Router Architecture for Junction Based Source Routing: Design and FPGA Prototyping

Muhammad Awais Aslam

MASTER THESIS 2011
Electrical Engineering: Specialisation in Embedded Systems
Router Architecture for Junction Based Source Routing: Design and FPGA Prototyping

Muhammad Awais Aslam

This exam work has been carried out at the School of Engineering in Jönköping in the subject area Electronics. The work is a part of the two-year Master of Science programme. The authors take full responsibility for opinions, conclusions and findings presented.

Supervisor: Shashi Kumar and Rickard Holsmark

 Examiner: Shashi Kumar

Scope: 30 credits (D-level)

Date:
Abstract

The increase in the number of cores that can be integrated on a single chip has forced the designer to use computer network concepts for design of System on Chip (SoC). This idea led to development of Network on Chip (NoC) to deal with more cores on a single chip. NoC has three main parts, namely routers, link and network interface through which cores are connected to NoC. Router is one of the most important parts because cores communicate with other cores through routers. One of the important tasks for a NoC designer is to design router with low latency.

Router design depends on the routing protocol and routing algorithm used. Two kinds of routing algorithms are source routing and distributed routing. In source routing, complete route information is available in Head flit while in distributed routing, routing decisions are taken inside every router on the path. Source routing has speed advantage over distributed routing because the packet itself contains the routing information. But source routing leads to overhead to store complete path information in the header of each packet. To overcome this flaw, junction based source routing has been introduced. If destination is far away from the source then first packet will go to a junction and get the new path information from the junction to the destination. Thus we need to store the path information only for a few hops in the packet header. This idea has been taken from the daily experience of train journey. In this thesis we have developed design of a router for junction based routing. Main component of simple router includes buffering, header modification and making route decision. Router includes a table called Path Table which stores information about paths from junction to various destinations. JB router also includes, picking up the new path information from Path Table and modify the header by adding new path information.

We have developed VHDL designs of two versions of the routers for Junction Based Routing. The delay performance of routers have been analysed through simulation. A simple prototype of the router has also been implemented in Altera FPGA to find out the resource requirements of the new router designs.
Keywords

System on Chip (SoC)
Network on Chip (NoC)
Source routing
Distributing routing
Junction Based routing
Router architectures
Path Table
Altera FPGA
NiosII Core
First of all I am thankful to Almighty Allah, the Most Beneficent, the Most Merciful whose blessings provided me the strength to successfully complete this Master Thesis.

I would like to thank my supervisor Professor Shashi Kumar, who introduced me to the field of NoC and provided me the opportunity to work under his kind supervision. I am thankful to him for providing me with invaluable guidance throughout the thesis work. His research experience and regular meetings with motivating solutions provided me a great opportunity to learn many things throughout the thesis. I am grateful and feel proud to work under the supervision of one of the founders of Network on Chip (NoC) paradigm.

I thank Rickard Holsmark for his supervision and support. I appreciate his time and cooperation that helped me in enhancing my thesis design.

I would like to thank Alf Johansson, Master program coordinator, for being helpful and caring throughout the Master program.

Last but not least, I am very thankful to my parents for their motivation during my difficult hours and support throughout my life.
# Table of Contents

1 **Introduction** ........................................................................................................... 1  
1.1 **SYSTEM ON CHIP** ................................................................................................ 1  
1.2 **NETWORK ON CHIP** .............................................................................................. 1  
1.3 **IMPORTANT ISSUES IN NOC BASED DESIGN** ....................................................... 2  
1.4 **GOALS, SCOPE AND TASKS OF THE THESIS** .................................................... 3  
1.5 **THESIS LAYOUT** .................................................................................................... 4  

2 **Theoretical Background** .......................................................................................... 5  
2.1 **NOC CONCEPT** ....................................................................................................... 5  
  2.1.1 **Core to Core Connections** .................................................................................. 5  
  2.1.2 **Bus Based Connections** ..................................................................................... 6  
  2.1.3 **Router Based Connections** ................................................................................ 7  
2.2 **NOC TERMINOLOGIES AND SWITCHING TECHNIQUES** .................................... 8  
2.3 **COMPONENTS OF NOC ARCHITECTURE** ............................................................. 9  
2.4 **ROUTING ALGORITHM** ......................................................................................... 10  
  2.4.1 **Distributed vs. Source routing** ......................................................................... 10  
  2.4.2 **Junction Based Routing (JBR)** ........................................................................... 11  
  2.4.3 **Comparison of Header Format** ......................................................................... 13  
2.5 **ROUTER ARCHITECTURE** ....................................................................................... 14  
  2.5.1 **Functionality of router** .................................................................................... 14  
  2.5.2 **Input buffer** .................................................................................................... 14  
  2.5.3 **Cross Bar Switch** ............................................................................................ 15  
  2.5.4 **Arbitration and Control** ................................................................................... 15  
2.6 **HANDSHAKING PROTOCOL** .................................................................................. 16  
2.7 **GAISLER’S TWO PROCESS METHOD** .................................................................... 17  
2.8 **PROTOTYPE OF DESIGN** ....................................................................................... 18  
  2.8.1 **FPGA Technology** ........................................................................................... 18  
  2.8.2 **FPGA Prototyping Design Flow** ...................................................................... 19  

3 **Router Design Options** ............................................................................................ 21  
3.1 **HARDWARE ROUTER** ............................................................................................ 21  
  3.1.1 **Architecture** .................................................................................................... 21  
3.2 **NETWORK PROCESSOR AND SOFTWARE ROUTER** ........................................... 23  
3.3 **TABLE BASED ROUTER** ....................................................................................... 25  
3.4 **COMPARISON** ...................................................................................................... 26  

4 **Router Design for Junction Based Routing (JBR)** .................................................. 27  
4.1 **PATH TABLE STORAGE OPTIONS** ......................................................................... 28  
  4.1.1 **Path Table in router for each Input Port and one Path Table in RNI** ............... 28  
  4.1.2 **Common Path Table for all Input Ports in router and one Path Table in RNI** .... 29  
  4.1.3 **Common Path Table in RNI** ............................................................................ 30  
  4.1.4 **Only One Path Table in router** ....................................................................... 31  
4.2 **PACKET FORMAT** ................................................................................................ 32  
  4.2.1 **Flit format** ..................................................................................................... 32  
4.3 **DESIGN OVERVIEW** ............................................................................................. 35  
4.4 **ROUTER DESIGN DETAILS** ................................................................................ 37  
  4.4.1 **Input Handler (IH)** ......................................................................................... 37  
  4.4.2 **Path Table Arbiter Component (PTAC)** ............................................................ 43  
  4.4.3 **Arbiter & Control** .......................................................................................... 46
Table of Contents

