Technology

Low Latency Router Microarchitecture for Network on Chip

Nikesh Lama

2015

Project Report submitted in partial fulfilment of the requirements of Nottingham Trent University for the degree of MSc Engineering (Cybernetics and Communications)

Abstract

The increasing chip density has enabled multiple cores to be integrated within a single chip giving rise to System on Chip (SoC). To overcome scalability and performance bottleneck in SoC, networking concept is introduced in the chip design called Network on Chip (NoC). NoC introduces routers within the chip which forms the communication framework. This allows the decoupling of communication aspect of the design from the computation. The architecture and the routing algorithm used for the router determines the latency induced in the router and the induced latency plays a vital role in determining the SoC performance.

The proposed router microarchitecture implemented on an FPGA has a low latency single pipeline stage, best suited for small networks with relatively lesser traffic. Results have shown that in comparison to generic four pipeline stage router, the proposed implementation reduces the router delay by up to 60%. In the proposed design, there are multiple stages running concurrently demonstrating parallelism. The design uses distributed deterministic routing algorithm which requires only the destination router *Id* information whereas source routing requires all the routing information embedded in the packet, making the packet size larger.

The FPGA implementation of each router utilized only 1% (410) of the total Adaptive Logic Modules (ALMs) available on the De1-SoC prototyping board packaged with Cyclone V FPGA. The maximum clock frequency possible for the design for the FPGA was observed to be 147.3 MHz.

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Dr Kofi Appiah, for the guidance, support, useful remarks and motivation throughout the Masters Project period. His expertise and research works in FPGAs helped me to learn a lot about the FPGAs and Network on Chip (NoC), which turned out to be invaluable for my Master thesis.

Furthermore, I would like to thank Mr. Pedro Baptista Machado, who is a Research Associate in Computational Intelligence Lab in Nottingham Trent University, for providing me guidance and assistance to better understand FPGAs and NoC concepts.

Lastly, I would like to thank my family, especially my Mom, Dad and my aunt (Mom's sister) for continuous love, support and motivation throughout my life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SoC	System on Chip
NoC	Network on Chip
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
CMOS	Complementary Metal Oxide Semiconductor
VLSI	Very Large Scale Integration
CPU	Central Processing Unit
GPU	Graphics Processor Unit
HDL	Hardware Description Language
VHDL	Very High Speed Integrated Circuit HDL
PLL	Phase Locked Loop
CLB	Configurable Logic Block
I/O	Input Output
MLAB	Memory Logic Array Block
FLIT	FLow control DigIT
VC	Virtual Channel
SOF	SRAM Object File
JTAG	Joint Test Access Group
FIFO	First In First Out
LSB	Least Significant Bit
MSB	Most Significant Bit

Chapter 1 Introduction and Overview

1.1 Introduction

Due to technological advancement, a single chip can contain more than a billion transistors which results in the accommodation of multiple different cores forming a system on its own. This integration also called System on Chip (SoC) has helped to revolutionise all aspects of chip industry. As more and more cores are embedded inside a single chip, point-to-point and shared bus communication framework are failing to keep up with the growing number of cores in terms of scalability and performance. A new paradigm called Network on Chip (NoC) is introduced to provide scalable and modular communication framework for SoC designs. NoC implementation has proven to overcome the limitations posed by traditional interconnects. Many research works have been conducted on NoC because NoC has proven to be scalable and really effective for complex SoC designs.

NoC decouples the communication aspect of the design away from the computation aspect of the design. NoC concept introduces routers or switches modules that are connected with cores and are responsible for handling communication with other cores and routers. Hence, the router architecture and the routing algorithm used by the router determines the overall performance of the network and the SoC.

1.2 System on Chip

The number of transistors in a single chip doubles every 18 months as predicted by Moore's law due to of advancement in CMOS (Complementary Metal Oxide Semiconductor) technology [1]. This has enabled VLSI (Very Large Scale Integration) technology to flourish allowing huge integration of transistors into a single chip. Multiple computing resources like CPU (Central Processing Unit), DSPs (Digital Signal Processors), memory, input output (I/O) controllers, specific functional IP (Intellectual Properties) can all be integrated in a single chip giving rise to a new era of System on Chip (SoC) [2]. The challenge with SoC design however, is the communication between these increasing different computing resource blocks which

are interconnected via shared bus and point-to-point links. Both of these communication infrastructures have limitations in terms of performance and scalability because with complex design in shared bus, high traffic is experienced, causing delays and congestion resulting in performance degradation. Point-to-point is not at all scalable because it is higher chip-area demanding to implement point-to-point links in complex SoCs [3]. Simply put, increment in the number of cores proportionally increases the complexity of the interconnects.

1.3 Network on Chip

Realizing the architectural and performance bottleneck with the ever increasing SoC core density, a scalable architectural approach for SoCs, Network-on-Chip (NoC), is developed as a potential solution to provide scalable, reliable and modular communication infrastructure platform [2]. NoC introduces the concept of packet switched network communication within the chip by introducing switches or routers between the cores rather than connecting them directly [4]. In NoC architecture, data is divided into smaller data packets with overhead containing necessary information like source address, destination address, Time to Live (TTL) value which determines the maximum hops the data packet can take and so on .

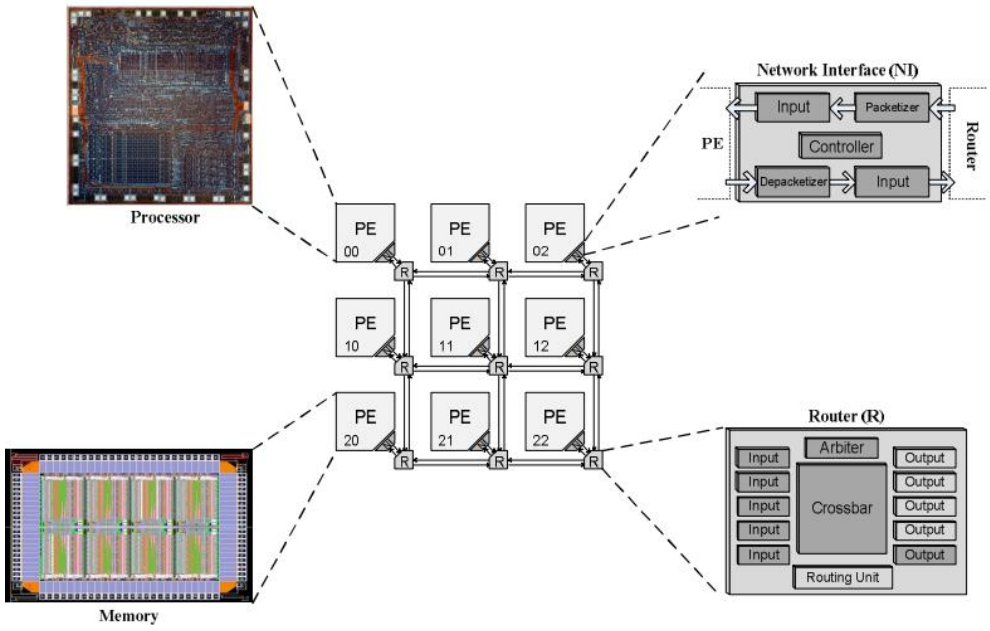


Figure 1.1 Generic Network On Chip Layout [5]

Figure 1.1 represents a generic NoC layout. Each resource, also can be referred to as processing element (PE), which could be DSPs, I/O interface, Memory, Graphic controller and so on, is connected to the router. The resource block sends packets to the router and the data packets are routed over the network based on the destination address and the routing protocol being used.

Route taken by the data packets are governed by the routing algorithms and the router architecture itself, and hence, they play a vital role in determining the performance of the network. Latency is one of the most important factors when it comes to determining the performance of the NoC. A well designed router algorithm can reduce the latency, but still maintain the packet integrity during the transmission which is equally important. Over the time, many different kinds of routing algorithms and router architectures have been developed with different characteristics such are path determination and adaptivity features of the algorithm to reduce the latency [6].

NoC provides communication framework based on predefined topologies like mesh [6] , torus [7], star [8], ring [8] and more, upon which the SoCs can be implemented. This kind of predefined structured layout reduces the complexity of the SoC designs.

1.4 Goals and scope of the thesis

Architecture of the router is the key component in designing the NoC communication framework and evaluating the performance of the network. And the architecture of the router depends on the performance requirement of the network. Latency plays a big part in determining the performance of the network.

The main goal of this thesis is to design a router microarchitecture using deterministic distributed routing protocol with an objective of reducing the latency by reducing the number of pipeline stages in the router. The router design is then replicated to form a 2x2 mesh topology with credit based flow control mechanism. The next phase of the goal is to model the design with Verilog Hardware Description Language(HDL) using Altera's Quartus II 14.1 and run tests using simulation tool, ModelSim, and analysing the waveforms to validate the design. The final phase comprises implementing the design onto the Terasic De1-SoC board packaged with Altera's Cyclone V FPGA [9]. The resource requirements are then analysed to make further evaluation of the design.

The latency directly corresponds to the number of clock cycles it takes for a data to get processed out from the router. The design therefore, is kept relatively simpler with simpler flow control to reduce the number of router pipeline stages. So, the design is proposed as a single pipeline stage router where in one clock cycle data packets are injected into the router and stored in the buffer, and in the next clock cycle, previously stored data are routed out of the router to the destination or the next hop making the router design fast and simple for small networks.

To meet the goal requirements of the thesis, the process can be generalised into following tasks

- The specifications of the router designs are sketched out first.
- The router is then broken down into different components. Each component is further divided into smaller sub components if required.
- Each component block is modelled in Verilog and tested using testbenches to validate the design.
- Then all of these components are put together to form a whole router.
- The router is further replicated to form a 2x2 mesh topology. Each router has five input and five output ports where one of the input and output ports is connected to the resource block or core.
- The design is validated by running simulation of the design in ModelSim. Further tests are conducted to calculate the router delay and the latency to make comparisons with the already existing architectures.
- The design is finally compiled and implemented onto the Terasic De1-SoC board with Cyclone V FPGA to evaluate the resource requirements.

1.5 Thesis organisation

This section gives a brief overview on the organisation and structure of the thesis.

Chapter 2 gives brief background introduction about NoC, router architecture and routing algorithms. This chapter discusses about the concepts and terminologies used in NoC to facilitate the readers for further reading of the thesis. The second part of this chapter includes an extensive literature review on the related topics.

Since, router design is the main part of the thesis, **chapter 3** gives a detail explanation and justification of the design choices.

Chapter 4 discusses about the modelling of the router, simulation and the results obtained from the simulation of the different components as well as the network as a whole.

Chapter 5 talks about the FPGA implementation of the design and evaluates the results obtained from the prototyping of the design onto the De1-SoC board which uses Altera's Cyclone V FPGA. This chapter also does the critical comparative evaluation of the proposed design with the existing designs.

Chapter 6 concludes the thesis with a summary of the work done so far. Furthermore, limitations of the design is also discussed with future improvement prospects.

Chapter 2 Theoretical Background and Literature Review

This section provides a background materials on how Network on Chip helps to bring improvements over the conventional bus and crossbar interconnections. This chapter also introduces the theoretical concepts on NoC networks, router architecture, routing algorithm and different components of the NoC design which prepares readers for the design development and explanation phase in chapter 3.

Justification of use of FPGA over use of CPUs, GPUs for the prototyping of the design is discussed. Further discussion on basic concepts on FPGA architecture and FPGA design flow is also presented. The last part of the chapter presents a critical review on the related work from where few of the literature are considered for the router design.

2.1 Concept of NoC

The reason NoC came into existence is due to the limitations faced by conventional bus and crossbar interconnects in SoC design. As the chip density is increasing exponentially, the conventional point to point or shared bus interconnection is simply inefficient in terms of performance, resource utilisation and thermal heat dissipation. We will look into some conventional interconnects and make comparison with the NoC to better understand the significance of NoC in SoC designs.

2.1.1 Point-to-point interconnects

In early days of SoCs, the cores were connected directly forming a point-to-point interconnection. The data are send directly from one core to another. The communication is fast since there is no need for flow control or arbitration or any intermediate nodes in between. However, as the number of cores increases, so does the complexity in the interconnects. The main problem with point to point interconnect is the large number of wires and input output ports needed. For example, for VGA core in figure 2.1 , it needs three I/O ports to connect to the other three cores and it

seems a reasonably good choice, but if there are hundreds of cores then it is simply not scalable to accommodate all. Also with every new core added, the VGA input output port needs to be increased to establish connection. It takes a lot of chip area and is not flexible with the changes [3]. The resource is dedicated, so even if it is not in used, it is fixed for a particular connection which results in poor utilisation of the resources.

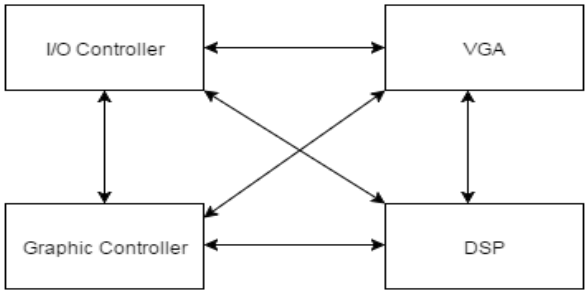


Figure 2.1: Point to point interconnect

2.1.2 Shared Bus Interconnects

To mitigate the limitations faced by point to point interconnects which are mainly, large number of input and output ports and high chip area requirement, shared bus based interconnect is introduced. All the cores are now connected via a shared bus as shown in figure 2.2 instead of direct point to point. This provide more scalability compared to the point to point connection because cores can be added and removed with only few changes as the design of the core need not to be altered. The chip area is also reduced since the number of I/O pins are also reduced. Since the bus is shared, some sort of arbitration is needed to allow fair use of the bus and monitor inter core communication.

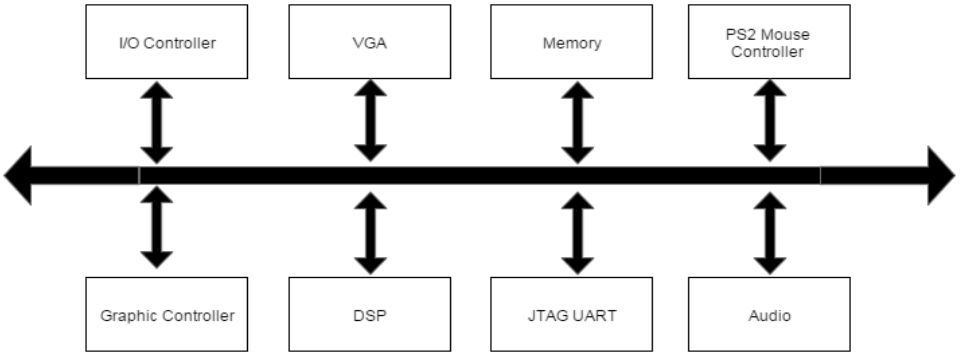


Figure 2.2: Shared Bus Architecture

The bus based interconnect is definitely better than point-to-point interconnects in terms of scalability and I/O pins used. However, the bus interconnect is scalable only to a certain extent [10] .With increasing number of cores, more and more cores are sharing a single bus. Since, only two cores can communicate at a time, this causes latency and reduced bandwidth [3] resulting in poor performance.

2.1.3 On chip router based interconnects

Scientists and engineers have been contemplating the idea of creating an on-chip network so as to make better use of the resources by separating communication from computation for efficiency and to make the communication more structured and most importantly make it scalable to accommodate ever so growing SoC complexity. Hence, the concept of Network on chip was started in early 2000 by Kumar et al [11] to mitigate the bottleneck in SoC communication.

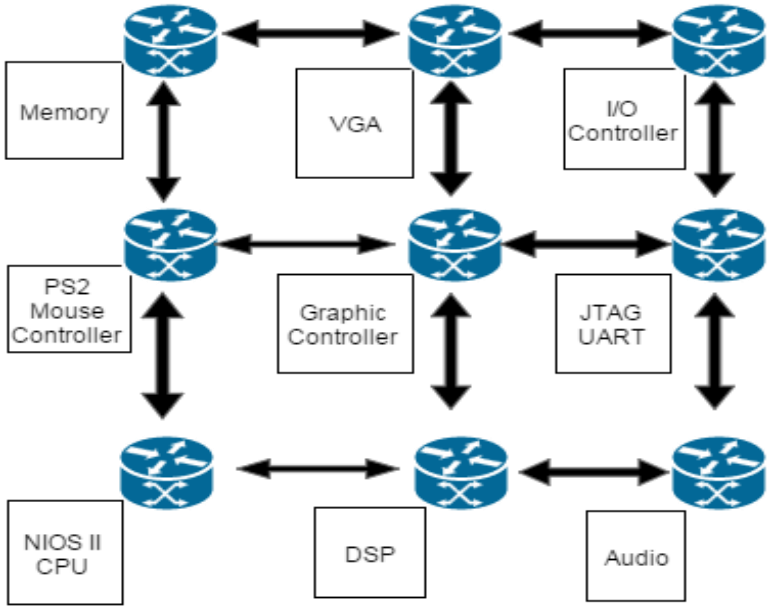


Figure 2.3: Router based Interconnect

The networking concepts used in conventional network is also applicable for NoC. Routers are introduced before each core which take care of data routing and arbitration logic. The figure 2.3 shows the NoC layout with 9 different cores connected with their individual routers . The communication aspect of the chip is undertaken by the NoC structure as demonstrated in the figure 2.3. Data are divided into data packets before sending from the core and the destination router assembles the data

packets upon arrival of the packets before forwarding it to the destination core [12]. Hence, the processing cores need not to take part in the communication aspect of the design. This decoupling of communication and computation helps to make the communication more structured, modular and scalable [13].

Some of the major advantages of NoC are listed below [14] [13] [15]:

- High Scalability and simplified customization.
- Better resource utilisation.
- Low Cost.
- Less chip design time.
- Better performance.

2.2 Terminologies used in NoC

This section introduces some of the important and common terminologies used in the realm of network on chip which are relevant to this thesis.

Message

Message is an actual data that has been sent from the source to the destination. It could be fixed or variable length.

Packets

Larger messages are simply too big to send it as a whole, so they are divided into many smaller data packets which are routed over the network independently and are assembled at the destination. Packets contain overhead information about the source address, destination address, packet sequence information for reassembly and so on. The actual body of the packet is called payload.

Flits

Packets can further be divided into smaller units called Flits(Flow Control Digit) [16]. Flits have fixed size and unlike packets, flits cannot route independently. The head flit travels first followed by the body flits and terminated with a tail flit.

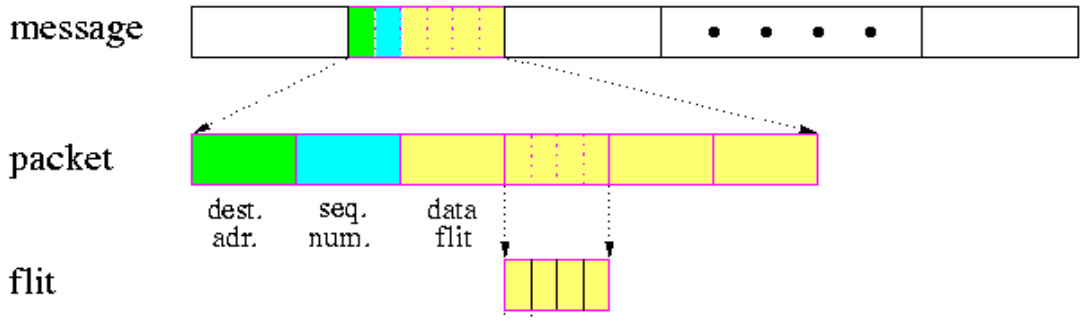


Figure 2.4: Message, packet and flit [16] .

Buffer

Buffer is a memory storage unit for packets or flits in transit in the router. Upon receiving the packets or flits, they are stored in the buffer before the arbitration and switch traversal are done. Buffers help to avoid packet loss during the congestion and longer router computation.

Virtual Channel

Virtual channel is a concept of using the single physical channel in a time multiplexed fashion. The channel is virtually divided into multiple channels and are given time slots to transmit.

2.3 Routing Algorithms

Routing is a process of appropriately forwarding the received data to the right output port. Routing algorithms govern how the data packets are traversed over the network. Appropriate choice of routing algorithm can result in effective router design. Routing involves computing the set of possible output channels for the data followed by selection of the appropriate channel [17].

According to [18], two of the most important features that all routing algorithms need to possess are livelock and deadlock freedom. Livelock freedom ensures that packets do not wander across the network indefinitely without ever reaching the destination. This could happen when the packets are stuck in a loop. This is particularly bad for the network performance because it takes up the network resources and causes

unnecessary congestion. Deadlock on the other hand happens if the packets are blocked for indefinite period. If deadlocks happen then the packets could never reach the destination creating a long queue for the following packets as well [19]. So, a routing algorithm should be deadlock and livelock free to add any benefit to the network performance.

Routing algorithms can be classified into two types based on the path determination feature of the algorithms.

2.3.1 Source Routing

Source routing is when the paths are predetermined from the source to the destination. The path information is stored in the packet header. Since, the path is pre-determined, there is no need for routing computation in the router. The router simply uses the information in the overhead to route the packets. Since packet stores routing information in the header, as the network grows, so does the size of the header. With large network, the header size grows significantly making it impractical for NoC [17].

2.3.2 Distributed Routing

Unlike source routing, in distributed routing, the routing computation is handled by the router itself. The source only has to include information of the destination address. The router analyses the destination address and makes decision to route the packet to the right output. Since the routing computation is part of the router, the complexity of the router is more compared to the source routing. On the other hand, the size of the header doesn't increase significantly with the increase in the network size. For a mesh network with k routers, the destination id size is $log2(k)$ [17] for deterministic distributed routing.

2.4 FPGA Technology

Since, the router design is prototyped in an FPGA board for this project, this section give some insights about what FPGAs are and how are they useful for prototyping.

FPGAs are integrated circuit with programmable semiconductor devices laid out in a matrix of configurable logic blocks (CLBs), hence called an array of logic blocks. Along these array of logic blocks runs configurable interconnects as shown in figure 2.5, which can be electronically programmed with Hardware Description Language (HDL) like Verilog or Very High Speed Integrated Circuit HDL(VHDL) [20].

The idea of programmable logic started since 1960's with many variations and the first modern FPGA was developed by Xilinx in 1984 which contained only 64 logic blocks and 58 input and output ports [20]. Nowadays, the number has gone up to hundreds of thousands of logic blocks, and hundreds of I/O ports with addition of IP cores like DSPs, memory, Multipliers, PLLs(Phase Locked Loop) and so on. FPGAs consists of combinational blocks which are simple flip-flops and primitive logic gates, and special blocks such as block RAM (Random Access Memory), PLLs and so on. Due to this ample amount of resources, digital fabrication for almost any kind of circuit is easily manageable in FPGA even with large area and power usage. And for the same reasons, many people use FPGA as a prototyping/ fabrication tool to test router architecture and algorithms . Figure 2.5 shows the general architecture of an FPGA which shows the configurable logic block with configurable interconnects and the I/O ports.

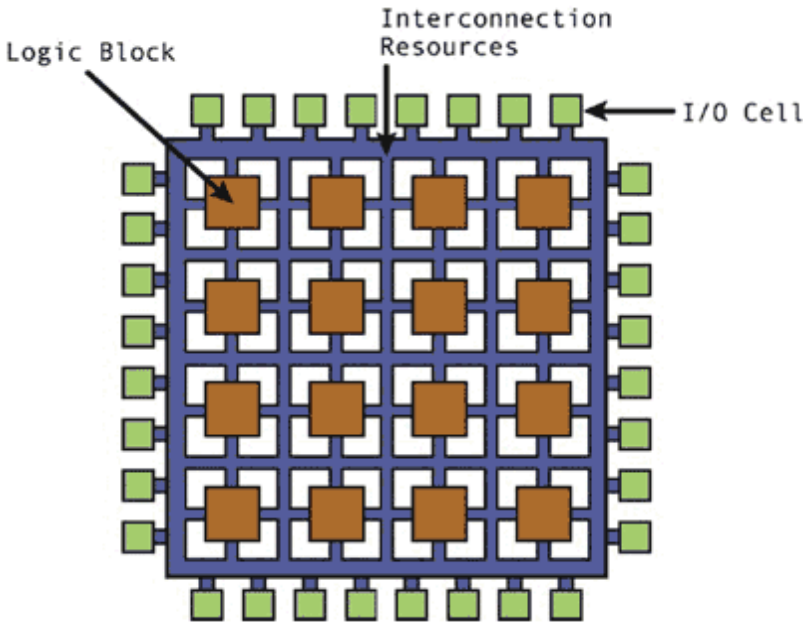


Figure 2.5: FPGA architecture [21]

2.4.1 FPGA vs ASIC

Unlike Application Specific Integrated Circuit (ASIC) like CPUs or GPUs, which are designed for a particular task, FPGA on its own doesn't have any functionality unless programmed. ASIC designs cost a lot of money for prototyping and can take months for fabrication whereas FPGAs are much cheaper and can be fabricated into virtually any devices in the digital domain in no time and can be reconfigured if mishaps happen which otherwise would have cost a fortune with ASIC fabrication. Cost and time are the two driving factors for engineers to use FPGA for prototyping purposes. However, FPGA's extreme flexibility and re-configurability comes at the cost of area usage, power consumption and latency. According to [22], FPGAs are 3 to 4 times slower, takes 20-35 times more area and takes as 10 times as much power than ASICs. So, the incredible re-configurability and fabrication features have trade-off in terms of area, power consumption and delay. Regardless, the benefits of FPGA outweigh these trade-offs because FPGAs are economically accessible for anyone and any organisation whereas the ASIC designs keep getting expensive due to deep submicron processes.

2.4.2 Cyclone V architecture and Design Flow

The FPGA used for prototyping on this project is Altera's cyclone V FPGA. The board is De1-SoC from Terasic which uses Altera's Cyclone V FPGA and it also has an embedded Arm Cortex A9 dual processor as a hard processor [9]. Please refer to *Appendix A* for a picture of De1-SoC with the labelling showing hard processor and the FPGA part of the board. Altera's FPGA basic architecture is same regardless of having many different families. The figure 2.6 shows the basic architecture of the Cyclone V SoC.

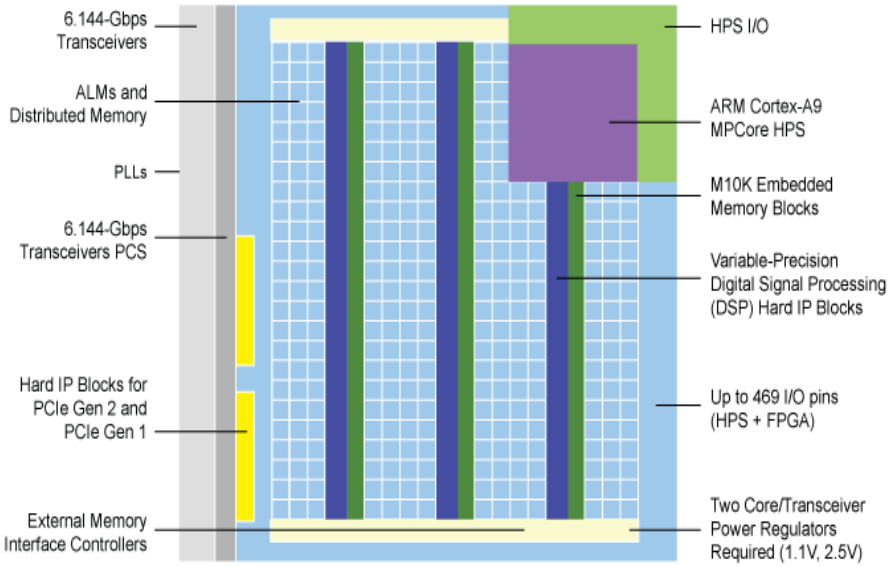


Figure 2.6: Cyclone V SoC Architecture [9]

Some of the features of Cyclone V FPGA architecture are listed below [23]

- Up to 300,000 logic elements arranged vertically as ALMs(Adaptive Logic Modules).
- It has about 12 Mb of block RAM arranged as M10K block and about 1.7 Mb of distributed memory as small storing units arranged as MLABs (Memory Logic Array Blocks).
- As a clock source, it has around 6-8 fractional PLLs(Phase Locked Loops)

Quartus II is a development tool from Altera to design and program Altera FPGAs. The first stage of development process is to write modules in Verilog of VHDL. The modules can be written in Quartus IDE or can be imported. The design is then compiled in Quartus which comprises of different stages of analysis to generate a compilation report about the design. Figure 2.7 shows the design flow in Quartus.

Firstly, the errors are checked in analysis and synthesis. If any found, the compilation fails and the design cannot be synthesized. Analysis and synthesis generates a schematic of the design which can be visually viewed using the RTL netlist option. Routing and fitting analysis is done to make sure that the design can be programmed into the targeted device. Summary of the resources used up with the design is generated. Furthermore, timing analysis is also carried on to verify the operation of the design. After complete compilation, Quartus generates .sof (SRAM Object File)

file which is a binary file containing information to configure SRAM based Altera device . The .sof file is then used to program the target FPGA board. [24] [25]

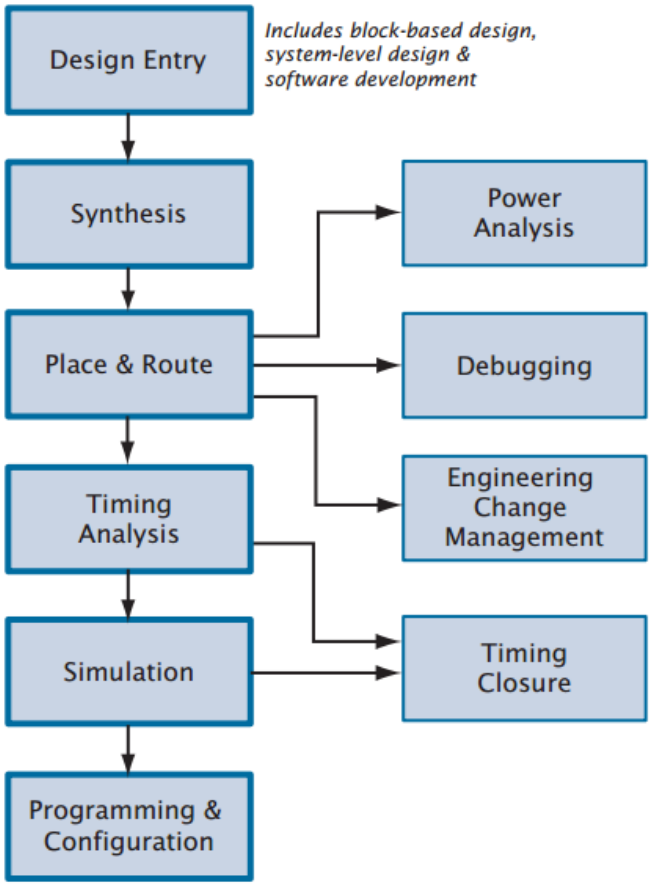


Figure 2.7: Quartus II Design Flow [26]

2.5 Related Work

The exploration of the NoC started around 90's and Kumar et al. were the first to implement the concept of NoC with a packet switched communication in a 2D mesh topology in 2002 [11]. The area of NoC has been particularly intriguing and challenging at the same time, and is still yet to be explored to reach its maximum potential. They proposed a methodology where the architecture of the network is first laid out and then the desired application is mapped onto that architecture. This methodology has enabled the network to be well structured and scalable because of the decoupling of actual application and the network architecture.

Their research [11] was motivated to meet the three major necessities for NoC design, which are, increasing demand of interconnection bandwidth, need to gradually reduce the cost of engineering and the best utilisation of available resources. They also proposed a 4 layered protocols adapted from the Open System Interconnection (OSI) model, namely, physical, data-link, network and transport layer adapted from computer network. However, the authors haven't discussed any algorithms explicitly when it comes to routing. The paper focuses mainly on the architectural aspect of NoC design and not the explicit routing algorithms, nevertheless, it sets a foundation for further works.

In NoC design, every node is connected to a router which are further connected to other routers and those to their respective nodes. So traversal distance becomes more as the network grows and so does the latency. Reducing latency is a crucial part for system performance as the latency is introduced at every communication pair and especially for real-time SoCs where timing is critical, latency becomes a vital issue.

Initially the routers used simple routing algorithms like Dimension-order routing (DOR) which is a simple routing algorithm where the packets are sent along X- axis and Y-axis or vice versa [27] coupled with switching techniques like wormhole switching. With more complex SoCs, the virtual channels (VC) are also introduced to time multiplex the physical channel to handle congestion . All of these different additions made the router design more complex with more induced latency. Many engineers have conducted research to reduce the latency issue in SoC designs.

Realising the latency issue, Roca et al. in [28] proposed a router architecture design with the motivation to reduce the critical path of the router pipeline to decrease the latency as shortening the longest pipeline length increases the clock frequency. The research was driven with the idea that having a single complex arbiter limits the performance so they used multiple simpler arbiters that can run parallel and used DOR routing to reduce the latency. In that sense, the A. Roca et al. introduced parallelism in the architecture of the router.

Roca et al. [28] successfully managed to demonstrate reduction in the router latency from 10% to 20% and the network latency from 11% to 15%. However, it comes at a cost of larger area usage by router. And also the actual overall network latency has only reduced by 4%. Moreover, the router algorithm used is rather simpler and more focus is put forward on the actual architecture of the router than the routing algorithm.

Contrary to [28] , Frazzetta et al. in [29] proposed a rather different approach to reduce latency issues in NoC. Most NoCs are densely packed with large number of interconnecting wide data bus. Due to the dense packing, it is likely to have faulty links and cross talk between wires. It is a trade-off that comes with large integration of transistors. Rather than discarding those partially faulty lines, D. Frazzetta et al. proposed to use those faulty bus links and instead make the router architecture more robust to fault tolerance, meaning the system's ability to operate even with some faulty components. For example, if a data bus of 32-bit line has 16 faulty wires, then that data bus can still be used to transmit 16 bit from that bus making efficient use of the available resources rather than completely avoiding them. However, changes are needed for the NoC routers to be fault tolerant of the faulty lines. Unsurprisingly, there was a trade-off of area and power as the router would take 25% extra area and 5% extra power consumption for the incorporation of the design for the upgraded router. However, the overall performance yields 20% more energy efficiency. Even though this results in a more complex router taking up more area and power, this approach is significant in terms of efficient available resource usage especially when there are more than billion transistors in a single chip, the chances of faulty lines are more and this approach would make better utilisation of these discarded resources.

Similar to [29], Kumar et al. in [30] came up a more complex router architecture for large NoCs which requires routing table in each router. They proposed a Junction Based routing algorithm which is an extension of source routing (predetermined path)

with reduced overhead information for the destination. Rather than using the final destination address, the overhead only keeps information for the next hop decreasing the packet overhead. Intermediate nodes called junctions are introduced between far apart nodes so as to reduce hop information in the header. On the other hand, the router has to now know which hops the packets should take to reach the destination. This architecture is relatively complex with intricate flow control systems. Similar to [28] they are also using pipelined design for the router to achieve low latency. The prototype implementation of the router architecture was done in a Terasic De2 board using Altera Cyclone II FPGA and modelled in VHDL. The authors successfully demonstrated that it is feasible to prototype large NoCs on an FPGA. Compilation report of the test showed that only 7% (2259)of the logic elements and 7% (2244) of the combinational functions were used. The resource usage of the router is relatively higher compared to other simpler routing like source, DOR etc. due to complexity in the design. But the junction based routing technique reduced the packet overhead by replacing all the traversal information for the packet with just the information for next hop node.

As the number of functional blocks increases, so does the congestion in a network [8]. Not only the performance has to be considered but also the thermal dissipation which can affect the life of the chip has to be considered too. Gaoming et al. in [6] addressed the performance trade-offs with power consumption and heat dissipation and proposed a dynamic and mixed routing algorithm that focuses on congestion and thermal dissipation effects in NoC. Congestion is caused due to packet overcrowding on the links which may be caused due to ineffective load balancing techniques or deterministic routing (which uses predefined path). A mixed routing (MIXROUT) was proposed by Gaoming et al. in [6] which is a combination of XY routing and Multiple and Load-Balance Path Routing (MULTI). MULTI is an adaptive routing that helps reduce the congestion but at the cost of high energy and heat dissipation. So, basically MULTI is activated during high congestion to reduce the congestion and after that XY routing is activated when there is less congestion to yield better performance. So there is a dynamic routing that changes as per the load of the link. This allows a dynamic load balancing mechanism which reduces latency dynamically.

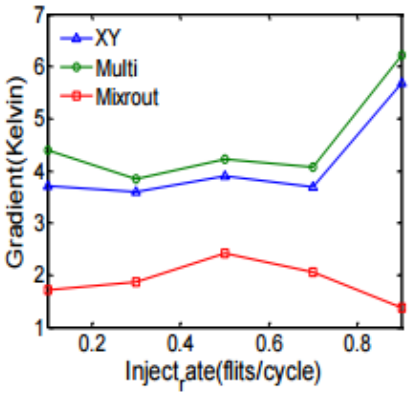


Figure 2.8: The Thermal Gradient [6]

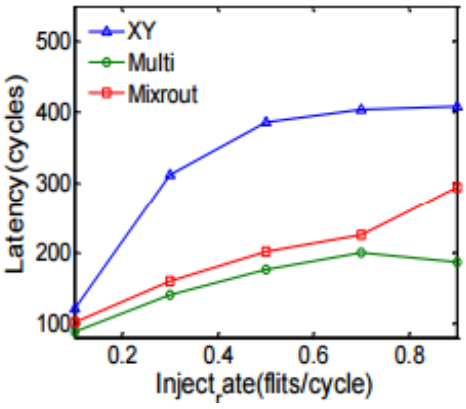


Figure 2.9: The Average Latency [6]

From the figure 2.8 and figure 2.9 , the authors clearly demonstrate that MIXROUTE algorithm has latency as low as MULTI algorithm as it switches to MULTI algorithm mode during high congestion. Since, it dynamically does load balancing. Figure 2.8 shows that MIXROUTE algorithm exhibits relatively low temperature gradient compared to XY and MULTI, yielding better performance in terms of thermal balance of the system [6].

Similar to what Gaoming et al. in [6] have come up with a hybrid routing technique with focus on the congestion and heat dissipation, Yaghini et al. in [31] proposed a hybrid innovative routing algorithm combining the source and the distributed routing techniques by introducing a coding-based Tag architecture called TagNoC. The motivation however, is to reduce latency and increase throughput by reducing the packet overhead of source routing and per-hop running of a distributed routing.

Source routing requires the information for all the intermediate routers which results in a large packet overhead. With distributed routing the packet overhead is minimum as it only needs information for the next hop. TagNoC introduces a 2 bit tag number system that encodes the information to determine the output port of the router. This enables the determination of forwarding output port in parallel with the input buffering. Yaghini et al. [31] made quantifiable comparisons of TagNoC implemented on a 2 D topology with the existing baseline distributed router architectures and other extensions of source routing in terms of header size, latency and throughput. Yaghini et al. in [31] successfully demonstrated that in a 10X10 mesh topology, zero load network latency for TagNoC is about 33% less than the distributed and other extensions of source routing with just marginal 2 bit more packet overhead than the

distributed routing. Furthermore, there is 2% increment in the clock period compared to the baseline distributed and 6-12 % increment compared to other extensions of source routing. Also the dynamic power consumption has been decreased by almost 10% and routing logic area by significant 53% for a 4X4 mesh in comparison to baseline architecture. The authors have produced a significant results in terms of latency and area consumption. TagNoC however does not support the adaptive routing algorithm whereas Gaoming et al. in [6] used adaptive routing techniques to dynamically change as per the congestion.

In 2005, Sethuraman et al. [32] proposed a completely soft NoC router design implemented on an FPGA. Hard router design are faster, better performance, low power consuming and requires less chip area compared to soft router designs. However, hard router are not configurable meaning the unused resources cannot be re-used for other purposes.