4.4.4 Cross Bar .................................................................................................................. 49

5 VHDL Modeling, Simulation and Results ................................................................. 51

5.1 TEST RESULTS ........................................................................................................... 52
  5.1.1 Testing with Zero load .............................................................................................. 52
  5.1.2 Testing with load and no output port conflict .......................................................... 53
  5.1.3 Testing with output port conflict ............................................................................. 55
5.2 IMPROVED DESIGN ................................................................................................. 56
5.3 RESULTS ....................................................................................................................... 57
  5.3.1 Testing with Zero load .............................................................................................. 57
  5.3.2 Testing with no output port conflict ......................................................................... 58
  5.3.3 Testing with output port conflict ............................................................................. 60

6 FPGA Prototyping ......................................................................................................... 64

6.1 PROTOTYPE RESULT .................................................................................................. 66

7 Conclusions .................................................................................................................... 70

7.1 SUMMARY OF CONTRIBUTION ................................................................................. 70
  7.1.1 Path Table Options .................................................................................................. 70
  7.1.2 Router Design, Modeling and Prototyping ............................................................... 70
7.2 LIMITATION AND FUTURE WORK ........................................................................... 71
  7.2.1 Path Table Compression ........................................................................................... 71
  7.2.2 Virtual Channel in Router ....................................................................................... 71
  7.2.3 Enhanced Pipelining Technique .............................................................................. 71
  7.2.4 Multiple FPGA Board for Prototyping ................................................................. 71

8 References ..................................................................................................................... 72

9 Appendix I ....................................................................................................................... 74
List of Figures

FIGURE 1.1: NETWORK ON CHIP .................................................................................. 2
FIGURE 2.1: CORE TO CORE BASED CONNECTION .................................................. 5
FIGURE 2.2: BUS BASED COMMUNICATION INTERFACE ........................................... 6
FIGURE 2.3: A 3X3 ROUTER BASED NETWORK ....................................................... 7
FIGURE 2.4: NOC COMPONENTS .................................................................................. 10
FIGURE 2.5: A 7X7 JUNCTION BASED NETWORK ON CHIP BASED SYSTEM ............. 12
FIGURE 2.6: FORMAT OF HEAD FLIT IN SOURCE ROUTING ................................. 13
FIGURE 2.7: FORMAT OF HEAD FLIT IN JUNCTION BASED ROUTING ................... 13
FIGURE 2.8: ROUTER ARCHITECTURE ...................................................................... 15
FIGURE 2.9: HANDSHAKING PROTOCOL BETWEEN SENDER AND RECEIVER ........... 16
FIGURE 2.10: GENERIC TWO PROCESS CIRCUIT [14] .............................................. 17
FIGURE 2.11: CYCLONE II BLOCK DIAGRAM [12] .................................................... 18
FIGURE 3.1: A TYPICAL NOC ROUTER ................................................................. 22
FIGURE 3.3: LATENCY/FLEXIBILITY VS. PERFORMANCE GRAPH ......................... 25
FIGURE 4.1: SEPARATE PATH TABLE IN ROUTER FOR EACH INPUT PORT AND ONE PATH TABLE IN RNI ................................................. 29
FIGURE 4.2: COMMON PATH TABLE FOR ALL INPUT PORTS IN ROUTER AND ONE PATH TABLE IN RNI ............................................................... 30
FIGURE 4.3: COMMON PATH TABLE IN RNI ........................................................... 31
FIGURE 4.4: ONLY ONE PATH TABLE IN ROUTER .................................................. 31
FIGURE 4.5: FORMAT OF HEAD FLIT IN JUNCTION BASED ROUTING ................. 34
FIGURE 4.6: FORMAT OF BODY FLIT IN JUNCTION BASED ROUTING .................. 34
FIGURE 4.7: FORMAT OF END FLIT IN JUNCTION BASED ROUTING ..................... 34
FIGURE 4.8: FORMAT OF FULL FLIT IN JUNCTION BASED ROUTING ................... 35
FIGURE 4.9: BLOCK DIAGRAM OF JUNCTION BASED ROUTER ................................. 35
FIGURE 4.10: ARCHITECTURE OF INPUT PORT ...................................................... 38
FIGURE 4.11: CIRCULAR MEMORY BUFFER ........................................................ 39
FIGURE 4.12: REGISTER CHAIN FOR IMPLEMENTING BUFFER ............................... 39
FIGURE 4.13: HEADER MODIFIER ........................................................................... 40
FIGURE 4.14: JUNCTION & RESOURCE CONTROL .................................................. 42
FIGURE 4.15: PATH TABLE ARBITER ......................................................................... 44
FIGURE 4.16: HIERARCHICAL FSM ......................................................................... 45
FIGURE 4.17: TABLE COMPONENT ......................................................................... 45
FIGURE 4.18: ARBITER & CONTROL ....................................................................... 46
Figure 4.19: Direction decoding ................................................................. 48
Figure 4.20: Cross bar .................................................................................. 50
Figure 4.21: Architecture of Junction Based Router .................................. 50
Figure 5.1: Router delay from input port to output port ............................... 51
Figure 5.2: Iterative circuit implementation in arbiter & control ............... 57
Figure 5.3: Simulation results when JB or RB is ‘0’ ................................... 61
Figure 5.4: Simulation results when JB or RB is ‘1’ ................................... 62
Figure 6.1: Hardware architecture of the system ....................................... 64
Figure 6.2: Quartus II architecture ............................................................... 65
Figure 6.3: Flow chart diagram of the design and test process .................. 67
Figure 9.1: Quartus II architecture for Router prototype ............................ 78
List of Tables

TABLE 2.1: CYCLONE II FAMILY OVERVIEW [19]..........................19
TABLE 3.1: LATENCY OF DIFFERENT ROUTERS..........................24
TABLE 3.2: COMPARISON BETWEEN DIFFERENT ROUTER DESIGN OPTIONS ..........26
TABLE 4.1: COMPARISON BETWEEN DIFFERENT ROUTER ORGANIZATIONS ..........32
TABLE 4.2: FLIT TYPES AND THEIR ENCODING................................33
TABLE 4.3: DIRECTION CODES..........................................................49
TABLE 5.1: SIMULATION RESULTS (WITHOUT LOAD)..........................53
TABLE 5.2: SIMULATION RESULTS (WITHOUT OUTPUT PORT CONFLICT).............54
TABLE 5.3: SIMULATION RESULTS (WITH LOAD AND OUTPUT PORT CONFLICT) ........55
TABLE 5.4: IMPROVED RESULTS (WITHOUT LOAD)................................58
TABLE 5.5: IMPROVED RESULTS (WITH LOAD AND NO OUTPUT PORT CONFLICT)........59
TABLE 5.6: IMPROVED RESULTS (WITH LOAD AND OUTPUT PORT CONFLICT) ........60
TABLE 6.1: FPGA IMPLEMENTATION RESULTS OF JBR ROUTER .................68
TABLE 6.2: A COMPARISON BETWEEN DIFFERENT ALTERA FAMILIES ..............69
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNI</td>
<td>Resource Network Interface</td>
</tr>
<tr>
<td>IHC</td>
<td>Input Handler Component</td>
</tr>
<tr>
<td>PTA</td>
<td>Path Table Arbiter</td>
</tr>
<tr>
<td>PT</td>
<td>Path Table</td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Access Group</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>RTR</td>
<td>Ready to Receive</td>
</tr>
<tr>
<td>NoC</td>
<td>Network on Chip</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>LE</td>
<td>Logic Elements</td>
</tr>
<tr>
<td>PIO</td>
<td>Parallel Input/Output</td>
</tr>
<tr>
<td>JB</td>
<td>Junction Bit</td>
</tr>
<tr>
<td>RB</td>
<td>Resource Bit</td>
</tr>
<tr>
<td>J&amp;R</td>
<td>Junction &amp; Resource</td>
</tr>
<tr>
<td>A&amp;C</td>
<td>Arbiter &amp; Control</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very High Speed Integrated Circuit Hardware Description Language</td>
</tr>
<tr>
<td>SOPC</td>
<td>System on Programmable Chip</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>LUT</td>
<td>Look Up Table</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
</tr>
<tr>
<td>LB</td>
<td>Logic Block</td>
</tr>
</tbody>
</table>
Introduction

New developments are necessary in every field, especially in technology, to improve quality of life. As we know this is the era of research and there is research in every field of life. In electronic, more emphasis is on how to reduce the chip area, its cost and how to develop a product in short interval of time. Integrating a complete system on a single chip is known as “System on Chip”. A lot of work has been done in this field. A regular increase in the number of transistors on a single chip has opened new fields and makes it possible to develop more complex systems on a single chip. Due to the increased complexity, there was a need of introducing a platform which makes it easy to integrate new component on a single chip with reduced complexity. In the last few years, a platform has been introduced name as Network on Chip (NOC). NOC has made it easy to develop systems with large number of cores on single chip in an easy way.

1.1 System on Chip

According to the Moore’s law the number of transistors doubles almost every 18 months [9]. All this is possible due to advancements in CMOS technology and it is easy to achieve billion of transistor on single chip. Due to the increase of numbers of transistors on a single chip, it is possible to develop more complex systems on a single silicon chip. The whole system is integrated in the using components which are called IP cores. These IP cores are directly connected to communicate with each other. These IP cores can be reused in different systems. These cores can be processors, I/O controllers, memory blocks, DSPs, video controller and many others. Increase in number of cores will increase interconnects complexity on the chip.

Earlier there were direct connections or busses used to connect cores. A bus is a shared resource which limits the number of cores connected to bus and it is difficult for multiple cores to communicate at the same time. More resources connected to bus also slow down the communication. This is one of the main problems in current SoCs. Problem with direct connections is that they have low scalability. To overcome these problems there was need of some new idea, which had been proposed a decade before. Next section will explain that idea.

1.2 Network on Chip

Interconnects and shared bus problems made researchers to think about a platform which can give an easy way to integrate new cores as well as old cores by following a protocol. On-chip packet-switched networks have been proposed as a solution for the problem of global interconnects in deep submicron VLSI Systems-on-Chip (SoC) [4]. A decade before an idea was introduced to use the networking concepts in SoC. Instead of using direct connection among cores, switches are introduced inside the chips for communication between the cores. One of the advantages of using switches is to make the SoC more scalable and flexible. Resources can share data with each other regardless they are connected
directly or not. Reuse of components (cores) has become easier than ever. These features have increased the importance of NoC in the field of electronics. A typical Network on Chip platform with mesh topology is shown in Figure 1.1.

A Core is connected to the router through RNI. RNI is the interface between router and the core. Core sends packet to the router through RNI, and from router it is send to the other routers or core depending on the destination. In literature, different NoC architectures have been proposed by the researchers [6] [7] [8].

**1.3 Important Issues in NoC based Design**

NoC design deals with many important issues. First issue in NoC is topology. Topology has importance for system performance and for complexity of router design. A good topology can make the router design less complex. There are different kinds of topologies in network like mesh, torus star, ring, spidergon and
Introduction

tree etc but it is found that most suitable topology for NoC is mesh topology. Routing algorithm is second issue in NoC because router design also depends on routing algorithm. There are two kind of routing algorithms called source routing and distributed routing. In source routing, path information is pre calculated and stored into packet header before sending the packet into network. The path information is stored in the form of Path Tables which can be placed either in router or in resource itself. In distributed routing, header packet only contains the destination address. Path decisions are taken inside the router by implementing routing function or by reading routing tables. Source routing has an advantage over distributed routing because path is pre-determined which make the router design simpler. But there is also a drawback in source routing. Due to packet overhead it needs more buffer size to store packet which results in more cost. To overcome this flaw junction based source routing has been introduced. The basic idea is to limit the required path information to be stored in every packet to a small number of bits. The large distance is covered by going through intermediate temporary destinations (called Junctions). Since router design depends on routing algorithm therefore we have chosen the junction based source routing for router design.

1.4 Goals, Scope and Tasks of the Thesis

Router is the most important component for design of communication backbone of a NoC system (like any other network). The design of a NoC router depends on various aspects of NoC architecture and the performance requirement. Each resource is connected to router through RNI.