Realising the benefits of both hard and soft router designs, Gindin et al. in [33], in contrast to [32], proposed a NoC design that have both hard (hard-coded in the silicon) router along with the soft routers to achieve high performance provided by the hard routers at the same time maintaining the flexibility. Also, the communication bottleneck between the hard cores and the soft cores can be reduced by using hard routers as an interface between the two. Gindin et al. [33] further proposed toggle XY (TXY) routing algorithm to avoid unbalanced capacity allocation in XY algorithm. However, this routing techniques has a disadvantage because now that a single flow is divided into two routes, there may be delay in the arrival of the packets. To re-order the packet, there needs to be an extra packet overhead encoding the sequence of the packet. Therefore, Gindin et al. [33] further improved it by adding weight to the flow division resulting in a weighted toggle XY (WTXY) algorithm [33]. Gindin et al. [33] later suggested an order routing by choosing one route to each source- destination flow, namely, weighted ordered toggle (WOT). WOT assigns XY and YX routes to the source-destination pair in such a way that reduces the latency. In that sense, WOT algorithm is simple yet balances the load and there is no need for large buffers for re-ordering the packets. Since, the algorithm is simple and ordered, the need for large buffers is also not required reducing the network cost. And Gindin et al. [33] have demonstrated it with that evaluation which shows, WOT's most loaded link takes 40% less link capacity in comparison to XY and 15% in comparison to TXY.

In contrast to all the literature covered so far, Seshasayan et al. [34] proposed a routing algorithm for a 3D NoC with the motivation of minimising the latency incurred. 3D topology is a topology where more than one 2D topologies are stack on top of each other. Seshasayan et al. [34] proposed a deterministic routing to choose the 3D node and XY routing for the 2D layer. So, to reach the destination, a packet has to traverse across horizontal and vertical direction if the source and the destination are in different layers. They implemented a 3D NoC with two 4X4 2D mesh layers. They also proposed to use the middle vertical links for more connection to the nodes i.e. 4 links for middle links and for edge links only 2. The implementation of the routing technique was done on Xilinx Virtex 2PXC2VP2 FPGA using Xilinx10.1 software to write modules and test benches. From the experiment, Seshasayan et al. [34] achieved almost 20% reduction in the time period required for the proposed scheme. Also, there has been decrement in the number of hop count to reach the destination. They found out that for the time period, there has been reduction of almost 1.5 ns from 9.036ns to 7.621ns compared to the existing algorithm and the hop count has decreased from 11 to 7. Since, they are not using all the vertical links, there is a reduction in the hardware requirement as well.

After looking at all the different literatures with different router architectures and algorithms it is quite clear that there is not a single architecture or algorithm that fits all. Depending on that the topology, functional requirement, available resources, bandwidth requirements, different router architecture would be best for different scenarios. Having a dynamic routing algorithm with some kind of intelligence would make the most efficient use of the resources with reduced latency, increased bandwidth and reduced packet loss just like [6] have incorporated two different type of routing algorithms to get benefits of both algorithms minimising the overall compromise in performance.

2.6 Summary

Due to the scalability and performance bottleneck of point-to-point and bus interconnection in SoC designs, Network on chip paradigms brings packet switching concept taken from conventional networks within the chip. NoC establishes a structured communication frame work within the chip making it more scalable with better resource utilisation and performance.

Routers are introduced within the chip which receives the incoming packet from different cores and route them based on routing protocols. There are mainly two categories of routing algorithms used in NoC, source and distributed. Source routing embeds all the routing information to reach the destination in its packet header whereas distributed routing only uses destination Id and routing computation is performed in each router to determine which direction to send based on the destination Id. The number of bits required to carry routing information increases linearly with the increasing network for source routing which causes serious packet overhead. In distributed routing, the number of bits for destination Id increases logarithmically with base 2.

The design is prototyped for De1-SoC board which uses Altera's Cyclone V FPGA. FPGA consists of configurable logic block with configurable interconnects. FPGA are extremely flexible and can be designed to anything in the digital domain. In contrast to CPUs and GPUs, FPGA allows to achieve parallelism at the hardware level.

Different literature are presented which covers the beginning of the NoC concept to more complex NoC architectures which uses dynamic routing algorithms which adapts to the change in network traffic. TagNoC proposed by Yaghini et al [31] is considered for the proposed design where the packets will be tagged with the output port number based on the destination Id similar to TagNoC. The FPGA resource usage for the design is considered to compare with the Junction based router proposed by Kumar et al. in [30].

Chapter 3 Router Architecture

This chapter focuses on providing detail description of the proposed router architecture design for a simple 2x2 mesh topology network. The router design is kept relatively simpler with credit based flow control system focusing mainly on reducing latency. The design employs a single channel router even though the packet structure has provision for virtual channel bits. Unlike typical multiple pipeline stage routers, the proposed router is a single pipeline stage router, hence, reduced clock cycles, hardware requirement and lower latency.

The figure 3.1 shows the topological view of the network in 2x2 mesh fashion with nodes[1] connected to the each router. This is the network structure which is used to test the functionality of the router. For testing, the packets are injected from the nodes and are also read at the nodes which could be any sort of processing core.

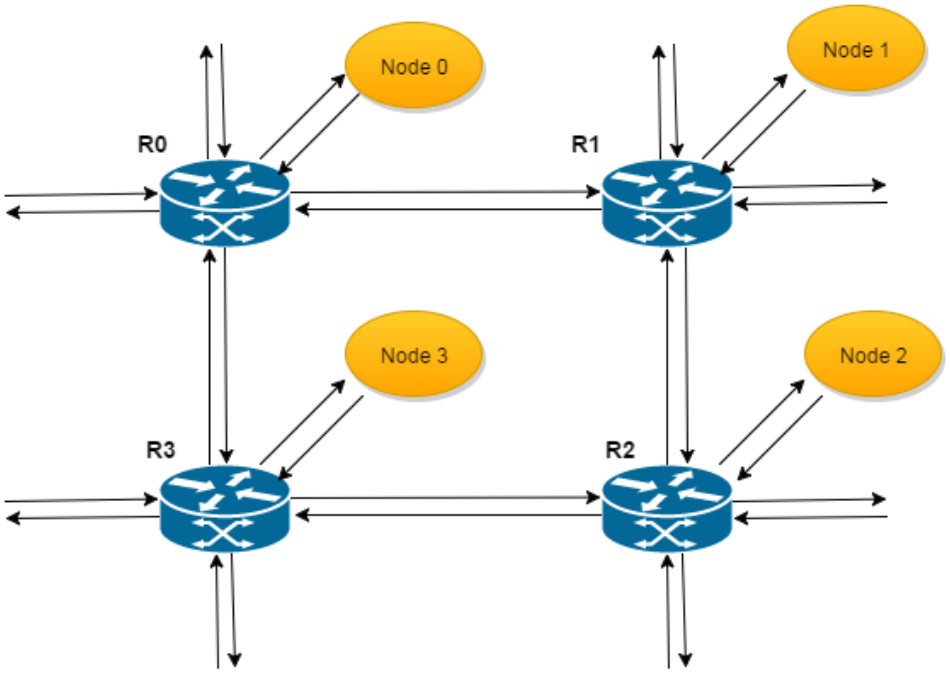


Figure 3.1: Topological View of the Network

[1] Nodes and Cores are used interchangeably in this thesis. Both refers to processing cores like CPUs, GPUs, VGA, Nios II processors and so on.

3.1 Router Design Overview

The router communicates with cores and other routers via I/O ports of the router. I/O ports are comprised of two channels each, for sending and receiving data and flow control bits separately as shown in the architectural block diagram in figure 3.2. With every data transmission there is a corresponding signal going in the opposite direction to advertise the buffer slot availability so that the router does not get overwhelmed with bursts of packets beyond the capacity.

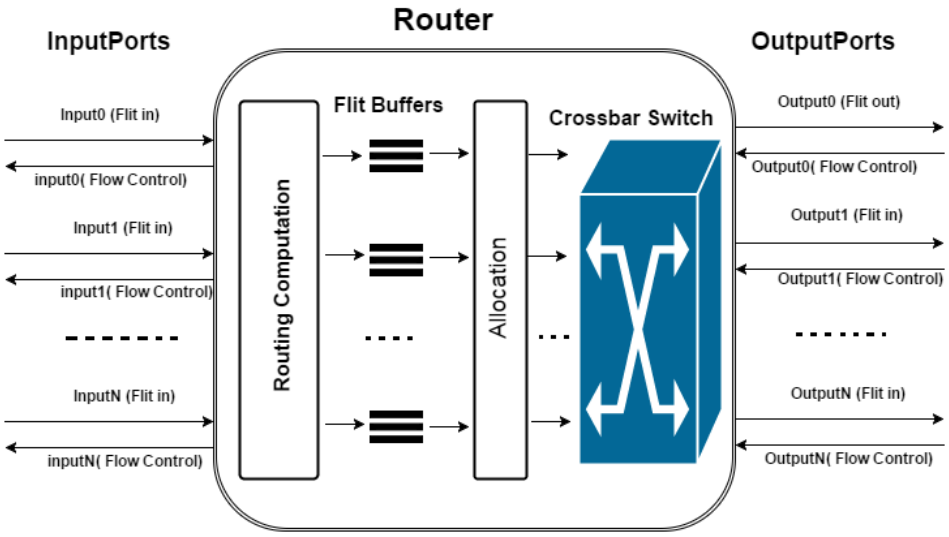


Figure 3.2: Proposed Router Architecture

The routing algorithm used is a look up table based distributed deterministic routing to keep the flit size small and the routing relatively simple. As the data flits are injected into the router, routing computation is carried out using the routing table stored in the form of hexadecimal file. The flit is then tagged with the output port number based on the destination Id retrieved from the flit. The data flits are then stored in the flit buffer before allocating to the switch for the right output port. Input-first Separable allocator is used to conduct the arbitration and allocation. The tagged output port number is used to arbitrate and allocate the output ports. After allocation, the flits are sent through the crossbar switch which connects to the right output port. Since, it is a single pipeline stage router, in one clock cycle, the flits are read in from the input port and are stored in the flit buffer. In the same clock cycle, previously stored flits are read out from the router. Therefore, it takes two clock cycles for a flit to inject into the router

and eject out of the router. This is a simplistic overview of the router architecture and the general working of the router.

3.2 Details of the Design

3.2.1 Packet/flit structure

The router works in the flit level in terms of granularity of the data transmission unit. The format of the flit contains all the necessary information about the data being transferred as an overhead. It is very important to keep the flit size concise because unnecessary overhead only degrades the performance and increases the resource usage.

For the proposed design, the flit size is kept at minimal with very few overheads as demonstrated from the figure 3.3. The flit gets tagged with three more bits at the least significant bit (LSB) right before entering the input port of the router to represent the right output port right before the flit gets into the router. After the flit enters the router, that output port information is used to route the flit out of the router to the direction towards the destination. The binary bit stream of the flit can be seen into different small regions which represent different kinds of information from the figure 3.3.

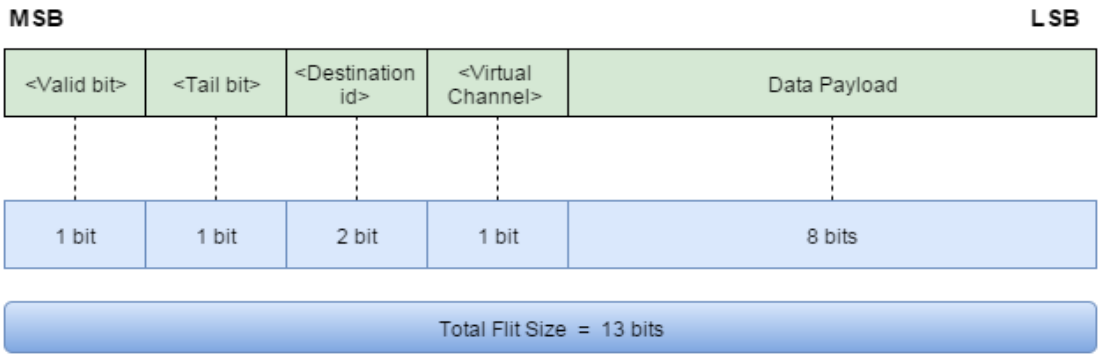


Figure 3.3: Proposed Flit Format

Valid bit

Valid bit indicates a valid flit. It is 1 bit as the flit can be valid or not. The value is set to 1 when sending a flit. This bit is discarded when the flit is inside the router. The valid bit is re-assigned to the flit before it leaves the router.

Tail bit

Tail bit indicates if a flit is the last flit of a packet. Upon receiving a flit with tail bit zero, it means that the packet is a multi-flit packet and the logic dedicates the channel for that packet until the flit with tail flit 1 is read. The value is set to 1 for single flit packet.

Destination Id

Destination Id represents the destination router id where the packet is supposed to be sent. This field is populated with a binary encoded router Id corresponding to the right router in the network. Since, the network is 2x2 mesh, there are 4 routers in total. Four routers can be encoded with 2 bits, hence the field size is 2 bits.

Virtual Channel (VC)

This field specifies the virtual channel to use in the network. The proposed design is a single virtual channel so this field is set to 0 at all times. There is no any particular advantage for keeping the virtual channel field for a single VC router. This field is kept for further future works.

Data Payload

This field specifies the actual data to be transferred. The size is kept at only 8 bits for the demonstration purpose. However, the size can be easily changed from the parameters.v file.

3.2.2 Routing Path Table

When flits are sent from the source the flit size is 13 bits as shown in the figure 3.4 as ***send_ports_2_outFlit_flit_in[12..0] .*** Right before entering the router, the two bits of destination Id (9 to 10) retrieved from the incoming flit is feed into the routing table storage block. The destination Id is used to map the 3 bit output port where the flit is supposed to be ejected from the router. Figure 3.4 is a RTL netlist generated by Quartus and it clearly shows when the flit is injected in the network, the 10^{th} and 11^{th} bits [2](i.e. [10:9]), the destination bits, are passed into the routing table and the outcome which is a 3 bit encoded output port number is added to the LSB of the original incoming flit making the flit size as 16 bits as

[2] The LSB starts from position zero, so the 10^{th} and 11^{th} bits are actually represented as a vector [10:9] in HDL.

in_ports_0_outRoutedflit_flit_in[15..0]. Please refer to appendix B for a more clearer and bigger diagram.

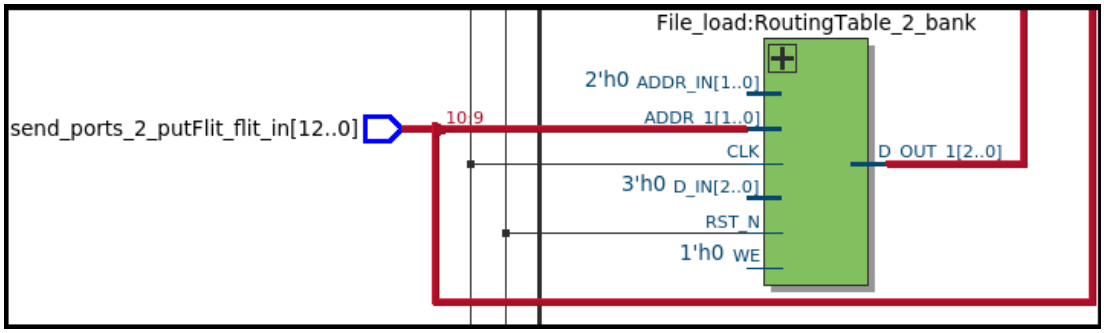


Figure 3.4: RTL netlist view of a flit entering the router

The routing table is a small RAM memory that stores output ports. The destination Id is used as an address to read the memory as seen in the figure 3.5 . Upon reading, the designated output port is send out as an output.

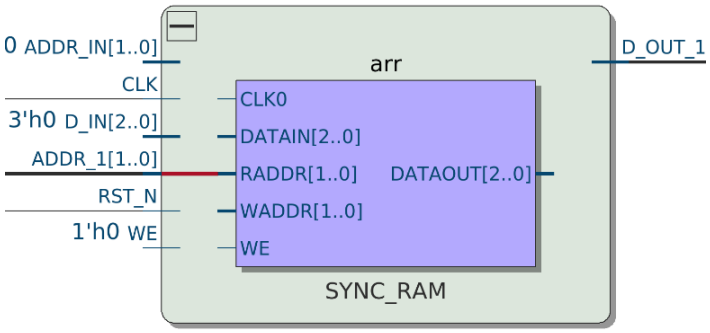


Figure 3.5: Routing Lookup table block (from RTL netlist)

Since there are four routers, four different routing tables are required, one for each router. The routing information is first stored as a hex file which is then read and stored in the memory. This makes it easy to change the routing without actually changing the code. Table 3.1 shows router to output port mapping in the hex file which is used to route the packets. R0 refers to router with Id 0 and R1 refers to router with Id 1 and so on.

Table 3.1: Router to output port mapping in the hex file

RoutingTable_0 (hex)	RoutingTable_1 (Hex)	RoutingTable_2 (Hex)	RoutingTable_3 (Hex)
R0 → 0 (send to port 0)	R0 →1	R0 →2	R0 →1

R1 →3	R1 →0	R1 →3	R1 →2
R2 →4	R2 →1	R2 →0	R2 →1
R3 →3	R3 →4	R3 →3	R3 →0

3.2.3 Packet Routing

Since the routing path is determined after the flit leaves the source and is not encoded into the packet header, the routing algorithm is a distributed deterministic routing. This allows the packet to only keep information about the final router destination Id for the routing. The destination Id is then used to map it to the right output port of each router using the encoded router and port number. Table 3.2 lists the router number and port number with their respective codes.

Table 3.2: Router Id and output port number with their respective encoded code.

Destination ID (router ID)	Code	Output Port	Code
0	00	0	000
1	01	1	001
2	10	2	010
3	11	3	011
		4	100

Each input port needs a routing look up table for that particular router. So, it requires five instances of routing table for each router and since we have 4 routers there are 20 routing tables. Since, k number of bits can encode maximum of 2^K number of ports, the word length in the memory is only 3 bits for 5 port router. Since, the word length is only 3 bits, the design uses the distributed memory rather than the block RAMs which maps very efficiently on the FPGA making better use of the resources. Block RAMs are limited resources and should only be used for larger storage.

3.2.4 Input Channel

After tagging the flit with 3 bit output port, the new flit size is 16 bits as shown in the figure 3.6.

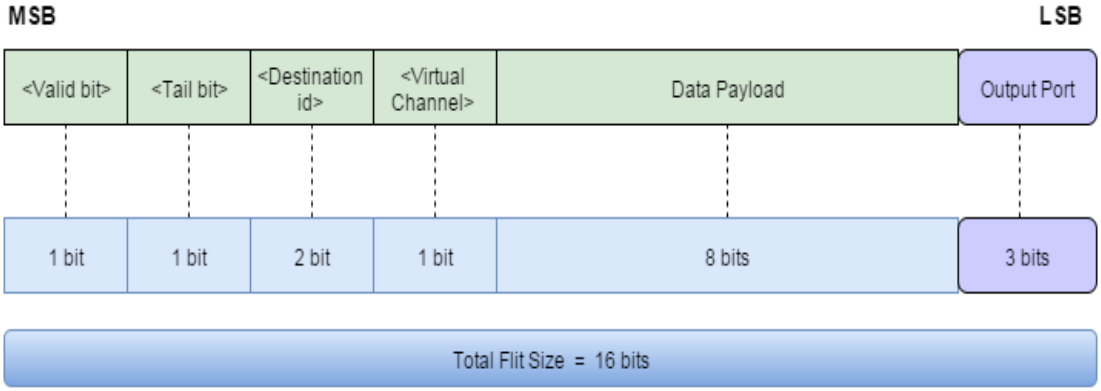


Figure 3.6: Flit size after tagging with output port

After receiving the incoming flits, the output port information i.e. first 3 bits[2..0], is retrieved from the newly tagged field of the flit and sent to OutPortFIFO Block to buffer the information for arbitration and allocation of the port as shown in the figure 3.7. Out of the remaining part of the flit i.e. ***[15..3]*** (bits 3 to 15), the 16th bit at the MSB which is a valid bit is discarded and the rest are send to the respective FIFO buffers of the input ports. Hence, the flit size heading towards the FIFO buffer is of 12 bits size ***[14:3]*** as shown in the figure 3.7 and figure 3.8***.***

The Input channel can be taken as a combination of buffers for storing output port information and the flit itself.