In research work, goals are of great importance. Appropriate goals become cause of good research. First goal in this thesis is to design router architecture for junction based source routing. Junction based source routing needs two kind of router design: one for normal router and other for junction router. In the beginning of the thesis, the idea was to design two different routers but while analysing router design options it was decided that we will design a single router which works as normal router as well as junction router. Results are the most significant part of the research. Second goal of the thesis is to model the design in VHDL and simulate the design to evaluate its performance in various situations. The third goal was to prototype the design on Altera FPGA DE2 board and to evaluate its resource requirements.

In junction based routing, there is a need to store Path Tables in a router for new path information. This is a new issue because storing a Path Table in router makes it costly. One of the goals of the thesis was to develop a way to store a Path Table in router to make it efficient but less costly. How many copies of Path Table need to store in router to have efficient working was the question addressed? There is always trade-off between efficiency and cost to get good results. We will discuss about all possible design options and their comparison later in this thesis.

To achieve these goals, firstly specifications are made for router architecture by using systematic and modular approach. Router is divided into blocks and sub
blocks. Each block is designed using hardware description language VHDL. Each block is also simulated by using test benches in VHDL. To achieve the high performance an improved design is proposed and simulated after this. Simulation results of improved design are compared with earlier design and performance is evaluated. Since junction based router needs Path Table inside the router which uses memory to store path information. Due to increase in design complexity and Path Table, resource requirements also increase for junction based routing. Secondly, after simulation results, the design is prototyped on Altera FPGA DE2 board for cost and speed analysis. Purpose of the prototyping is to check the resources used by the router design on FPGA. Resources occupied by the router are also compared with NIOS II CPU. Design is also tested by connecting NIOS II with router to send packets to router and then receive it from the same router by NIOS II.

1.5 Thesis Layout

Detailed discussion about NoC and routing algorithms has been described in chapter 2. This chapter also includes discussion about router architecture and their specification. In the end it describes about which tools have been used for design prototyping.

Design option is an important issue in the field of electronics. It varies from place to place and depends where the system would be used. Sometimes pure hardware design is good and sometimes mixes of software and hardware design is good. Since this thesis is about router design, a detailed discussion about different kinds of routers has been described in chapter 3. In the end, a comparison is given between different kinds of router design options.

Chapter 4 describes the detailed design of router architecture which is also the area of the thesis. Different Path Table storage options are also described in this chapter.

In chapter 5, Improved router design, Simulation results obtained from first design and improved design are described in this chapter.

Prototyping results obtained from design by using Altera FPGA DE2 board are described in chapter 6. This chapter also includes the comparison of resources used by the JBR router and NIOSII on FPGA.

Chapter 7 describes the conclusion of thesis work. Future work is also described in this chapter.
2 Theoretical Background

In this chapter, theoretical background of NoC platform, routing algorithms, router architectures and prototype of design has been described. Starting from NoC, we will discuss the basic concept of NoC taking from network communication to improve SoC and after that basic component of NoC. It is important to have knowledge of network communication to work in NoC. In the end detail discussion about prototyping will be described.

2.1 NOC Concept

Before starting about NoC, we will discuss the main problem in SoC due to which NoC idea came to existence. Before 2000, SoC interconnects have not been designed as networks. Core to core connections or Bus based connections were used in SoC for communication between different cores. Due to increase in number of cores in a silicon chip has affected the salability, resource utilization and speed of communication of the cores.

2.1.1 Core to Core Connections

In 1990s in a SoC all cores were directly connected with each other. In other way we can also say that these cores have point to point interconnection with each other. If a core wants to send data to another core, it sends that data directly to the destination there is no intermediate state. There is no arbiter inside the chip which can control the communication between the cores. Communication speed is very fast between the cores because of direct connection. For small number of cores it is best to use direct connections but complexity increase with the increase of number of cores. A large number of cores need large number of I/O pins to communicate with each other which results in large routing area, as well as large number of wires.

Figure 2.1: Core to core based connection
Direct interconnections have following disadvantages.

- Low scalability
- Low routing resources utilization since connections are dedicated for each pair
- Uncertainty regarding quality of signals

SoC using direction connections does not have the feature of scalability. It is difficult to add new cores in the system. Reuse of component is also a big issue for SoC. Each connection between the cores is used by self-connected cores. Other cores cannot use the same connection which reduces the utilization of routing resources. There are also no criteria to test the quality of signal as it can be uncertain because of increasing clock and changed geometry of wires which may increase the noise.

### 2.1.2 Bus Based Connections

In SoC while using direct interconnections, the main problem is increase in number of I/O pins as well as cost with the increase in chip area. To avoid these problems bus based interconnection has been introduced. Instead of connecting the cores directly, all cores are directly connected to the bus which behaves as the shared medium between all cores. Cores in SoC are connected to the bus through an interface. An arbiter is used to handle the communication between the cores. If any core wants to communicate with other core then it has to wait until bus is free to use and arbiter generates some control signal to handle the bus and give access to the relevant cores. A bus based communication interface between different IP is shown in the Figure 2.2.
In this figure, mouse controller receives the data from the mouse and sends it to the Nios II CPU through Avalon Bus or vice versa. Then CPU sends the mouse movement data to the VGA through Avalon Bus which generates the mouse cursor on the screen.

One of the advantages of using bus is scalability as compared to direct interconnections. Number of wires and I/O pins are also reduced which results in the reduction of chip area. Still there are some flaws which are described below.

- Scalability reduced with increase of cores after a certain limit
- Low speed

As the number of cores increased beyond the limit, it affects the performance and scalability of the system. Since all cores are sharing the same medium therefore only two cores can communicate with each other at a time. Due to arbitration and increase in length of the bus speed of communication is also decreased.

### 2.1.3 Router Based Connections

As it has been described above that SoC has some major problems as we increase the area of the chip. To overcome these flaws a network based platform has been introduced which is known as Network on Chip (NoC). In NoC, network related concepts are used. For example, instead of using direct connections or busses, a network of routers are used to connect cores with each other through interface known as RNI (Router to Network Interface). Data is sent from one core to other core in the form of packets through routers.
Theoretical Background

NoC based systems have the following advantages.

- High scalability
- High performance
- Low cost

One of the advantages of the NoC is high scalability as compared to the bus based systems which allows the reuse of cores. NoC based systems also have high performance, low cost and short time to market.

2.2 NoC Terminologies and Switching Techniques

In this section, we will discuss about some commonly used terminologies and switching techniques which are suitable for NoC router.

Message

A message is a real data which is sent from source to destination. It could be of a fixed size or of any variable length depending on the protocol used. A message can have many packets.

Packets

Message can be broken into different parts due to large amount of data. These fixed size parts are known as packets. Packet can move from source to destination through routers independently. Packet needs some information before sending it into the network like some control information and destination address etc. This information is attached to the packet and known as header of the packet. Rest of the packet contains the original data and also known as payload. A packet can also be divided into smaller parts known as flits.

Flit (Flow Control Digit)

Flit also called flow control digit is the smallest unit of the message which has fixed length. The storage elements in the network routers have the same length as of the flits.

Switching Techniques

Switches are used in the networks to control the flow of data and manage the right path by following some predefined protocols. Data is sent from source to destination through switches. Performance of the network is very much dependent on the switching techniques used. There are few switching techniques which are used in the networks.

- Store and forward
- Circuit switching
- Wormhole switching
Theoretical Background

In store and forward switching technique, a complete packet is sent from one router to another. No packet can move to the next router until it is completely received by the previous router. This results in higher delay in communication. Also buffer size increases because of the complete packet storage.

In circuit switching, a physical connection is created from source to destination before starting the communication. Data travel from source to destination through these established connections and after successful transmission of data, connection is removed. This switching technique has disadvantage in the form of resource utilization which are not completely used.

In wormhole switching technique, a packet is divided into flits and sent towards the destination through routers. There are three types of flits known as Head flit, Body flit and End flit. First Head flit is sent into the network which locks the path for rest of the flits. Body flit which contains the actual data or payload follows the same path as locked by the Head flit. Body flit may be more than one depending on the size of data. End flit is sent in the end which means there are no more Body flits. End flit also unlocks the path. Since flits are moved like worm in the network that’s why it is called wormhole switching. One of the advantages of wormhole switching technique is their reduced buffer size. There is no need of buffer size equal to a complete packet.

2.3 Components of NOC Architecture

There are three main components of NoC.

- Resource
- Resource Network Interface (RNI)
- Router

Components of NoC architecture are shown in Figure 2.4. We will discuss these components one by one.

Resource

Resource is the basic component of the platform using a NoC which is sometimes called core. These cores can be processors, I/O controllers, memory blocks, DSPs, video controller, audio controllers Tx/Rx and many others. A core could be a new component manufactured by the vendor or it could be the reuse of an old component. Sometimes pre-designed components are available free of cost but sometimes one needs to buy them. These pre-designed components are also known as Intellectual Property (IP).

Resource Network Interface (RNI)

RNI is the second component in NoC. RNI behave as an interface between resource and router that is why it is called Resource Network Interface. Resource sends packet to the router through RNI. The RNI in NoC and network card in internet have the similar purpose. RNI is divided into two parts. One part is known as resource dependent part and other is known as resource independent
part. As it is clear from the name that resource dependent part deals with the resources while resource independent part deals with router. The purpose of resource dependent part is to receive packet from the resource, flitize the packet and send it to the router. Flitization is the process to dividing packet into flits. If there is no space in router then flits are stored in buffer in RNI and waits until router is ready to receive the flits. Similarly the purpose of resource independent part is to receive the flits from the router, deflitize it and send it to the resource. The process of converting flits into packet is known as deflitization. Flits are stored in a buffer if resource is not ready to receive the packet. Path Table can also be stored in RNI to put the path information inside the Head flit. A thesis work has already been done on RNI [5].

![Diagram of NoC components](image)

**Figure 2.4: NoC components**

**Router**

Router is the most important component in the NoC. All resources communicate with each other through router. Purpose of the router is to receive packet from the resource and send it to another resource through router or multiple routers depending on how far is the destination. Design of the router depends on the routing algorithm used. Choice of the algorithm can make the router design simpler and less costly.

**2.4 Routing Algorithm**

**2.4.1 Distributed vs. Source routing**

Routing in NoC can be classified in many ways. But here we will discuss two way classifications. Routing can be classified into source routing and distributed routing. These will be discussed one by one.
Source routing

In source routing, path followed by the flit from source to destination is predetermined. All the path information from source to destination is stored in the Head flit. Router takes decision by looking at the Head flit. There is no extra logic used to calculate the path inside the router. Path information is calculated inside the source before sending flits into the network. For this purpose Path Tables are stored inside the source core or resource network interface (RNI). These Path Tables contain the complete path information for a specific destination in the network. Path information is calculated by applying routing algorithms [10]. When a core wants to send the packet into the network, it stores path information into packet header.

Distributed routing

Distributed routing can be static/deterministic or adaptive. In static routing the path between a pair is fixed. In adaptive routing there are multiple paths and actual path taken depends on network condition. In distributed routing, routing logic is implemented inside the router because there is no information about the path inside the header packet. Only destination address and some control bits are added in header packet. Each router in the network has information about the neighbouring router which is stored in the form of tables. Design for router is complex in this routing technique because extra hardware is needed for routing logic and bigger memories are required to store the routing tables.

2.4.2 Junction Based Routing (JBR)

Source routing is very suitable for NoC due to its deterministic nature. But there is need of change in this algorithm due to its packet overhead. Drawback in this algorithm is that header of the packet contains all information of the path from source to destination which means it needs more buffer size to store the packet. A large buffer size means it increases the cost of the router.

A new algorithm called Junction Based Source Routing has been developed to overcome this flaw. The basic idea is to limit the required path information to be stored in every packet to a small number of bits (corresponding to only a few hops). The large distance is covered by going through intermediate temporary destinations (called Junctions) such that sub-paths (from source to a junction, junction to another junction, and junction to the destination) are always smaller than or equal to a maximum hop count. If a packet needs to go through a junction (or many junctions) the source just appends path information from source to the junction. On reaching the junction, the packet picks up path information to reach the destination (or another junction) from this junction [18].