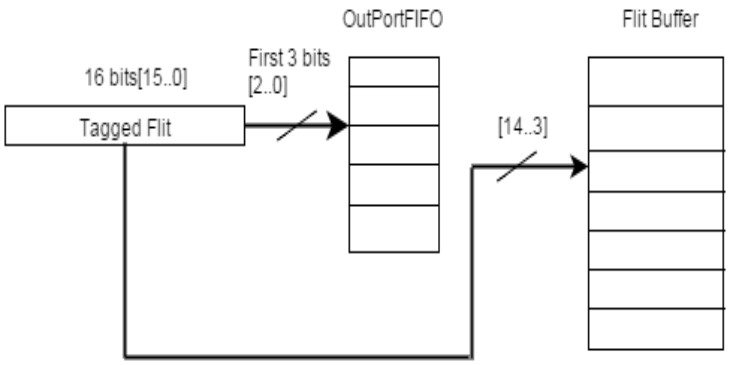


Figure 3.7: Flit after entering the input channel of the router

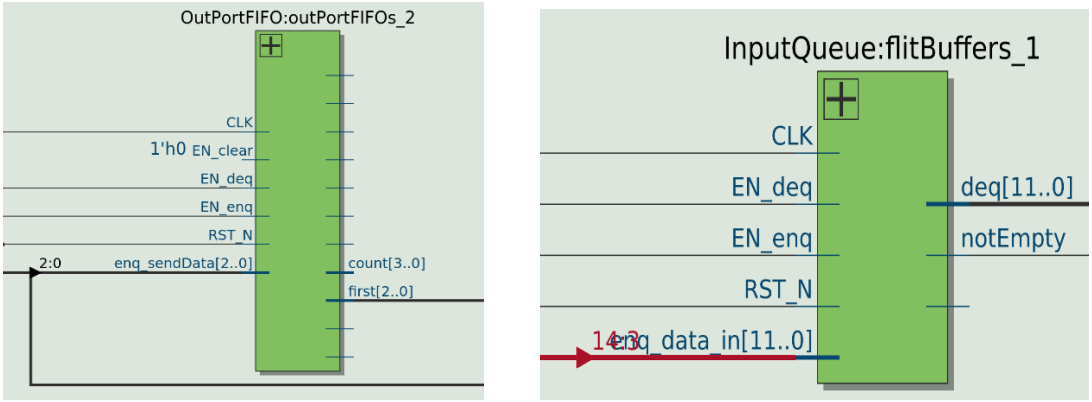


Figure 3.8: RTL schematic of flit entering the router

3.2.4.1 OutPortFIFO buffer

When the flits are injected into the router, the first 3 bits representing the output port are stored into the FIFO buffer. Those output port encoded bits are then read from the buffer and are computed to generate grants[3] as a result of arbitration so that the corresponding flit can be ejected from the router. After the requests are placed for the arbitration by the flit, the winner port gets the grant and the corresponding flit is read out from the router.

As indicated by the name itself, FIFO (First In First Out) works on first-in, first-out basis. The data that is stored first into the memory is the first one to be read out of the memory as shown in the figure 3.9. The data is stored in an array of memory of certain width (word length) and buffer depth. The place where the data is written is called the 'head' and the region from where the data is read is called 'tail'. There are two pointers namely head pointers and tail pointers which points to the head and tail region of the memory respectively.

For the OutPortFIFOs the word size is 3 bits and buffer depth is 8 since the flit buffer depth is also 8. This FIFO buffer utilises the distributed memory (MLABs) instead of the block memory. Block memory are dedicated prewired resources and are not as abundant as distributed memory. Using block memory for small buffers is generally

[3] Grants are the matched resources which are output ports for the requesting flits. It is further discussed in sub section 3.2.6 under arbitration and allocation.

not a good utilisation of the resources because block memory are used for larger memory storage.

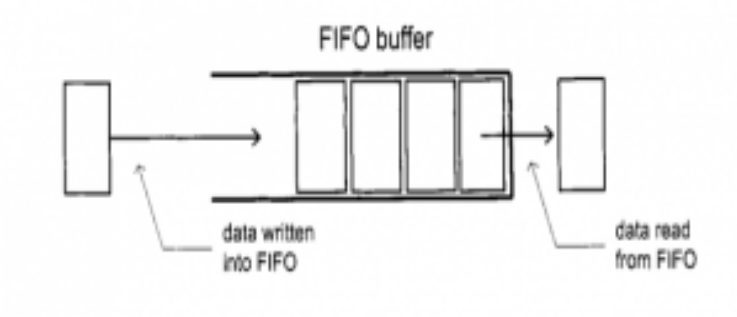


Figure 3.9: FIFO buffer [35]

The FIFO memory can be viewed as a circular buffer with a read and write pointers pointing to the memory address to write or read respectively.

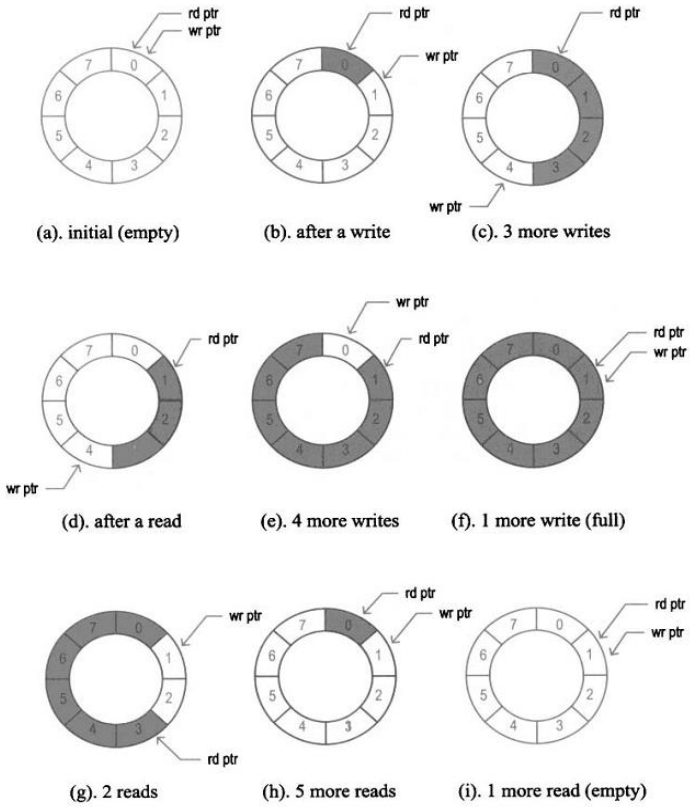


Figure 3.10: Different stages in FIFO buffer [35]

As evident from the figure 3.10, initially both read and write pointers are at the position 0 which indicates that the buffer is empty and there is nothing in the buffer to read. After writing data into the memory, the write pointer moves correspondingly to point to the next empty region in the buffer. As more and more data are written into the memory, eventually, the write pointer reaches the last empty region in the buffer and after that both read and write pointers are pointing in the same region. This indicates that the buffer is full as shown in the figure 3.10 (f).

When the data are read, the read pointer reads the first data that was written and subsequently keeps moving down the buffer stack. When the read pointer finally catches up with the write pointer and are pointing to the same region that means the buffer is now empty and all the data have been read out from the memory as show in figure 3.10(i).

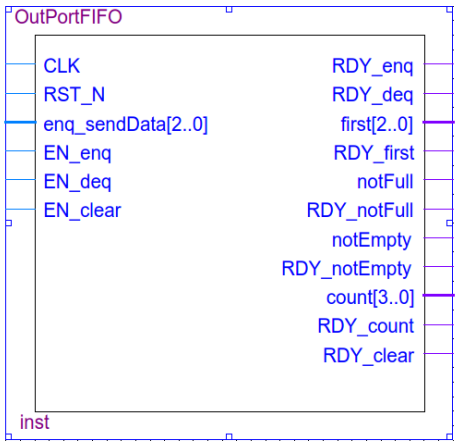


Figure 3.11: OutPortFIFO Block Diagram in Quartus

The block diagram in figure 3.11 shows that the FIFO block has ***CLK, RST_N, enq_sendData[2..0]*** (3 bit input bus), ***EN_enq*** (Enable enqueue; flag that signals that it is ok to receive new data), ***EN_deq*** (signals that it is ok to dequeue the data) and ***EN_clear*** as inputs and number of outputs indicating different stages of the buffer. The ***first[2..0]*** is the output read from the buffer. The ***count[3..0]*** is one bit more than the number of bits for buffer depth because with 3 bits the maximum count can only reach up to 7 whereas there are eight addresses to count.

3.2.4.2 Flit Buffer

Unlike OutPortFIFOs, flit buffer uses the dedicated memory blocks to store the flits. The flit buffer is designed as a single port synchronous RAM buffer with buffer depth of 8 and word length of 12 bits to accommodate the incoming flits as shown in the figure 3.12. Each flit buffer is connected to each input port. Hence, five FIFO buffers for five ports in each router.

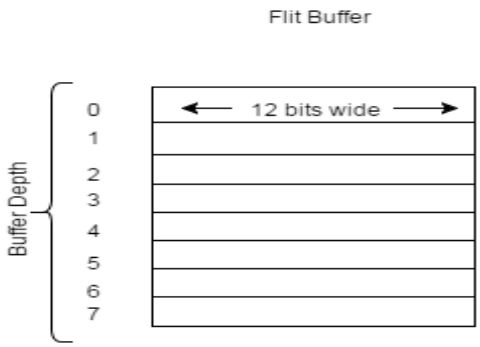


Figure 3.12: Flit Buffer Dimension

The compartments of the buffer are the memory addresses which are required to write the data into the memory or read the data from the memory. The memory address is of 3 bits which can have values from 0 to 7 in decimal form.

The buffer has two pointers, namely ***ADDR_IN*** and ***ADDR_OUT*** , pointing to the memory address to write the data into the memory and read the data from the memory respectively as shown in figure 3.13. Both of the pointers are 3 bits which can locate all the memory addresses since the memory address is 3 bits wide. ***WE*** (write enabled) is asserted high during data storage.

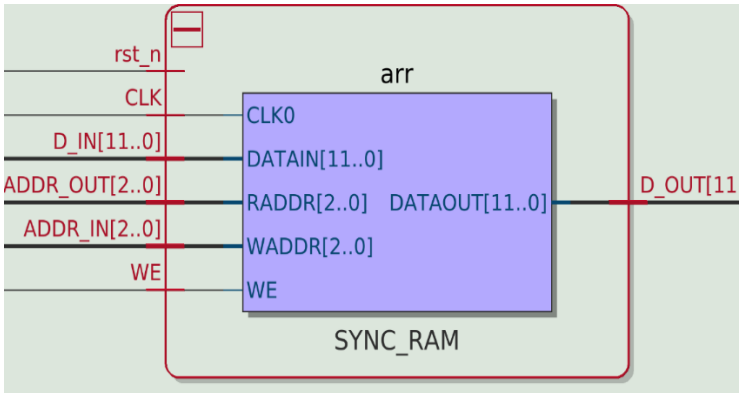


Figure 3.13: Flit Buffer core memory RTL synthesized block.

The incoming flit size at the flit buffers is 12 bits even though the original flit size is 13 bits. The valid bit is discarded for now and will be attached to it as the flit leaves the router. When the flits are stored in the FIFO, the notEmpty flag and the output port number from the OutPortFIFO are used to compete to go first. The ***ADDR_OUT*** value is assigned after the arbitration and allocation. This handles which address is to be read from the memory.

Since each input port has one flit buffer and each buffer can send data to any of the five output ports, this results in the five possible output ports from each flit buffer and total of 25 possibilities. All these 25 possibilities are put into a stream of 25 bits and are fed into RouterAllocator block where the grant is generated as a result of arbitration which will be discussed in the later chapter for arbitration and allocation.

3.2.5 Credit Based-Flow Control

During the transmission of data from one router to another, there has to be some sort of data flow management to control the flow of the data to allow efficient handling of the data and keep the transmission at a pace. According to Dally and Towles [36], credit based flow control keeps the counter and provides all the necessary information to the sender about the buffer space availability and knowledge to start, stop or resume the transmission. The sender always keeps the record of the available buffer slots in the receiving router so as not to cause packet loss by sending packets to already filled up buffer [36].

The slots in the buffer are called credits, hence the name credit based flow control. Credits are stored at the sender's side [37]. Each credit takes a slot in the buffer and is one word long. The credit count has to be more than zero to be able for sender to send flits. Every time the sender sends a flit, the credit is decremented showing that there is one less slot in the buffer and the information is sent to the sender. Similarly, with every ejection of the flit from the buffer, the credit is incremented and the information is again passed to the sender. So, each router has a credit information coming into the router from other receiving router and similarly, credit information is sent to the other connected sender router. The credit-based flow control can be better understood with the figure 3.14.

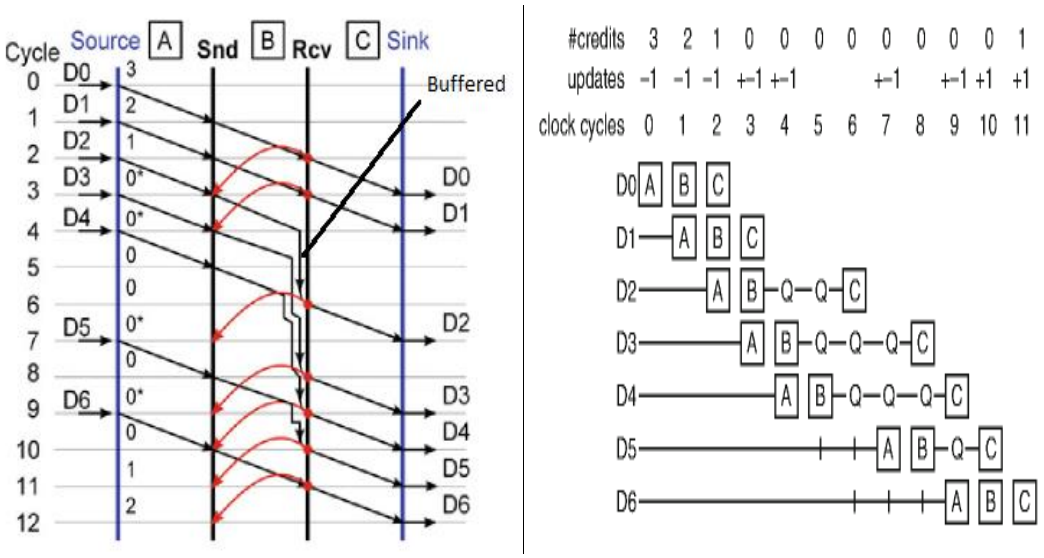


Figure 3.14: Credit based flow control between sender and a receiver [37]

Figure 3.14 (a) and (b) shows the flow control mechanism displayed in the temporal and spatial domain respectively. From figure 3.14, in the beginning, source A has 3 credits available, meaning that the receiver has 3 slots left in the buffer. Since, it's a non-zero value, the sender can send the data. When the data is consumed by the sink or ejected out from the receiver buffer, the credit counter is updated and result in decrement of the credit counter as shown in the figure 3.14 (a) with red lines. Even though there is a cycle delay in credit update, they are consumed in the same clock cycle. Consumption of the credit in the same clock cycle can be seen in the figure 3.14 (a) and (b) at the clock cycle 3. Same clock cycle credit reuse is denoted by * sign.

The RouterCore block diagram from the Quartus in figure 3.15 shows the input and output ports along with the flow control ports to get the credits and put the credits. For each input and output port, corresponding flow control input and output ports are designed to allow flow control.

As evident from the block diagram figure 3.15, an input for the credit information is a 2 bit input bus, ***out_ports_<portNumber>_putCredits_cr_in*** which basically contains information about the validity of the credit and the virtual channel as shown in figure 3.16 for which the outgoing credit is intended to. Since, it is a single channel router, the virtual channel bit is always set at 0 for now.

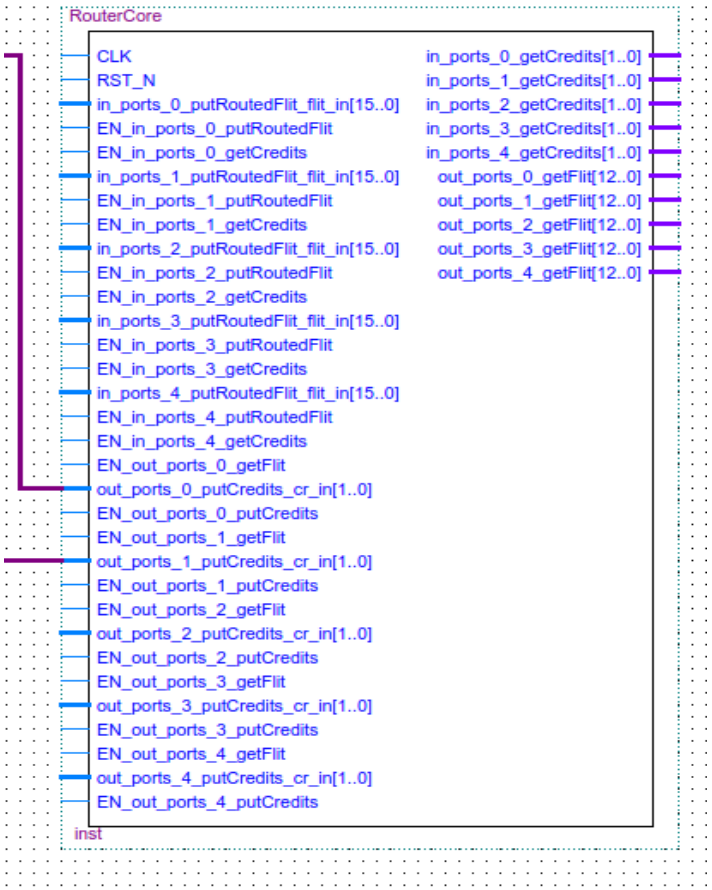


Figure 3.15: Router Core Block Diagram

Figure 3.16: Bit representation in credit information

The credit counter is 4 bit wide since we have eight slots in the buffer. With 3 bits, the counter can count only up to 7. The snippet of the code below shows that separate credit counters are declared for separate five buffers for five input ports . In the beginning the credit counter are asserted to 8 which means that the buffer is empty with 8 available slots .

```
// register credits

 reg [ 3 : 0 ] credits ;
  wire [ 3 : 0 ] credits$D_IN ;
  wire credits$EN ;
    . . . . . .
```

```
always @ ( posedge CLK )
begin
  if ( ! RST_N )
    begin
      credits <= `BSV_ASSIGNMENT_DELAY 4'd8 ;
      credits_1 <= `BSV_ASSIGNMENT_DELAY 4'd8 ;
      credits_2 <= `BSV_ASSIGNMENT_DELAY 4'd8 ;
      credits_3 <= `BSV_ASSIGNMENT_DELAY 4'd8 ;
      credits_4 <= `BSV_ASSIGNMENT_DELAY 4'd8 ;
```

3.2.6 Arbitration and Allocation

Scheduling of which flit gets to eject from the router is done with an arbitration logic. Arbitration logic checks the flit buffer and the credit availability to make decision on which flit gets the grant to traverse the switch. Arbitration always matches N requests to 1 resources[4]. Allocators on the other hand, allocates or matches number of resources to the number of requests. To generalise, allocators matches M requests to N resources. There are three basic rules when granting requests and they are [38] [36]:

- A resource is only granted when there is a request for that resource.
- Each requester can be assigned to one resource at most.
- Each resource can be granted for one resource at most.

The design implements Separable Input-First Allocator to match requests to the grants for the resources. A separable allocator, as the name suggests, separates the allocation process into two consecutive arbitration phases. In the first phase of the arbitration, which is input arbitration, the requesters select a particular resource. Then, in the next phase, which is output arbitration, an arbitration is performed again to select a winner from the incoming requests [38] [36] and matching just one request to the resource. So, the requests which wins the first arbitration may not always win the second arbitration.

[4] In allocation and arbitration, the inputs are referred as requests and outputs or available ports as resources. Input requests may also sometimes be referred as agents.