Real Life Example

Idea of junction based source routing resembles with real life train system. In train system it is very difficult to build big platform at every station from where you can find any train for any city. Development cost is increased for each big platform in every city and for some small cities resource utilization is not efficient like big
Theoretical Background

cities. To overcome this problem junctions are build in big cities. Purpose of the Junction is to give facility to passengers for the small cities or the places which are far away from the small cities. Suppose a passenger wants to go to a city or village which is not far away from its home city then he can go by direct train. But if he wants to go to a village or city which is not near then first he has to go to the Junction from there he will change the train for destination. If destination is is still far away then there might be possibility to change the train twice through another Junction. Using this idea make the source routing more efficient in term of cost.

A packet reaching a Junction may or may not use path information stored in the Junction. There are three distinct cases:

i. If the destination of the packet is the resource connected to the Junction itself, then it should be routed to the resource through RNI.

Figure 2.5: A 7X7 Junction Based Network on Chip based system
ii. If the destination is not very far and the packet header has enough information to reach the destination, then the router forwards the packet just looking at the relevant field in the header.

iii. If the destination is far, then the Junction will be the intermediate destination. This will be clear from the relevant field in the header. In this case, Junction modifies the path information in the packet header for onward journey to the destination, if possible otherwise to another junction as intermediate destination.

A 7X7 Junction Based routing NoC with three different distinct cases is shown in Figure 2.5.

In Figure 2.5, green arrow shows the packet movement when the destination is not far away, red arrow shows the packet movement when the destination is connected to the junction and blue arrow shows the packet movement through junction when the destination is far away from source.

### 2.4.3 Comparison of Header Format

Head flit format of source routing and junction based routing is shown in the Figures 2.6 and 2.7 respectively.

![Figure 2.6: Format of Head flit in Source Routing](image)

![Figure 2.7: Format of Head flit in Junction Based Routing](image)

Source routing and junction based source routing both are 34 bits long and have first two bits as flit type which must be “00” for Head flit. Source routing Head flit used six bits for source address which means it can carry the source address of maximum 8X8 mesh NoC. The information about source address is necessary for the destination to know where it came from. Twenty six bits are reserved for path information which carries total information from source to destination. Since there are five output ports in a router so two bits are enough for each output port.
selection. It can also say that each hop needs two bits. Total 13 hops can be stored in Head flit which is enough for 7X7 mesh topology NoC. It is a big problem for larger network to reach the destination with such a limited number of hops.

Junction based routing has some extra field in the Head flit which are Resource Bit (RB), Junction Bit (JB), Destination address and payload. It does not have any source address because we have assumed that when we reach a junction to pick up path for the next segment we do not need to remember where we came from. Resource Bit (RB) is used by the resource when it needs path information from Path Table. It is not necessary for the router to work as junction. If RB is ‘1’ then flit will get the path information from the Path Table first then move towards the destination. If Resource Bit (RB) is ‘0’ then its mean Head flit already has the path information. Junction Bit (JB) is used when Head flit has to change its path information from junction router. If Junction Bit (JB) is ‘0’ then its mean Head flit contains the path information till destination. No need to change the path information because destination is not far away from the source. Sixteen bits are reserved for path information. Total eight hops are allowed from source to destination. If destination is more than eight hops then it must go to the destination through junction. Eight bits are reserved for destination address. Max of 16X16 mesh topology is allowed in the network. Rest of the six bits is used for payload. These bits can also be used for sending signals. Head flit locks the corresponding path for Body and End flits in both kind of routing.

2.5 Router Architecture

2.5.1 Functionality of router

Router plays an important role in networks. Its main purpose is to receive packets from the sender, take decision by executing an algorithm and send the packet towards the destination. While going towards the destination, a packet may have to pass from several routers. Design of a router is very much depends on the routing algorithm used. A suitable routing algorithm makes the router less complex and faster. A NoC based router has the same functionality. Router architecture has three main components.

- Input buffer
- Cross Bar
- Arbitration and Control

2.5.2 Input buffer

Input buffer is the temporary storage component. Main purpose of input buffer is to temporarily store the incoming packets if the router is busy or desire output is not empty. Input buffer is controlled by the Arbiter and Control block.
2.5.3 Cross Bar Switch

Cross Bar facilitates to connect an Input Port to any output port. Cross Bar is a combination logic which makes switching between different ports by receiving signals from Arbiter and Control block.

2.5.4 Arbitration and Control

Arbiter and Control is the most important component of the router. It behaves like its brain. All routing decisions are taken inside this component. It resolves conflicts for simultaneous requests to an output port. It also organizes the flow of packets through the router. This component is actually the implementation of routing algorithm. It controls the input buffer by sending some signals to it. When a desired output is empty then it selects the desired output by sending the selection signals to the Cross Bar and sends a control signal to the input buffer to pass through the Cross Bar to the desire output. Typical router architecture is shown in Figure 2.8.

![Router Architecture](image)

**Figure 2.8: Router architecture**

Functionality can be implemented in many ways. Two important ways are called:

- Hardware router
- Software router: using general purpose processor (GPP) or application specific processor (ASP)

We will discuss them in details in the next chapter.
2.6 Handshaking Protocol

Since each router has to be connected with other routers or different resources. It is difficult for both resources and router to have same communication speed. As we know processors are software based and speed of software is slower than the speed of hardware. Therefore a protocol is necessary for the communication of different components of NoC. In our design we have used handshaking protocol to communicate between different routers and resources.

According to this protocol if any router or resource wants to send a flit to another router or resource then it will check if the buffer of next router or resource has space or it is already full. This can be checked by reading RTR (ready to receive) signal. If it is ‘1’ then its mean buffer has space to store flit but if it is ‘0’ then its mean buffer is already full and there is no space. If there is space then a Put signal is sent to the receiver side until RTR signal does not become low. Data is also sent to the receiver at the same time. After saving data receiver makes the RTR signal to low and Put signal is also become low after getting RTR signal low. After receiving low Put signal by the receiver it will check the empty space in buffer. If there is any space left in buffer RTR signal will go high which means buffer has space to store further flit.

Handshaking protocol is a simple protocol but it is a time consuming process. But it ensures that no flit will be lost during communication. There is no need of this protocol if all the components of NoC are synchronous with each other but mostly resources are not synchronous with routers or RNI. Handshaking protocol in the form of signal is shown in the Figure 2.9.