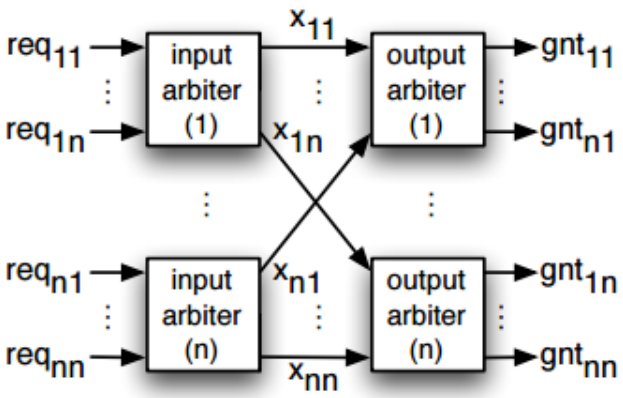


Figure 3.17: Input-first separable Allocation [38]

Figure 3.17 shows the input first separable allocation with requests and corresponding grands generated.

From section 3.2.4, we know that there are 5 flit buffers with each buffer having the possibility of sending through 5 output ports, making a total of 25 possibilities. All of these possibilities are arranged in a stream of 25 bits. This 25 bit bus is then input into the RouterAllocator block which carries separable input first allocation and generate grants or matches to the resources. The grants are further decoded to appropriate output ports which are finally assigned to the flit buffers for ejecting the flits to the right output port. The RTL synthesised schematic in figure 3.18 shows how the requests are fed into the RouterAllocator block.

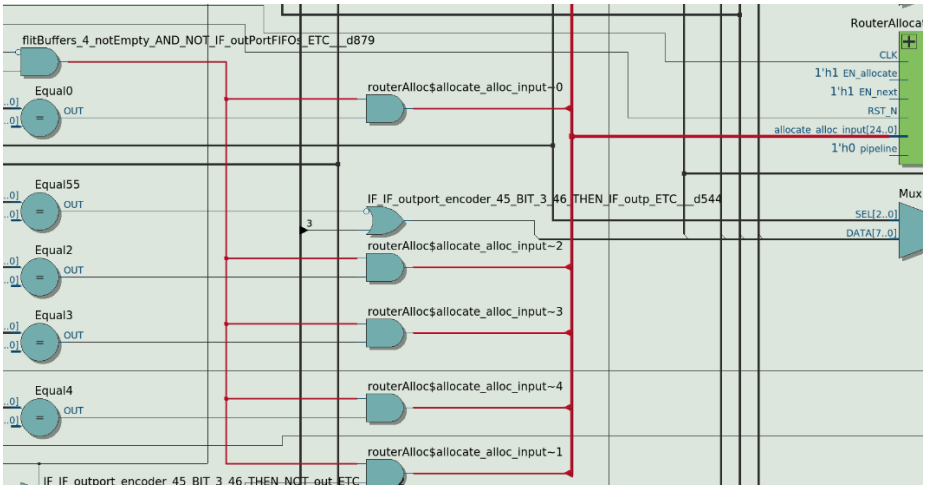


Figure 3.18: Request flow into the RouterAllocator block

Upon availability, arbitration logic sends signals to the flit buffer after choosing the desired output port to tell the buffer to send the flit to traverse the crossbar switch.

3.3 Summary

The details of the proposed router architecture has been discussed in this chapter. The router is designed such that it induces minimum delay in the router and hence minimum latency for a small network with relatively fewer traffic by reducing the number of pipeline stages to just one. Packets can be single flit or multiple flits. Each flit is 13 bits in length for a 2x2 mesh network with 8 bits data payload. Only single virtual channel is employed for the design but there is a provision for virtual channel in the flit for future works.

When the packets are sent from the cores to the local router which means router connected to the core, the destination Id bits of the flit is extracted and sent to routing computation block where the destination Id is read and appropriate output port is mapped for the flit based on the routing lookup table which are stored as hexadecimal file. That port number is then tagged to the LSB of the flit before entering the router similarly as demonstrated by Yaghini et al for TagNoC [31]. After the flit enters the router, then the tagged output port number is sent to be stored in the OutPortFIFO buffer from where the port number is used for arbitration and allocation of the output port. Concurrently, the remaining of the flit is stored in the flit buffer until their turn comes to be read out from the router. The flit buffer is 12 bits in width with depth of 8 meaning it can store eight 12 bits flits. Credit based flow control is used to control the flow of data. Credit is the availability of slots in the flit buffer. If a buffer is empty, the credit is 8 and the buffer can accept flit as long as the credit is non-zero.

Chapter 4 Verilog Modelling, Simulation (Verification) and Evaluation

This chapter discusses in detail about how the design is modelled in Verilog Hardware Description Language. Different components or modules (in Verilog) of the design are listed in a table and each component's functionality is described. Also, the integration of these different components to form a core router and a whole 2x2 mesh network is also discussed.

After the discussions on the modelling, discussion on the verification of the design using ModelSim as a simulation tool is done. A basic simulation is run to test the minimal functionality of the design. Furthermore, more tests are run to generate more results to evaluate the design. Evaluation is focused mainly on the router delay meaning the number of clock cycles taken for router to process the packet and send it to the right output port. The router design is a single stage pipelined router, hence the simulation is carried out to demonstrate that behaviour of the design. The router delay is calculated on zero load network and network with many packets transmitting all at the same time destined for the same router. This test is a priority based as the priority is given after the arbitration.

4.1 Verilog Modelling

The router is designed using the Verilog Hardware Description Language using Altera's Quartus II 14.1 software and verified using ModelSim as a simulation tool. For the design of the router, a bottom up approach is taken. The design is first broken down into many smaller blocks or components. Each component is written separately in Verilog as modules and are all integrated together to form a router. After the core router is ready, yet another module is written which is named as Network. It basically creates four instances of the router core and form a 2x2 mesh topology by connecting the inputs and outputs of the router programmatically and also introduces routing table right before each router inputs. Network module is set as the top level entity of the design in Quartus.

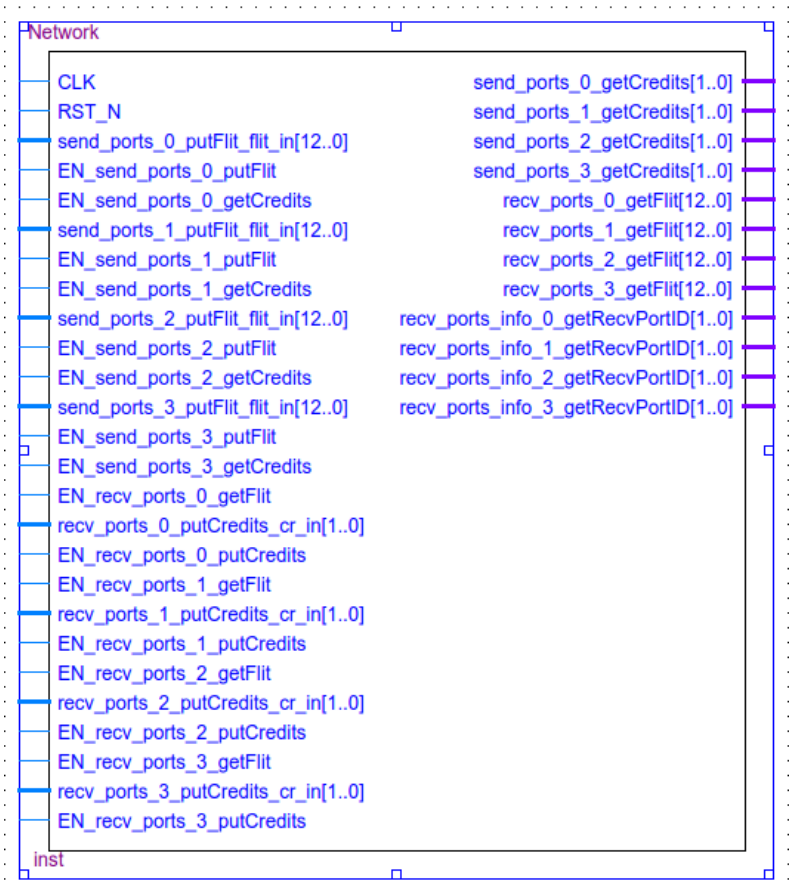


Figure 4.1: 2x2 mesh network represented in a single block diagram

Figure 4.1 is a block diagram generated from Quartus for the Network module which encompasses 2x2 mesh network with four routers connected. From the figure 4.1, ***send_ports_0_putflit_flit_in[12..0]*** and ***recv_ports_0_getFlit[12..0]*** are the input and output ports of node 0 connected to the router_0. Similarly, all the other nodes or say processing cores are connected to their respective routers with the same number i.e. node 1 connected to router 1, node 2 to router 2 and node 3 to router 3. Network topology diagram i.e. Figure 3.1 from chapter 3 helps to better understand how the I/O ports from the block diagram maps to the actual topology. Hence, there are four flit input ports and four flit output ports for four routers in the network. During the testing, packets are injected from these ports from the nodes or cores into the router and on successful traversal, packets are expected to be read from the right outputs which means from the right destination router or node.

Router is the main building block of the network. For this project there are seven Verilog modules that comprises the core router design. The table below gives a brief overview of each module and their functionality in the design.

Table 4.1: Design Files of a core router

File Name	Description
parameters.v	Contains information about the different parameters like number of ports, number of virtual channels, number of nodes, number of routers etc. The file is imported in other modules like testbench to inherit the design parameters.
Registers.v	This is a storage that creates an array of RAM memory using the address width and data width which can be assigned at the time of instantiation.
InputQueue.v	This module creates a Flit Buffer by instantiating Register.v with appropriate address and data width to accommodate incoming flits. In addition, extra wires are created such as for head pointers and tail pointers.
OutPortFifo.v	Creates small distributed style RAM storage using MLABs and not dedicated block RAMs. Creates 8 instances of 3 bit wide storage units to store 3 bit output port numbers.
InputArbiter.v	This module forms a part of Separable-Input first allocator where there is an input arbiter followed by an output arbiter. It accepts 5 bit requests for each output port from all the input ports who want to send to a particular output ports. A grant is given to just one bit from each input port and further arbitration is carried out at the output arbiter end.
OutputArbiter.v	After input arbitration, the winners are again competed in the output arbitration to allow grants.
RouterAllocator.v	This modules combines InputArbiter.v and OutputArbiter.v to form a separable input first allocator. This module collects all the requests from all the input ports and send it to the input arbiter followed by output to generate grants.

Output_encoder.v	This module encodes the 3 bit output port number that is used to map with the winner of the arbitration.

Network.v instantiates the RouterCore module and creates a network in a 2x2 mesh topology fashion. Apart from the RouterCore module, Network module consists of routing table which fetches the routing information from the hexadecimal file.

Table 4.2: Modules used in the Network module

File Name	Description
RouterCore.v	Forms the router in the network. Four instances of the router is created and are connected such that it forms a 2x2 mesh topology.
File_load.v	This module is used to import routing information from the hexadecimal files. This module is introduced before each of the router input port. The destination Id from the incoming flit is fed into this module and it gives a corresponding output port which is then attached to the LSB of the incoming flit making the flit size as 16 bits.
RoutingTable_0.hex	RoutingTable hexadecimal file contains information about which output port to go to for a given destination router Id. This routing table is for router_0. Five instances of this table is created for five input ports of the router.
RoutingTable_1.hex	This routing table is for router_1. Five instances of this table is created for five input ports of the router.
RoutingTable_2.hex	This routing table is for router_2. Five instances of this table is created for five input ports of the router.
RoutingTable_3.hex	This routing table is for router_3. Five instances of this table is created for five input ports of the router.
Testbench.v	This testbench file is used to verify workings of the network. It verifies that the router can successfully route the flits to the right output towards the destination router.

4.2 Simulation and Verification of the Design

After modelling of the design, the design is verified using Mentor Graphics ModelSim Starter Edition 10.3c simulation tool. Initially a minimalistic simulation was run to verify the working of the design independently such as memory buffer, FIFO memory and so on. Then after the integration of all the design components, packets of variable flit number are injected from different nodes into the router targeted at different routers via the simulation.

The following part of the chapter talks about the simulation results that help to verify the workings of the design. Since, FIFO buffer is one of the most important parts of the design, we will look at the simulation of FIFO buffer first then the overall network simulation. FIFO buffer testbench is discussed since it is relatively simple to understand and also helps the reader to better understand the overall network design simulation waveform discussed later which can be a little bit convoluted.

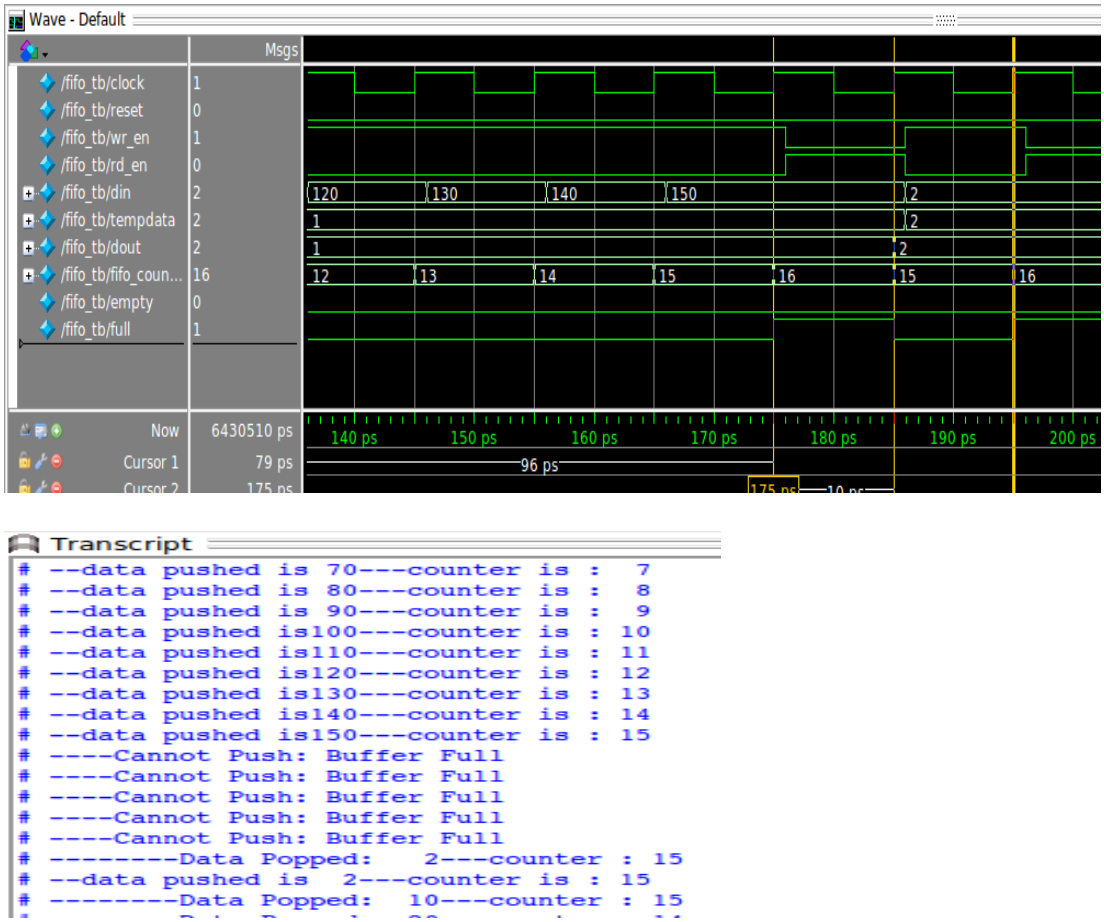


Figure 4.2: FIFO buffer simulation waveform and console log

Figure 4.2 shows the waveform generated from the testbench for the FIFO buffer. The testbench is written before the actual network design. Hence, the parameters are not same as for the proposed network design, however, the concept and the workings of the buffer is the same. The buffer is 8 bits wide with depth of 16.

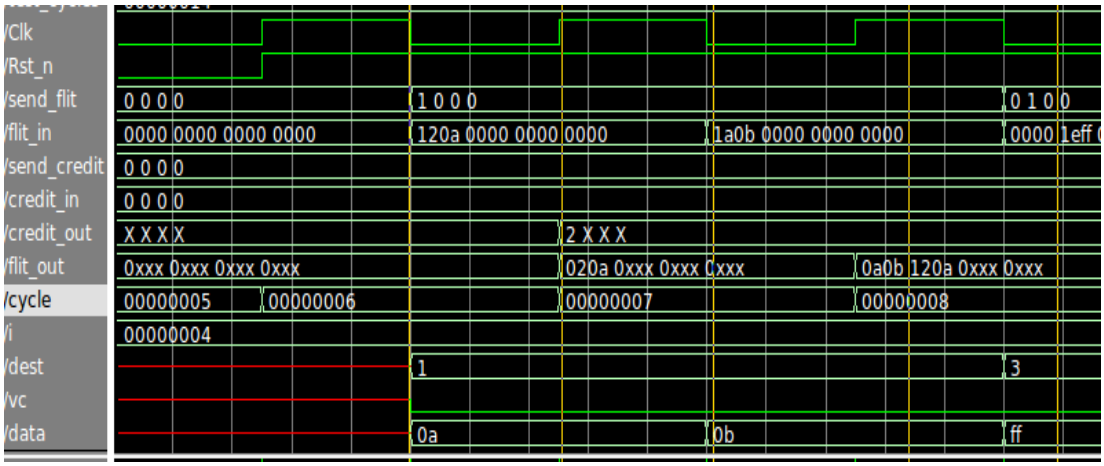
The changes are asserted at the rising edge of the clock. A certain part of the waveform where the FIFO gets full is selected to validate the design of the FIFO buffer.

The values in the waveform are all shown as unsigned decimal representation. The changes in the waveform are logged into the console.

When the wr_en flag is set to high, the buffer accepts input data and writes into the memory. Every time the buffer stores a word into one of its slots, the counter value increments. Since, it's a 16 slot buffer, when the counter goes to a value of 16 it means that the buffer is full and there are no more slots available. At 165 ps, the buffer receives data value of 150 at the counter 15 and at the next rising edge of the clock, the counter value 15 is incremented to 16 since the buffer received value of 150. When counter is read to be 16, the buffer is full so the wr_en is pulled down to 0 and buffer full flag is asserted as high. After that, buffer full message is shown in the console log. When the data is read from the buffer, the data written after the last read or popped out data, is popped out from the buffer demonstrating the FIFO nature. The testbench helps to visualise the changes happening at every clock cycle and helps to verify the working of the design. Note that, the data are written and read at the rising edge of the clock since the FIFO is triggered at positive edge of the clock.

Once, all the components are integrated, the simulation is run to verify the working of the whole network. In this next simulation, the main design aspect that is targeted for verification is the data transmission to the right destination router and subsequently right node.

For this simulation, a testbench file called testbench.v is written to inject packets into the network with an arbitrary destination Id. Testbench works as a dummy core that injects the packets into the network and also acts as a dummy core that receives the packets. Validity of the design is confirmed by checking the packet received from the destination router of the network. Also, the traversal of the packets from one router to another and then to the destination router can be verified from the waveform if there is an intermediate router to reach the destination router.




```
# ---- Performing Reset ----
# @   6: Injecting flit 120a into send port 0
# @   7: Injecting flit 1a0b into send port 0
# @   8: Injecting flit 1eff into send port 1
# @   8: Ejecting flit 120a at receive port 1
# @   9: Ejecting flit 1a0b at receive port 1
# @ 10: Ejecting flit 1eff at receive port 3
# @ 11: Ejecting flit 1eff at receive port 3
# @ 12: Ejecting flit 1eff at receive port 3
# @ 13: Ejecting flit 1eff at receive port 3
# @ 14: Ejecting flit 1eff at receive port 3
# @ 15: Ejecting flit 1eff at receive port 3
# @ 16: Ejecting flit 1eff at receive port 3
# @ 17: Ejecting flit 1eff at receive port 3
```

Figure 4.3: Simulation of Packet traversal in the network with subsequent logging.

Figure 4.3 shows the waveform of the simulation of a packet with 2 flits. Each flit is 13 bits in size represented with 4 hex characters for four routers core.

A head flit with hex value 120a is injected into the router 0 with destination Id 1.

120a in binary is 1 0 01 0 00001010 (segmented as valid, tail, destination Id, VC and data payload respectively from MSB to LSB). From the flit value, we can tell that the flit is not the tail flit, hence the traversal path is dedicated for the this flit until the tail flit is read. The details of the flit structure has been presented in the Figure 3.3 from Chapter 3. At the next rising edge of the clock after the flit is injected, the flit is stored into the flit buffer in the router_0 as 12 bits with flit value 020a as evident from the figure 4.3 in clock cycle 7. The valid bit is discarded as discussed in the sub-chapter 3.2.4 of chapter 3. In the waveform, it is shown in the flitout section but since, there is no valid bit, it implicitly mean that the flit is stored in the flit buffer. Since, the destination Id is 1, the flit has to traverse to router_1 from the current router_0. At the next rising edge of the clock, we can see that the flit has been placed into the output port allocated for router_1. So, a flit takes a total of 2 clock cycles after being injected to the router and sent to the right output port as shown in a flow diagram in figure 4.4.

And the router takes a further 2 clock cycles to go from the input channel of the router_1 to the right output channel for the node. So each router takes 2 clock cycles for a flit when there is no other transmission happening at the time. Figure 4.4 shows what happens at each clock cycle in the router.

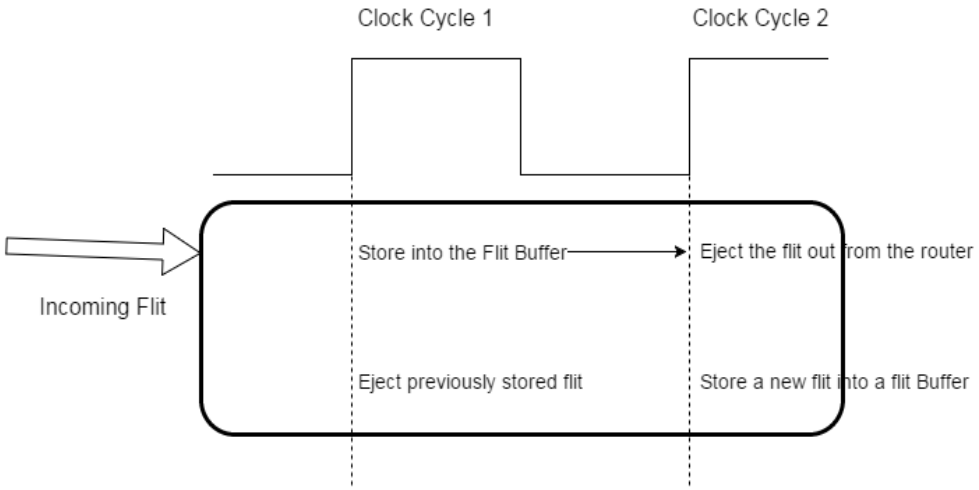


Figure 4.4: Flit flow diagram in a router

Also from the console log from figure 4.3, at clock cycle 6, flit is injected into the router and at clock cycle 8, the flit is ejected from the router. This verifies the number of clock cycles required for each flit in the router is 2.

There is another successive flit with the value 1a0b which is the tail flit of the previous flit.

1aob = 1 1 01 0 00001011 in binary representation (segmented as valid, tail, destination Id, VC and data payload). We can see that when the 120a flit has already placed into the output port directed for router_1, tail flit 1a0b is still in input flit buffer of router_0 demonstrating the wormhole switching nature of the design. Please refer to *Appendix C* for explanation on Wormhole Switching. The tail flit follows head flit at an interval of 1 clock cycle. From the console log from figure 4.3, the tail flit is injected into the router at clock cycle 7 and is ejected from the router at clock cycle 9 verifying that it takes 2 clock cycles for each router.

The simulation is run at an ideal situation where there is only one node sending data at a time. Regardless of being an ideal situation, the simulation successfully demonstrates the working of the network. The design can successfully map the router Id from flit to the right router. The packets can traverse from one router to the

destination router successfully in a single pipeline stage as proposed because as demonstrated from the simulation results the flit is injected into the router in one clock cycle and is ejected from the router through the right output port in the next clock cycle. And the packet can be single or multi flit.

4.3 Results and Evaluation

The simulation results from the previous section validates the working of the network as the packets can successfully reach the destination in predicted number of clock cycles. Once the design is verified, then a number of tests are carried out to analyse the router delay under two scenarios as a reference. There are many more scenarios with even more test cases which is simply not possible due to limited time. So, two main cases are chosen as a reference tests for to evaluate the latency of the design.

The tests can be divided mainly into two categories:

- **Without Load Test**
  Without load test refers to when only one router is sending packets to another router without any other packets competing for the same output channel. This is an ideal situation that verifies the design and also helps to evaluate the best latency.
- **With Load Test**
  With load test refers to when two or more flit buffers want to transmit the packet to the same output channel at the same time.

4.3.1 Without Load (zero Load) Test

As mentioned earlier, zero load test is an ideal situation when there is only one node sending packets to another node in the network. To keep the simulation simple and easy to understand, all the packets set as single flit packets. Since, the router operates in the flit level, if a flit takes 2 clock cycles to traverse from one router to another then a packet with 3 flits will take 6 clock cycles.

Below is a list of tests undertaken to calculate the clock cycles to reach the destination for a zero load network. The results obtained from the tests conducted for zero load are logged in the table 4.3. From the Table 4.3 it is clear that the number of clock cycles for a node to transmit to a router directly connected to it is 2 and if there is an intermediate router in between the source and the destination, then it takes a further one more clock cycle to traverse through the intermediate router. It takes 3 clock cycles and not 4 to traverse through an intermediate router because in the first clock cycle flit is injected to the router and stored in the flit buffer, in second clock cycle the flit is ejected out of the router and in that same clock cycle the flit traverses to the next router and is then stored in the next router's flit buffer and finally in third clock cycle, the flit is ejected from that intermediate router and it reaches the destination router. It will further take one more clock cycle to reach the node or core connected to it but we are only looking at the destination router in these tests.

Table 4.3: Router delay for zero Load network

Flit traversal direction (Source Router Id to Destination Router Id)	**Number of Clock cycles to reach the destination router**
R0→R1	2 (adjacent router)
R0→R2	2
R0→R3	3 (one router in between)
R1→R0	2
R1→R2	3
R1→R3	2
R2→R0	2
R2→R1	3
R2→R3	2
R3→R0	3
R3→R1	2
R3→R2	2

Since the design is implemented as a 2x2 mesh topology, there can only be one intermediate router at most between the two distant routers. Hence, the total clock cycles at the worst case can only be 3 clock cycles for a zero load network. However, if the network is scaled with more routers then the max clock cycles increases depending on the routing algorithm used but the number of clock cycles each router takes remains the same. Since, this design uses routing lookup table, the path can be determined so that it takes least number of possible clock cycles to reach from one router to another in scaled up network.

The waveform from the figure 4.3 from previous section demonstrates without load transmission of flit in the network.

4.3.2 With Load Test

This test is carried out to see the behaviour of the router when there are more than one input channel trying to send flits to the same output channel. This is a priority based as only one request gets selected for the output port. Arbitration is carried out in case of multiple transmissions. This can occur when a router wants to send a packet from more than one input channel destined for the same router or same output port. Which input port gets the priority is decided by the arbitration logic. The winner is assigned that output port and then the remaining requests again compete for the output port.

In this test, testbenches are written such that all the routers send a single flit for a same destination router. The flit is read at the destination router and the number of clock cycles it took for the flit to reach the destination router is recorded and compared. For zero load network, if a router is directly connected, it takes 2 clock cycles and if there is an intermediate router it takes 3 clock cycles. However, with load, there is competition to get the output port if there are more than one input channel trying to send the flits through the same output port.

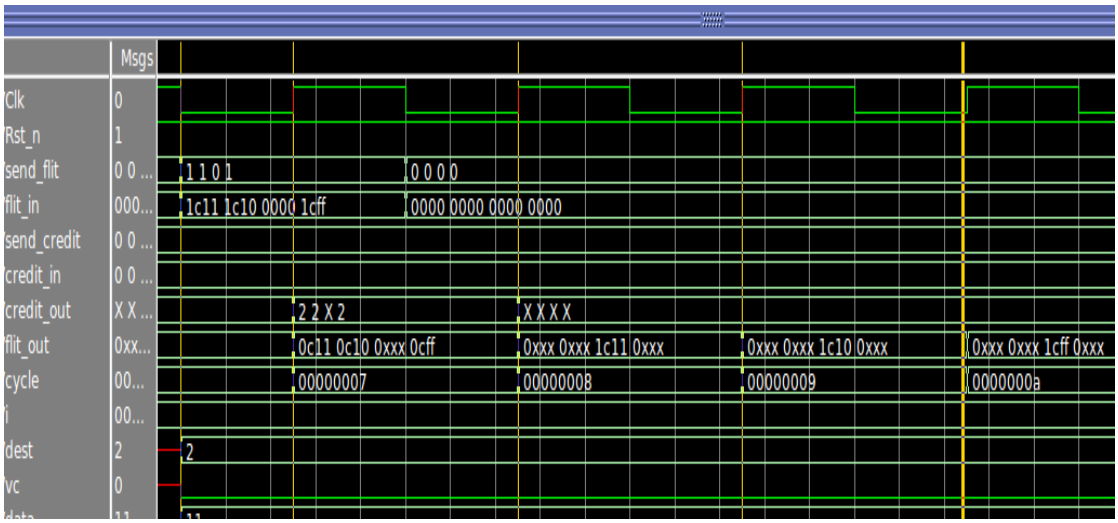
As evident from the Table 4.4, the number of clock cycles has increased in this scenario. It is because of the fact that now it is not an ideal situation and there are more than one input channels trying to get access of the output port to go to the destination router at the same time. Collision in this scenario is avoided by setting priority and allowing only one transmission at a time from a particular output port.

Table 4.4: Router delay with concurrent input from all the inputs

Source to destination (Source Router Id to Destination Router Id)	Number of Clock cycles to reach the destination Router
R0→R1 **R2→R1** **R3→R1**	2 4 3
R1→R0 **R2→R0** **R3→R0**	4 2 3
R0→R2 **R1→R2** **R3→R2**	2 3 4
R0→R3 **R1→R3** **R2→R3**	3 2 4

From the tests conducted above, the clock cycle can go up to 4 clock cycles in this scenario. Here, we are not dealing with more than one flit being stored in the flit buffer or multi-flit packets. There are simply too many test cases and due to lack of time, it is simply not possible and hence only the tests that can be taken as a reference are conducted. The delay can occur more if there is any disruption in the network and the incoming flits are stored in the flit buffer. For example, if a flit is stored in the buffer and there are already 4 more flits stored previously then the flit has to wait another 4 clock cycles to get its turn to be ejected out from the buffer. Also, if a packet consists of 5 flits then the packet will take 5 times more clock cycles to reach the destination compared to single flit packet. Once, a head flit of a packet wins the arbitration then the rest of the packet flits are automatically assigned the channel until the tail flit is received.

The figure 4.5 shows how flits from different routers all targeted for a particular router traverse around the network and reach the destination router at different clock cycles.




```
# ---- Performing Reset ----
# @  6: Injecting flit 1c10 into send port 1
# @  6: Injecting flit 1cff into send port 3
# @  6: Injecting flit 1c11 into send port 0
# @  8: Ejecting flit 1c11 at receive port 2
# @  9: Ejecting flit 1c10 at receive port 2
# @ 10: Ejecting flit 1cff at receive port 2
```

Figure 4.5: Test with Load. Simulation waveform with subsequent console logging.

From the log file and waveform from figure 4.5, three different flits from router 0, 1 and 3 are all destined for router 2 injected at clock cycle 6. The flits are injected all at clock cycle 6, but they are read from the same output port at different clock cycles. Ideally it would be 2 clock cycles but now there are competition to access the output port. The number of clock cycles needed for flit from router 1 to send flit to router 2 is 3 clock cycles and for router 3 is 4 clock cycles due to waiting for the output port to be free since all the flits are traversing via the same channel. The snippet of the testbench code is listed in *Appendix D* which shows how the flits are injected into the router.

4.4 Design Evaluation

So far we have evaluated the router delay based on the simulation run under zero load and with load scenarios in the network. This section will discuss the overall design and make comparisons with the existing router micro architecture to evaluate the strengths and weaknesses of the design. The main focus of the design is to keep the router delay and correspondingly the latency to minimum.

One of the most common approach to reduce the latency of a router micro architecture is to reduce the number of pipeline stages in the router [39]. A generic NoC router consists of 4 pipeline stages within the router which generally are routing computation (RC), Virtual Channel Allocation (VA), Switch Allocation(SA) and Crossbar Traversal (CT) [39] as shown in the figure 4.6 .

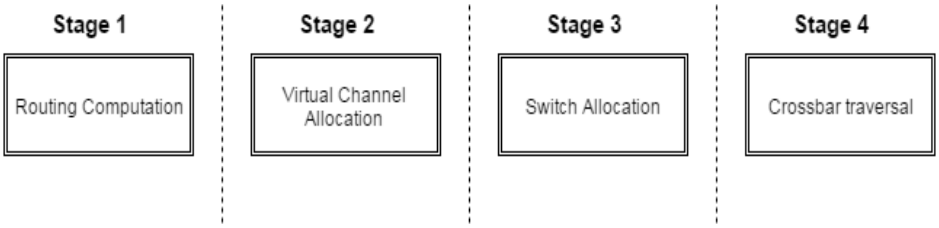


Figure 4.6: Pipeline stages of a generic NoC Router [39]

Each stage in the pipeline takes one clock cycle and operate in a serial manner which means that a simple router takes at least 4 clock cycles to process the incoming flit and send it to the right output port. However, with the proposed router architecture, the router employs a single pipeline stage which results in flit coming into the router and flit read from the router in the same stage. Figure 4.7 helps to visualize different stages in the single pipeline stage router and also how parallelism is achieved in a single stage.

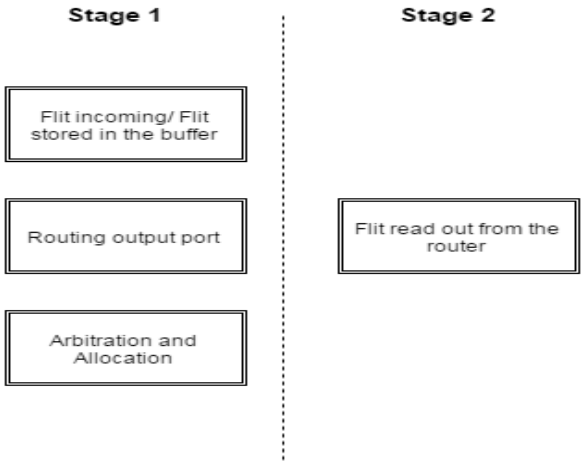


Figure 4.7: Parallelism achieved in single stage pipeline router

The single stage pipeline is achieved by running different modules simultaneously in a parallel manner as shown in the figure 4.7. Since, the proposed router uses deterministic routing with routing computation outside the router before a flit enters the router, the routing computation is not needed inside the router. When the routing computation is done, the whole flit is not used, but just the destination Id bit is

extracted and fed into the routing computation block. This allows to achieve parallelism because as the flit is going into the router, part of it goes for the routing computation and the output port number as result of routing computation, is tagged to the flit right before it enters the router. Similarly, after the flit enters the router, flit is divided into two parts and are processed separately in parallel manner. One part, which is the tagged output port Id, is sent for arbitration and allocation, and simultaneously at the same time the actual flit is stored into the buffer until it is time to be read out from the buffer. In the next clock cycle, one of the flits win the arbitration for a particular output port and the flit is read out from the output port. Hence, the latency is reduced since each router now has reduced pipeline stage which results in more operation in a single clock cycle. In comparison to general NoC router which normally takes about 4 clock cycles, the proposed architecture only takes 2 clock cycles at an ideal case where there are no load in the network.

The figure 4.8 and 4.9 shows the number of clock cycles required during the packet routing over the network for generic four staged and the proposed single stage pipelined router respectively. The clock cycles are shown at each stage for the scenario where a flit is sent from source router R0 destined for destination router R3 with two intermediate routers. The scenarios make assumptions that the network is zero load meaning only one router is sending flit across the network.

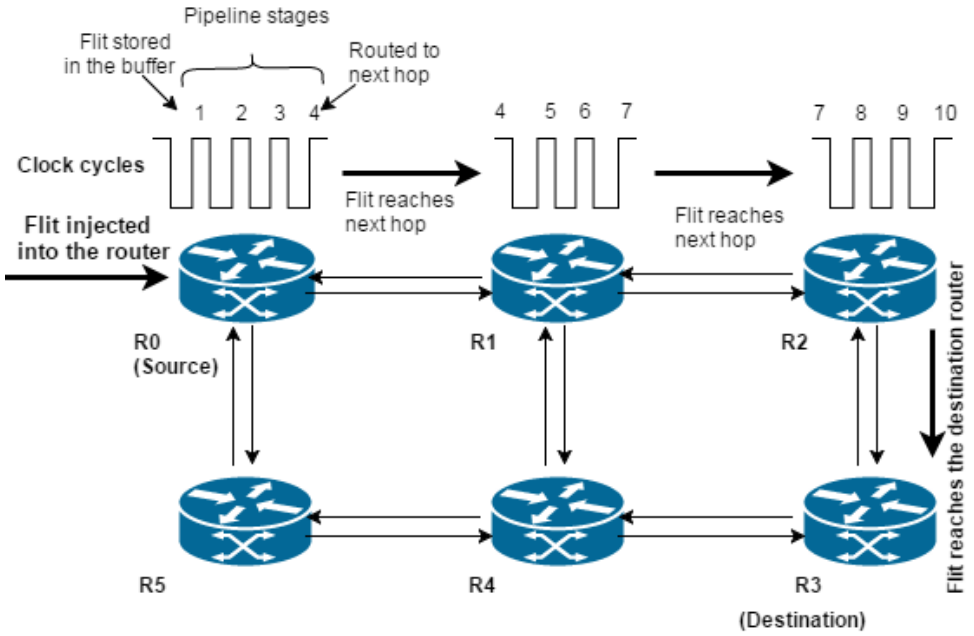


Figure 4.8: Clock cycles requirement assumption for four pipelined stage router in a 2x3 mesh network for zero load

Figure 4.8 shows the number of clock cycles required for generic four stage pipelined router where each router takes four clock cycles to process a single flit. As demonstrated by the figure 4.8, the flit is injected and are stored in the flit buffer of first router in clock cycle 1. At clock cycle 2 and 3, the flits go through virtual channel allocation and switch allocation stages and at cycle 4, the flits are finally ejected from the router via right output port. Given the routers are tightly coupled together, the flit can be stored in the next router at the same clock cycle as shown in the figure. Similarly, the same stages happen in the next router and only at the 10th clock cycle the flit reaches its destination. Hence, it takes 10 clock cycles for a four stage pipelined router. In real scenarios, there will always be additional router delay due to traffic congestion, limited buffer space and so on resulting in even larger router delay.