![Figure 2.9: Handshaking protocol between sender and receiver](image-url)
2.7 Gaisler’s two process Method

The most common style for modelling a Design in VHDL is known as “Data flow” style. These kind of modelling styles are good when designs are simple but with the increase in design complexity it is difficult to read and understand the VHDL data flow code as VHDL contains a large number of concurrent statements and concurrent processes. Due to this characteristic it is difficult to guess the execution order of statements.

Due to this problem, Jiri Gaisler has introduced a new coding style to improve the readability and uniform way of encoding the algorithm. This coding style is known as Gaisler’s two process method [14]. According to this method, VHDL code consists of two concurrent processes. One process is used for combination (asynchronous) logic and the other process is used for sequential logic (Registers). Generic two process circuit is shown in the Figure 2.10.

Gaisler’s two process method is used to achieve the following goals.

- To provide uniform algorithm encoding
- To make abstraction level high
- To improve readability
- To distinguish between combinational logic and sequential logic.
- To improve simulation speed
- Common model for simulation and synthesis

We have used Gaisler’s two process method to design various component of our design.

![Figure 2.10: Generic two process circuit [14]](image-url)
2.8 Prototype of Design

2.8.1 FPGA Technology

In the past, Application Specific Integrated Circuit (ASIC) was used for the specific purpose designs. Due to their high cost and non-reprogrammable behavior, FPGA has become an emerging technology now a day. FPGA is an abbreviation of Field Programmable Gate Array. It can be configured or programmed on the circuit board. FPGA is a technology which is used for prototyping of different designs. Programmable logic blocks and programmable interconnects allow the FPGA to be used and re-used in many different applications. Due to their reprogrammable behavior, FPGA is more cost effective as compared to the ASIC.

FPGA consists of combinational blocks and special blocks. Combinational blocks include small number of gates and flip flops while special blocks include RAM blocks and Multipliers etc. In Xilinx, combination blocks are known as configurable logic blocks (CLB) while in Altera, it is known as Logic Blocks (LB). In this project work, we have selected Altera DE2 for prototyping of our design.

Altera has different families but their basic architecture is same. A logic block consists of four input look up table (LUT) and a flip flop. The outputs of LUT and flip flop are the input of a MUX. LUT is a memory having different combination values that can implement any function of four variables. Altera cyclone II basic architecture is shown in Figure 2.11.

![Cyclone II block diagram](image)

Figure 2.11: Cyclone II block diagram [12]
Different devices belonging to cyclone II family is shown in Table 2.1. It includes total number of LEs, Memory blocks, Multipliers, PLLs and I/O Pins.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LEs</td>
<td>4,608</td>
<td>8,256</td>
<td>14,448</td>
<td>18,752</td>
<td>33,216</td>
<td>50,528</td>
<td>68,416</td>
</tr>
<tr>
<td>M4K RAM Blocks (4 kbits + 512 Parity Bits)</td>
<td>26</td>
<td>36</td>
<td>52</td>
<td>52</td>
<td>105</td>
<td>129</td>
<td>250</td>
</tr>
<tr>
<td>Embedded Memory (Kbits)</td>
<td>117</td>
<td>162</td>
<td>234</td>
<td>234</td>
<td>473</td>
<td>581</td>
<td>1,125</td>
</tr>
<tr>
<td>Total RAM bits</td>
<td>119,808</td>
<td>165,888</td>
<td>239,616</td>
<td>239,616</td>
<td>483,840</td>
<td>594,432</td>
<td>1,152,000</td>
</tr>
<tr>
<td>Embedded Multipliers</td>
<td>13</td>
<td>18</td>
<td>26</td>
<td>26</td>
<td>35</td>
<td>86</td>
<td>150</td>
</tr>
<tr>
<td>PLLs</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Maximum User I/O Pins</td>
<td>158</td>
<td>182</td>
<td>315</td>
<td>315</td>
<td>475</td>
<td>450</td>
<td>622</td>
</tr>
</tbody>
</table>

Table 2.1: cyclone II Family overview [19]

### 2.8.2 FPGA Prototyping Design Flow

Before implementing the design on FPGA chip, it will pass through different stages. Quartus II development tool is used for this purpose which compiles the program by checking different requirements. Quartus II software design flow is shown in the Figure 2.12.

Design files are imported into Quartus II software tool which may be a VHDL files a Verilog HDL file. When we start compilation of our design then Quartus II software will analyse the design first for errors. If there is no error in the design then it will also be synthesized. Pin assignment analysis is also the part of this process. After analysis and synthesis a report of summary is created which describe the resource utilization. Quartus II will perform fitter after finishing the analysis and synthesis. Fitter perform rout and place to fit the design into the target device. After fitting a complete resource utilization comparison summary show the details. Timing analysis measures the delay along the various timing paths and verifies the performance and operation of the design [15]. After successful compilation, compiler will generate .sof/.pof files which mean design is ready to program the chip.
Figure 2.12: QuartusII Design flow [15]
3 Router Design Options

For any design it is important to keep in mind about some pros and cons and their effect on the system. Router is an important component in network. Speed and cost of the network mostly depends on the router used. There are different options of router design. For example, hardware router, network processor or software router and table based router. In this chapter we will discuss these different options, compare them and try to figure out which option is the best for our router design.

3.1 Hardware Router

Router is an important component in NoC as it provides routes for the communication between different cores. A router consists of registers, switches, arbitration and control logic that collectively implement the routing and flow control function required to buffer and forward flits to their destination [1]. Most of the NoC router proposals are of hardware type. In the past, hardware based router designs have been introduced having different architectures because architecture of hardware design depends on the algorithm used. For example, router design for source routing is different from the router design for distributed routing. In source routing, complete route information is stored in the Head flit by some pre calculations. Router takes decision by looking at the path information stored in Head flit. There is no function for path computation inside the router. While in distributed routing, decision is taken inside the router. These decisions can be taken by implementing hardware functions or by looking at the routing tables stored inside the router.

Hardware based routers are used for a single purpose. Since routers are designed especially for NoC therefore the purpose of the router is to route the flit into an appropriate direction according to the routing algorithm used as describe above. Due to their single purpose characteristics, the performance of these routers is very high. These routers are also small in size. One of the disadvantages of these routers is their inflexibility. Since router is specifically design to implement an algorithm, a slight change in the routing algorithm forces the designer to rebuild the complete architecture again. These routers have very high speed because of hardware implementation as well as low cost due to their single purpose. But development cost is very high, which is the main drawback of hardware router.