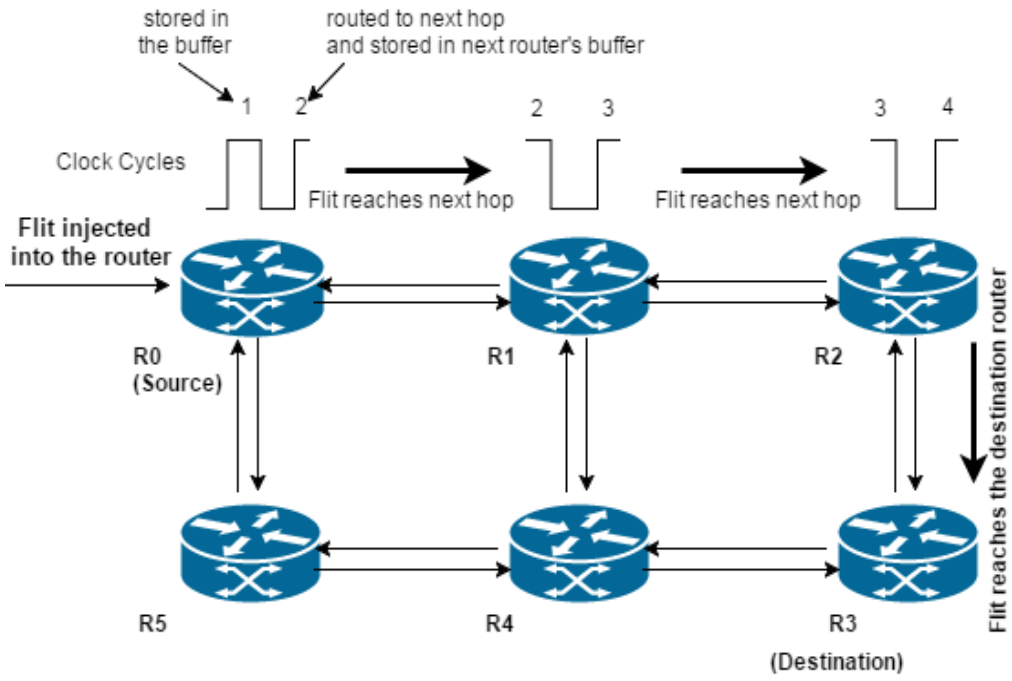


Figure 4.9: Clock cycles requirement for proposed single pipelined stage router in a 2x3 mesh network for zero load

On the other hand, for the proposed router as demonstrated from the figure 4.9, in a single clock cycle, the flit is stored in the buffer and simultaneously the arbitration and switch allocation happen which results in flit ejection from the router in next clock cycle. From the figure, with the proposed router architecture, the flit can reach its destination in the 4th clock cycle reducing the router delay by 60% compared to the generic router. The parallelism achieved by the design helps to reduce the router delay to minimum and correspondingly the latency as well.

The proposed design however is best suited for small networks with relatively smaller traffic. The latency is kept to minimum but with larger traffic due to lack of more sophisticated flow control system and routing algorithm, the latency can rise exponentially whereas for more advance router micro architecture the latency in a zero load may be relatively higher compared to the proposed router but they employ more sophisticated architecture to better control the data flow with more advance routing algorithms.

The graph in figure 4.10 shows how the router delay for proposed router compare with the more sophisticated router architecture with better flow control and routing algorithms. The graph is an assumption on the behaviour of the proposed router and more sophisticated router with increasing traffic. Based on the results obtained from the proposed router tests and making predictions on the average latency behaviour of more sophisticated routers with better flow control and routing algorithm like one proposed by Gaoming et al. in [6] , the graph in figure 4.10 shows the comparisons between the two.

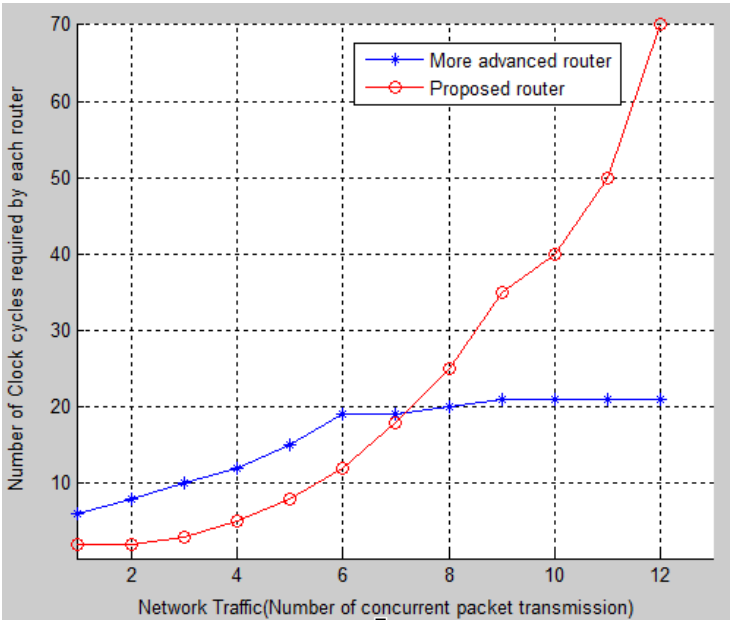


Figure 4.10: Assumption of router delay with increasing network traffic for proposed and more advanced router

The graph in the figure 4.10 shows that the proposed simple router has much less latency in comparison with more advanced router when the traffic is not overwhelming. The more advance router induces more latency even for low network traffic due to

complex flow control and arbitration which could take multiple clock cycles to execute. However, after a threshold network traffic (7 concurrent transmission in this case), the router delay for proposed router keeps on increasing whereas more advance router obtains a steady delay as it is more suited to deal with larger traffic. This demonstrates that the proposed router is best suited for low traffic small network. Further work on better flow control to handle more traffic could help to reduce latency in larger network traffic as well for the proposed design.

The proposed router architecture uses a simple lookup table based distributed deterministic routing where the routing path is not embedded in the packet header but computed at the router. The size of the destination Id bit in the packet remains relatively small even with the growing network size since it uses deterministic routing and not source routing. The next hop is calculated at each router so extra information for the routing need not to be embedded in the packet itself. This reduces the overall flit size. Larger flit size requires larger data bus and hence increases the resource usage and heat dissipation. Small flit size leads to reduced latency. For a mesh network with k routers, the destination Id size is $log2(k)$ [17]. The table 4.5 shows the number of bits required for destination Id for NxN mesh network with k number of routers.

Table 4.5: The number of bits required for destination Id with deterministic routing.

Mesh Network (NxN) using deterministic routing		
NxN (Mesh)	Number of routers(k)	Required number of bits for destination Id in a flit ($log2(k)$)
2x2	4	2
3x3	9	4
4x4	16	4
5x5	25	5
6x6	36	6

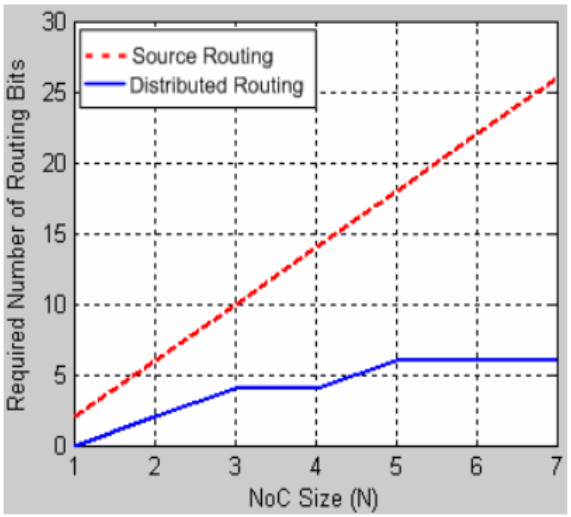


Figure 4.11: Routing bits requirement for NXN sized NoC for Source and Distributed routing [40]

Mubeen et al. [40] demonstrated the number of bits required for NXN mesh network for source and distributed routing or deterministic routing via a graph shown in the figure 4.11. The findings of Mubeen et al. [40] can be used for the design to evaluate and justify the choice for using distributed deterministic routing over source routing. The graph clearly shows that the number of bits required for distributed deterministic routing is much less than the source routing which results in the smaller packet size. Realising this advantage, the proposed design employs distributed deterministic routing to keep the packet size small and hence less resource usage, less heat dissipation and less latency.

4.5 Summary

The design is modelled in Verilog HDL using Altera's Quartus II 14.1. The design is broken down in to many smaller components which are separately modelled and then assembled together to form a NoC router. The NoC router is then instantiated four times and are connected in a 2x2 mesh fashion. After successful compilation, a minimal testbench is written in Verilog to test the working of the design. ModelSim is used for simulation and verification of the design.

The first test was to send a two flits packet from router 0 to router 1. The simulation waveform shows that the flits can traverse successfully to the destination router. Then

a single flit is sent from router 1 to router 3 and this flit as well can traverse successfully. This confirms the validity of the design.

After the minimalistic testbench, more tests are conducted with zero load (only one router transmitting) and with load (multiple routers transmitting). The number of clock cycles it takes the packet to reach the destination are calculated. In both cases, the packets are single flit. For zero load test, the maximum clock cycles required is 3 whereas for with load test it is found to be minimum of 2 and maximum of 4 clock cycles when three routers are all sending a single flit to the same destination router. This is a reference test and the clock cycle can be lot more than that if there are more flits already stored in the buffer or at the time of more packet congestion.

The design is evaluated mainly in terms of router delay and latency, and routing algorithm used. The router is designed as a single stage pipelined router. A generic NoC router is composed of 3-5 pipeline stages. Comparisons are made with the generic NoC router with 4 pipeline stages and the proposed router based on the router delay. Upon evaluation, it is found that proposed router reduces the router delay by up to 60% and hence the latency. The parallelism feature shown by the design enables the integration of more than one stage in a single stage in the router pipeline. However, the design is better suited for small network with fewer traffic. Figure 4.10 shows the limitations of the design in comparison with more sophisticated designs.

Figure 4.11 shows the benefits of having a distributed routing over source routing. The packet size remains relatively smaller for distributed which results in better resource utilisation and eventually lower latency.

Chapter 5 FPGA Resource Utilisation and Evaluation

This section talks about the implementation methodology of the design onto the De1-SoC board with cyclone V FPGA from Altera. Upon compilation, the synthesis results are analysed to make comparisons based on the resources usage and timing constraints of the design. Resources like logic elements, registers, RAM, pins and others are analysed to evaluate the design.

5.1 FPGA implementation methodology

Altera's Quartus II 14.1 tool is used to write the modules and compile to generate the compilation results. While creating the project in the beginning right FPGA needs to be chosen to generate resource usage report for the right FPGA. The FPGA used is Altera Cyclone V in a De1-SoC board with serial number 5CSEMA5F31C6. The FPGA board is connected to the PC using a USB JTAG (Joint Test Access Group) cable and is programmed using that same cable. After all the modules are successfully compiled with appropriate top level entity of the design, the compilation generates .sof (SRAM Object File) file which basically contains all the FPGA configurations that maps the design onto the FPGA board. After that Quartus programmer is run and .sof file is programmed into the FPGA which eventually configures the FPGA fabric with the desired design. FPGA system clock can be used to provide clock source to the design.

The original idea was to test the network on an FPGA by connecting the router in an FPGA to a NiosII processor which acts as a dummy core that sends and receives packets to and from the routers and use FPGA PLLs to provide clock source. Due to limited time, this part of the project is left for future work. However, the modules are successfully synthesizable and the synthesis results are analysed to make evaluation and compare based on the resource usage and timing requirements.

5.2 Evaluation of the resource utilisation

The table 5.1 lists the resources utilization for each of the components and the network as a whole for Cyclone V FPGA.

Table 5.1: Resource utilisation observed after the compilation for each component

Components	Logic Utilization(in ALMs) Total = 32070	Registers	Block Memory(4,065,280)	Pins(457)
Network	515(2%)	558	1152 (<1%)	146(32%)
CoreRouter	401(1%)	285	0	187(41%)
InputQueue	25(<1%)	30	0	30(7%)
OutPortFIFO	28(<1%)	36	0	24(5%)
InputArbiter	11(<1%)	0	0	57(12%)
OutputArbiter	11(<1%)	0	0	57(12%)
RouterAllcoator	44(<1%)	25	0	55(12%)
OutputEncoder	3(<1%)	0	0	9(2%)

From the table 5.1, it is clear that the total logic utilisation for the whole Network is only 2 % and each core router takes only 1% of the total Logic in terms of ALM (Adaptive Logic Modules) for a 2x2 mesh topology on De1-SoC. The Logic utilisation is very low which means that this design is not very resource hungry and can be achieved at low cost. The proposed router architecture by Aslam et al. [30] which is a Junction Based Router, is prototyped onto a Altera De2 board. Their results showed that the router architecture takes 7 % of the available logic elements which is 2259 logic elements whereas our proposed design only takes 1 % of the available logic resources which is only 401 logic elements. In addition, the number of block memory

bits used in the proposed router is only 1152 bits compared to [30] which is 5712 bits for the router. The proposed router only uses around 285 registers that are used as a distributed memory to store output port number from the flit whereas for [30] the number of registers used is 754 which is around 2 % of the total registers.

Based on the total resources usage from the table, each of the component takes less than 1% of the total resources. Due to low resource usage, the tested 2x2 mesh network can scaled up with reasonable amount of resources used in total.

After the compilation, the maximum clock frequency possible for the design is observed to be 147.3 MHz .

The compilation results for the overall network can be referred from the *Appendix E*.

The router design as already mentioned is still in progress and even though the resource utilisation is quite low, the design is not yet optimised for very high traffic and large networks. The flow control is relatively simpler and only virtual channel is employed at the moment. However, after adding extra virtual channels and more complex routing algorithms, the design is predicted to work better for large traffic and larger networks with relatively lower latency and resource usage.

As mentioned in the section 4.4 of chapter 4, the design reduces router delay by executing different stages of the pipeline in a parallel manner. Even though FPGA operates at much lower frequency in comparison to CPU or GPU, the advantage of using FPGA over other technology lies in the fact that FPGA is highly flexible and allows parallel processing to be configured in the hardware level [41]. Parallel processing helps to achieve higher speed which could be way more than the operating speed. Since the proposed design demonstrate parallel processing, FPGA provides those features to be implemented on the FPGA and reduce number of pipeline stages to achieve lower latency.

5.3 Summary

The design is compiled for the De1-SoC with Cyclone V FPGA in Altera's Quratus II 14.1 tool. The synthesis results from the compilation shows the resources utilisation of the design in the De1-SoC board. The resource utilisation becomes important part of the design to evaluate if the design is actually compatible for the chosen FPGA. The synthesis result shows that the proposed router only takes 1% of the total ALMs which is 401 ALMs in comparison with the junction based router proposed by Aslam et al. [30] implemented on De2 board which uses 7% of the total logic elements which is 2259 logic elements. The resource usage is relatively less. This also means that the design can be scaled up to larger networks while still maintaining reasonable amount of FPGA logic element utilisation. Along with the logic elements, the number of registers and block RAM bits are also relatively less in comparison with the router proposed by Aslam et al. [30]. Upon compilation, the maximum frequency upon which the design can operate for the De1-SoC board is observed to be 147.3 MHz.

The original idea was to test the design by implementing it on to the FPGA and connecting it to a NiosII processor which can work as a dummy core that sends and receives packets to and from the router at variable rates via software. Due to limited time this task has been put into the future works section.

Chapter 6 Conclusions and Future Works

Due to advancement in the CMOS technology, more and more cores can be integrated within a single chip creating more complex System on Chip (SoC) architectures and these multi core architectures are becoming a common standard in chip design. The chip interconnection fabric becomes crucial part when it comes to evaluation of efficiency and performance of these multi core chips. The traditional bus and crossbar interconnection are not efficient in terms of scalability and performance of the design since these interconnections are part of the chip design. To overcome these performance bottleneck, and decouple communication framework from the computation, there has been a paradigm shift towards incorporation of packet switching concept within a chip called Network-on-Chip (NoC) borrowed from computer networks.

In NoC, data are divided into packets and may be flits, and are routed over the chip just like in a conventional computer network. The concept of routers and switches are introduced inside the chip that act as an intermediate components to drive the data across the chip among different cores. The architecture of the router itself, the routing algorithms used and the topology in which the routers are arranged governs the performance of the NoC. This thesis has focused mainly on the latency of the router. However, the thesis has also covered overall architectural topology and design of the router with explanation of different components that comprises the router.

In this thesis, a low latency single stage pipelined router has been proposed that helps to achieve some sort of parallelism where there are multiple stages running concurrently in a parallel manner. This reduces the overall clock cycles taken by each router to a minimum of 2 clock cycles. When compared to generic router which normally has 3-5 router pipeline stages, the router delay was found to be reduced by up to 60% due to parallel execution of different pipeline stages in a single clock cycle.

To keep the design simple and reduce latency, a rather simple lookup table based distributed deterministic routing is used. Source routing can be a little complex and also all the path information has to be embedded inside the packet itself which increases the packet size resulting in larger resource usage. The routing information is stored in a hex file which maps the destination router to the output port of a particular

router. The design helps to achieve parallelism because as the flit enters the router, a certain part of the flit is sent for arbitration and allocation and the remaining flit is stored in the flit buffers concurrently. If the output port for a flit wins an arbitration then the flit is read out from the flit buffer to the next router or destination router in the second clock cycle.

The design is then compiled for FPGA prototyping on De1-SoC board packaged with Altera's Cyclone V FPGA. The compilation results showed that only 1% of the logic elements (401) was used up by the core router. In comparison to router proposed by Aslam et al. [30] implemented on De2 board, their router uses 7% of the total logic elements which is 2259 logic elements. The overall network also only takes 2% (512) of the logic elements. The total number of registers used are 285 by each router and only 1152 bits of block RAM memory are used. The resource usage is relatively low which means the design is scalable in terms of resources required.

The initial idea was to connect FPGA with a NiosII soft processor where the processor would act as a dummy core and send and receive packet to and from the router. This would have allowed us to send and receive packets via software. Due to limited time this part of the project is kept as a future work.

The proposed design has limitations when it comes to larger network size and high traffic. The router uses simple routing algorithm with simple flow control system. Based on the figure 4.10, the graph demonstrates that the proposed router works best for smaller network with less network traffic. After a threshold traffic point, the router delay keeps increasing exponentially and cannot keep up the performance benchmark as proposed. Having more virtual channels could be used to better optimise the design. This allows more buffer space and better utilisation of the channel as it allows channel usage in time multiplexed fashion. Also, more sophisticated arbitration logic helps to make sure that the priority assigned for each packet is fair to avoid some packets waiting for a long time where newly arrived packets wins the arbitration.

The routing algorithm used for the design is lookup table based deterministic routing. The algorithm is not dynamic to sense the traffic load in the network and adapt to high traffic dynamically. Having a dynamic routing algorithm where upon less traffic the design employs simpler yet faster algorithm and when the traffic volume increases then switches to more sophisticated algorithm to better control the routing process.

References

[1] R. R. Schaller, “Moore's law: past, present and future,” *IEEE Spectrum,* vol. 34, no. 6, pp. 52-59, 1997.

[2] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks,” Computer Systems Laboratory, Stanford University, Stanford, CA.

[3] Arteris, “From “Bus” and “Crossbar” to "Network-On-Chip",” Arteris Inc, San Jose, 2009.

[4] C. Nicopoulus, V. Narayanan and C. R. Das, Network-on-Chip Architectures A Holistic Design Exploration, New York: Springer, 2009.

[5] M. Daneshtalab, “Exploring Adaptive Implementation,” 2011.

[6] D. Gaoming, L. Dayi, S. Yukun and Z. Duoli, “A Dynamic and Mixed Routing Algorithm for 2D Mesh NoC,” Hefei University of Technology, Hefei,China.

[7] P. Kotriki and P. Patil, “FPGA Based : Design and Implementation of NoC Torus Topology,” *International Journal of Research in Engineering and Techonology,* vol. 03, no. 03, pp. 399-402, 2014.

[8] J. Chen , C. Li and P. Gillard, “Network-on-Chip (NoC) Topologies and Performance: A Review”.

[9] Terasic, “http://www.terasic.com.tw/,” Terasic, [Online]. Available: http://www.terasic.com.tw/. [Accessed 05 07 2015].

[10] R. Balasubramonian and T. M. Pinkston, “Buses and Crossbars”.

[11] S. Kumar , A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg1, J. Öberg, K. Tiensyrjä and A. Hemani, “A Network on Chip Architecture and Design Methodology,” *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on VLSI,* pp. 105-112, 2002.

[12] J. L. Silva, N. Nedjah and L. d. M. Mourelle, “Routing for applications in NoC using ACO-based algorithms,” *Applied Soft Computing,* vol. 13, no. 5, pp. 2224-2231, 2013.

[13] L. P. Carloni, P. Pande and Y. Xie, “Networks-on-Chip in Emerging Interconnect Paradigms: Advantages and Challenges”.

[14] Altera, “Applying the Benefits of Network on a Chip Architecture to FPGA System Design,” Altera Corporation, 2011.

[15] L. Benini and G. D. Micheli, “Networks on Chips: A New SoC Paradigm,” in *Computer* , 2002, pp. 70-78.

[16] P. Tvrdik, “http://pages.cs.wisc.edu,” [Online]. Available: http://pages.cs.wisc.edu/~tvrdik/7/html/Section7.html. [Accessed 15 08 2015].

[17] M. Palesi and M. Daneshtalab, Routing Algorithms in Networks-on-Chip, New York: Springer, 2014.

[18] Y. Xu, J. Zhou and S. Liu, “Research and Analysis of Routing Algorithms for NoC,” *Computer Research and Development (ICCRD),* vol. 2, pp. 98-102, 2011.

[19] R. Holsmark, M. Palesi and S. Kumar , “Deadlock free routing algorithms for irregular mesh topology NoC systems with rectangular regions,” *Journal of Systems Architecture,* vol. 54, no. 3-4, pp. 427-440, 2008.

[20] I. Kuon, R. Tessier and J. Rose, “FPGA architecture:Survey and Challenges,” *Foundations and Trends in Electronic Design Automation,* vol. 2, no. 2, pp. 135-253, 2007.

[21] B. Zeidman, “http://chipdesignmag.com,” [Online]. Available: http://chipdesignmag.com/print.php?articleId=434?issueId=16. [Accessed 29 07 2015].

[22] I. IKuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 26, no. 2, pp. 203-215, 2007.

[23] Altera, “https://www.altera.com,” Altera, [Online]. Available: https://www.altera.com/products/fpga/cyclone-series/cyclone-v/features/cyv-architecture.html. [Accessed 2 08 2015].

[24] Altera, “http://quartushelp.altera.com/,” Altera, [Online]. Available: http://quartushelp.altera.com/13.1/mergedProjects/reference/glossary/def_sof.htm. [Accessed 26 07 2015].

[25] Altera, “Quartus II Handbook Volume 1: Design and Synthesis,” Altera, San Jose, Califronia, 2014.

[26] Altera, “Introduction to the Quartus II Software,” Altera, San Jose, California, 2010.

[27] P. . H. J. Kelly, “https://www.doc.ic.ac.uk/,” [Online]. Available: https://www.doc.ic.ac.uk/~phjk/AdvancedCompArchitecture/2001-02/Lectures.old/Ch06/node34.html. [Accessed 21 07 2015].

[28] A. Roca, J. Flich, F. Silla and Duato Jose, “A Latency-Efficient Router Architecture for CMP Systems,” *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools,* pp. 165-172, 2010.

[29] D. Frazzetta , G. Dimartino, M. Palesi, S. Kumar and V. Catania, “Efficient Application Specific Routing Algorithms for NoC Systems utilizing Partially Faulty Links,” *11th EUROMICRO CONFERENCE on DIGITAL SYSTEM DESIGN Architectures, Methods and Tools,* pp. 18-25, 2008.

[30] M. A. Aslam, S. Kumar and R. Holsmark, “An Efficient Router Architecture and its FPGA Prototyping to support Junction Based Routing in NoC Platforms,” *16th Euromicro Conference on Digital System Design,* pp. 297-300, 2013.

[31] P. M. Yaghini, A. Eghbal and N. Bagherzadeh, “On the design of hybrid routing mechanism for mesh-based network-on-chip,” *INTEGRATION, the VLSI journal,* vol. 50, pp. 183-192, 2015.

[32] B. Sethuraman, P. Bhattacharya, J. Khan and R. Vemuri, “LiPar: A Light Weight Parallel Router for FPGA based Networks on Chip,” *GLS VLSI,* 2005.

[33] R. Gindin, I. Cidon and I. Keidar, “NoC-Based FPGA: Architecture and Routing”.

[34] A. V. V. Rose, D. R. Seshasayanan and G. Oviya, “FPGA Implementation of Low Latency Routing Algorithm for 3D Network on Chip,” *IEEE- International Conference on Recent Trends in Information Technology,* pp. 385-388, 2011.

[35] Y. Yamak, “http://www.mcu-turkey.com,” MCU CPU Turkey, 2014. [Online]. Available: http://www.mcu-turkey.com/fpga-ile-uart-tx-modulu-tasarimi/. [Accessed 15 08 2015].

[36] W. J. Dally and B. Towles, Principles and Practices of Interconneciton Networks, San Francisco, CA: Morgan Kaufmann, 2004.

[37] G. Dimitrakopoulos, A. Psarras and I. Seitanidis, Microarchitecture of Network-on-Chip Routers : A Designer's Perspective, New York: Springer, 2014.

[38] D. U. Becker and W. J. Dally, “Allocator Implementation for Netowork-on-Chip Routers,” in *Conference on High Performance Computing, Netowrking,Storage and Analysis*, 2009.

[39] D. Park, “DESIGN OF HIGH-PERFORMANCE, ENERGY-EFFICIENT, AND RELIABLE NETWORK-ON-CHIP (NoC) ARCHITECTURES,” Pennsylvania State University, , 2008.

[40] S. Mubeen and S. Kumar, “On Source Routing for Mesh Topology Network on Chip,” Jönköping University, Sweden.

[41] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model and D. DiSabello, “Achieving High Performance with FPGA-Based Computing,” IEEE Computer Society, Boston, 2007.

[42] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä and A. Hemani, “A Network on Chip Architecture and Design

Methodology," *Proceedings. IEEE Computer Society Annual Symposium on,* pp. 105 - 112, 2002.

Appendix

Appendix A De1-SoC Board

Figure 7.1 shows the bird's eye view of the De1-SoC board. The label in the picture is categorised into three parts as FPGA, HPS and System. The green colour shows the FPGA part of the SoC, Yellow colour shows HPS part and blue shows the system components.

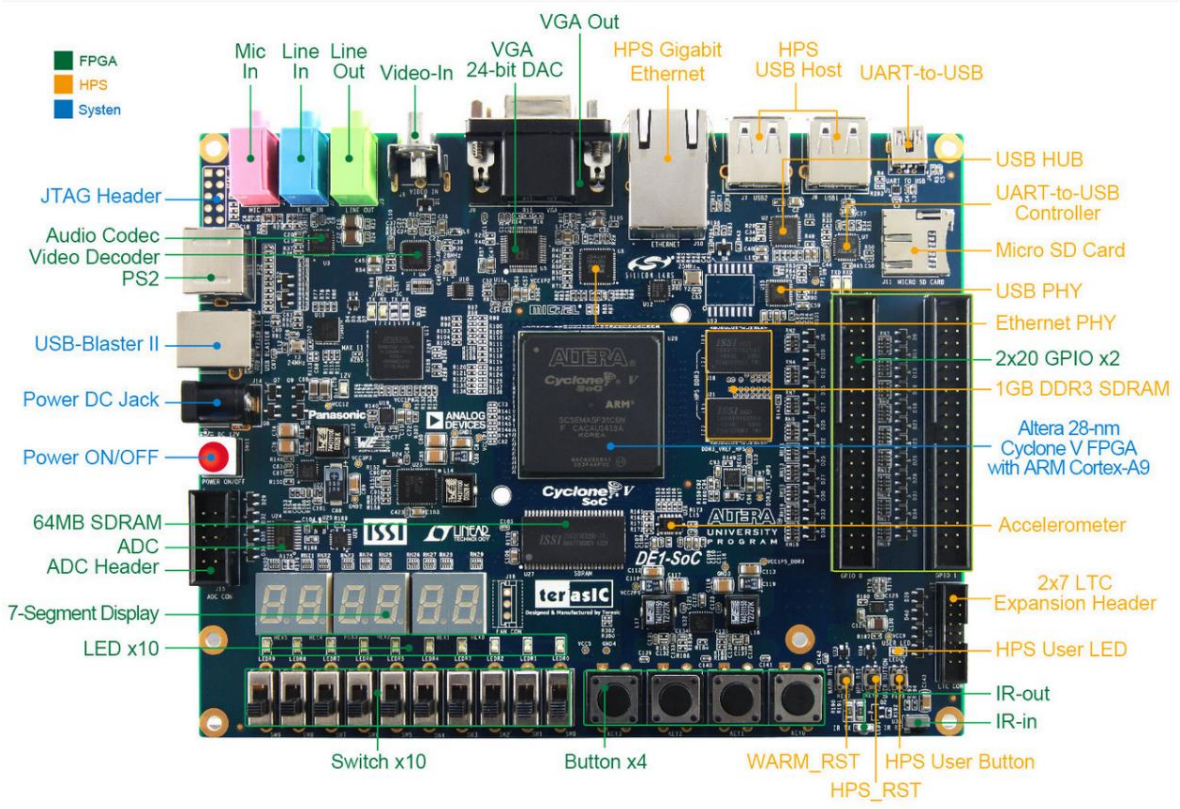


Figure 7.1: De1-SoC board [9]

Appendix B RTL netlist schematic of flit input

Figure 7.2 is a clearer and larger picture of the RTL netlist generated by Quartus showing the flit injection into the network. As discussed in sub section 3.2.2 of chapter 3, the destination bit [10:9] from the flit is extracted and sent to the RoutingTable_2 storage where the destination Id is mapped with the right output port and that output port is again tagged with the flit and sent as a 16 bits flit into the router. The routing path is calculated in this manner right before the flit enters the router.

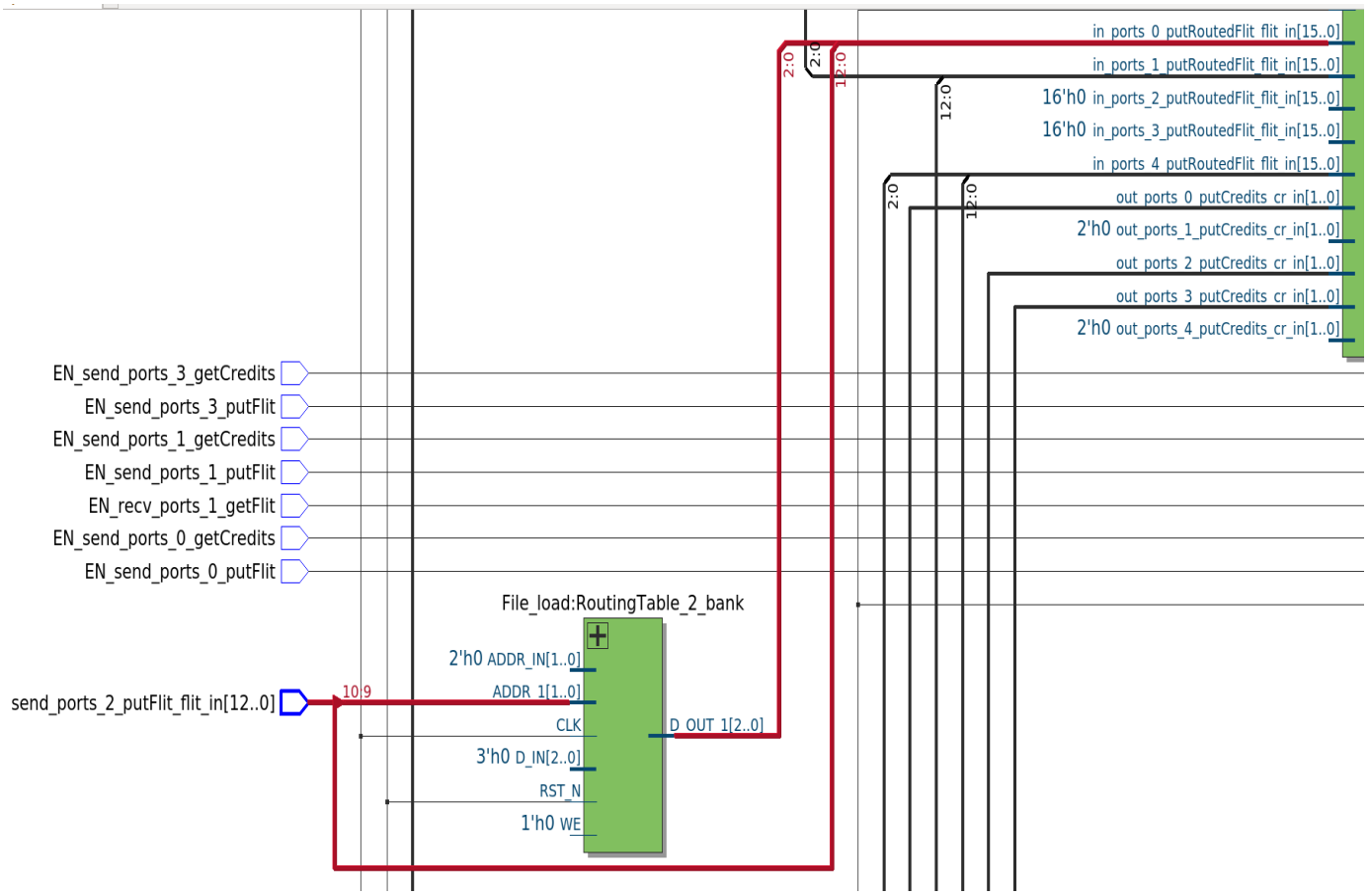


Figure 7.2: Flit injection into the network. The destination Id is sent to the OutPortFIFO

Appendix C Wormhole Switching

Wormhole switching is a flow control mechanism where all the flits of a packet need not be stored in a flit buffer of an intermediate router before sending it to another router. The flit buffer depth need not to be able to accommodate all the flits of a particular packet. In this method the, the flits continue to move around the network without waiting for all the other flits of that packet at every router. Hence, a packet of five flits could have head flit already reaching the destination, body flit somewhere in the network and the tail flit still at the source.

Appendix D Snippet of Testbench Code

Snippet of flit injection testbench code.

This verilog code is a snippet of the testbench written to inject flits from multiple router all destined for a single router.

```
// Generate Clock
  initial Clk = 0;
  always #(HalfClkPeriod) Clk = ~Clk;

  // Run simulation
  initial begin
    cycle = 0;
    for(i = 0; i < `NUM_USER_SEND_PORTS; i = i + 1) begin flit_in[i]
= 0; send_flit[i] = 0; end


    $display("---- Performing Reset ----");
    Rst_n = 0; // perform reset (active low)
    #(5*ClkPeriod+HalfClkPeriod);
    Rst_n = 1;


    //sending a single flit packet from router 1 to router 2
    send_flit[1] = 1'b1;
    dest = 2;
    vc = 0;
    data = 'h10;
    flit_in[1] = {1'b1 /*valid*/, 1'b1 /*tail*/, dest, vc, data};
    $display("@%3d: Injecting flit %x into send port %0d", cycle,
flit_in[1], 1);

    // send a 1-flit packet from router 3 to router 2
    send_flit[3] = 1'b1;
```

```
    dest = 2;
    vc = 0;
    data = 'hff;
    flit_in[3] = {1'b1 /*valid*/, 1'b1 /*tail*/, dest, vc, data};
    $display("@%3d: Injecting flit %x into send port %0d", cycle,
flit_in[3], 3);

    //sending a 2 flits packet from router 0 to router 2
    send_flit[0] = 1'b1;
    dest = 2;
    vc = 0;
    data = 'h11;
    flit_in[0] = {1'b1 /*valid*/, 1'b0 /*tail*/, dest, vc, data};
    $display("@%3d: Injecting flit %x into send port %0d", cycle,
flit_in[0], 0);

    #(ClkPeriod);

    // sending the 2nd flit of packet
    send_flit[0] = 1'b1;
    data = 'h12;
    flit_in[0] = {1'b1 /*valid*/, 1'b1 /*tail*/, dest, vc, data};
    $display("@%3d: Injecting flit %x into send port %0d", cycle,
flit_in[0], 0);

    #(ClkPeriod);

    // stop sending flits
    send_flit[0] = 1'b0;
    flit_in[0] = 1'b0; // valid bit
    // stop sending flits
    send_flit[3] = 1'b0;
    flit_in[3] = 1'b0; // valid bit
    // stop sending flits
    send_flit[1] = 1'b0;
    flit_in[1] = 1'b0; // valid bit

  end


  // Monitor arriving flits
  always @ (posedge Clk) begin
    cycle <= cycle + 1;
    //$display("cycle going on");
    for(i = 0; i < `NUM_USER_RECV_PORTS; i = i + 1) begin
      if(flit_out[i][flit_port_width-1]) begin // valid flit

        $display("@%3d: Ejecting flit %x at receive port %0d",
cycle, flit_out[i], i);
      end

    end

    // terminate simulation
```

```
    if (cycle > test_cycles) begin
      $finish();
    end
  end
```

Appendix E Compilation results of the overall network.

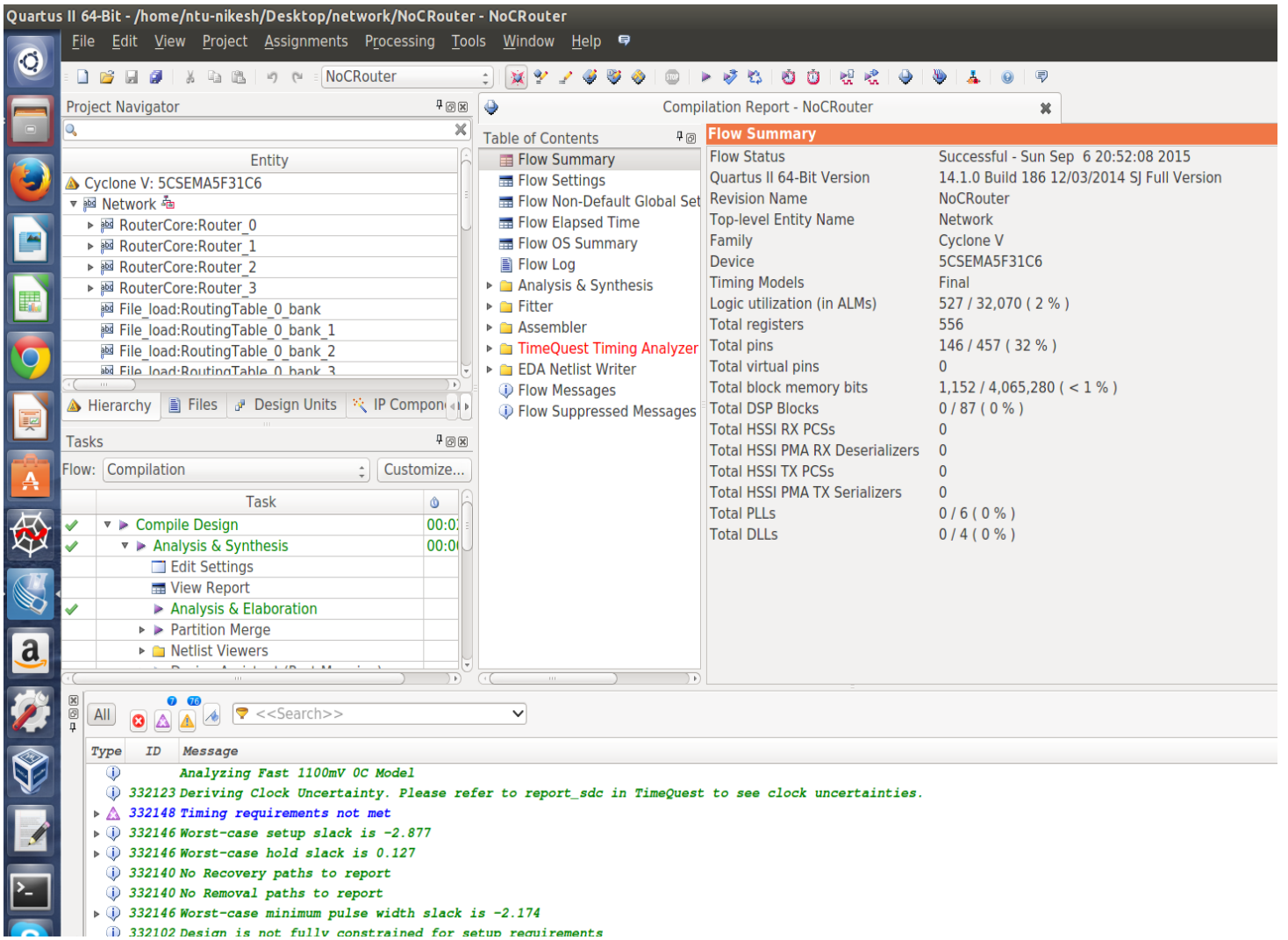


Figure 7.3: Compilation Report of network module