The idea of router architecture described in [10] has been used for this thesis to develop a junction based router. New components have been added to control the junction of the network. Developing new router architecture from scratch is time consuming process. It is better idea to reuse the previous concepts which helps to reduce the design time.

3.1.1 Architecture

The router architecture shown in Figure 3.1 is a typical NoC router. Design of a router depends on the routing algorithm used. In source routing, router receives
flits from neighboring router or from core through RNI then takes decision by looking at the Head flit and sends the flit towards the required port. In distributed routing, routing decision is taken inside the router through implemented routing functions. Routers based on source routing have simpler architecture as compared to the distributed routing based architecture because there is no need to take routing decisions inside the router. Main components in the router architecture are as follow.

- Arbiter and Control
- Cross Bar
- Input buffer

![Figure 3.1: A typical NoC Router](image)

### 3.1.1.1 Arbitration and Control

This is the main component of the router architecture. It is not wrong if we say it is the brain of the router. All decisions are taken inside this component. Main responsibility of Arbiter and Control is to control the communication between different input ports to different output ports. It also resolves the contention problem between the inputs ports which desire the same output port by giving them priority. It selects the output port by sending signal to Cross Bar. Locking and unlocking of path is also its responsibility. It locks the path after passing the Head flit and unlocks the path after sending the End flit.
3.1.1.2 Cross Bar

Cross Bar is the collection of multiplexers through which flits are sent to the outside of the router. The purpose of the Cross Bar is to receive signals from the Arbiter and Control to select the output direction line for the relevant input port. A flit is transferred from an input port buffer to an output through Cross Bar.

3.1.1.3 Input Buffer

The purpose of input buffer is to store flits coming from different routers or coming from the core. Input buffers could be register based or memory based depending on the size of the buffer. If buffer size is small then register based buffers are good but if buffer size is large then it is good to use memory based buffers. When input buffer receives signal from Arbiter and Control then it sends flit to the Cross Bar.

3.2 Network Processor and Software Router

Second option for designing a router will be to use a general purpose processor (GPP) or an application specific processor especially suitable for router functionality. In both cases router functionality is implemented by writing software for this processor. This application specific processor is also called Network processor [11]. Because of their application specific nature, the network processors have new instruction set as compared to GPP. GPP are more flexible than network processors because they can be used for any purpose by changing the software but these processors have very high latency and require large area. On the other hand, network processors have low latency and needs less area as compared to GPP because of their application specific nature. These network processors and software routers are very costly because they need large memory to store the software for specific applications.

NoC is based on computer networks ideas. These ideas are always used to improve NoC’s performance. Network processors are specific purpose processors and therefore these processors are used to implement routers in computer networks. These processors have also become part of NoC router design and therefore these processors are also known as Micro-Network Processor (μNP). A thesis work related to μNP has been described in [11]. Processor based on μNP has its own instruction set.

Example of Instructions

μNP has almost all the instructions of GPP but also has some special instructions for routing functions. An example of a routing instruction is QRT (Quick route). This instruction set the data path in μNP to move a flit from an input port to an output port [11]. There are also instructions for assigning priorities to various input ports as well as locking or unlocking output port for multi-flit packets.

μNP is a multi-threaded processor architecture as shown in the Figure 3.2. One program thread handles one input port therefore μNP has five threads (PC counters) for ports and one thread (PC counter) for general configuration and control.
Latency

Performance of a router depends on the latency of the packet. Performance is high if latency of the packet is low. Latency of the packet means that time taken by the packet from its entrance into the router until the time of its leaving the router. Performance of the hardware router is high as compared to the software router and µNP. Similarly performance of µNP is better than software routers. Reason is that hardware routers are faster than the µNP and software routers and therefore they have low latency and high performance. µNP are not good as hardware router but they are better than software routers. µNP has latency higher than hardware router but very less than software routers and their performance is also very good as compared to the software routers. Latency of different routers in terms of clock cycles is given in the following Table 3.1.

<table>
<thead>
<tr>
<th>Routers</th>
<th>Latency (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>2-5</td>
</tr>
<tr>
<td>Software</td>
<td>&gt;100</td>
</tr>
<tr>
<td>µNP</td>
<td>&lt; 20</td>
</tr>
</tbody>
</table>

Table 3.1: Latency of different routers
Latency vs. performance graph is shown in Figure 3.3. The purpose of the graph is only to highlight the large differences in latencies of various router implementations.

**Flexibility**

Hardware routers are not flexible like software routers or µNP. Because if there is any change in routing algorithm, all the design will be changed which leads to design a new one. Software routers and µNP both are software based which gives a great flexibility. Any change in routing algorithm can be fixed by changing only software. Software routers are more flexible than µNP because software routers are General Purpose (GPP) which gives more flexibility to deal with different kind of programs. On the other hand µNP are application specific which restrict them to deal with only routing algorithms. Flexibility vs. performance graph is shown in Figure 3.3.

![Latency/Flexibility vs. Performance graph](image)

Figure 3.3: Latency/Flexibility vs. Performance graph

### 3.3 Table Based Router

Third option for router design is table based router. These routers are similar to hardware router. The only difference is that routing functions are implemented through tables. In other way we can say that routers take decision by looking at tables. These routers are fast like hardware routers because no software programs are used inside the router. Therefore these routers are not flexible like network processors and software routers but still they are better than hardware router in term of flexibility because routing algorithms are implemented through tables. Any change in routing algorithm causes to change the table information only. These routers need large memories to store routing algorithm. Due to this reason,
table based routers are very costly. Still they have low latency because of hardware used. But altogether the performance of these routers is very good.

3.4 Comparison

A comparison between all three router design options is shown in Table 3.2.

<table>
<thead>
<tr>
<th>Router Designs</th>
<th>Hardware Router</th>
<th>Network Processors</th>
<th>Table Based Routers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Cost</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Latency</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 3.2: Comparison between different router design options
4 Router Design for Junction Based Routing (JBR)

Source routing is very suitable for NoC platforms and has many advantages over distributed routing algorithms [1] [2]. But there is a drawback in this algorithm that header of the packet contains all information of the path from source to destination which means it needs more buffer size to store the packet which results in more cost.

A new algorithm called Junction Based Source Routing has been developed to overcome this flaw. The basic idea is to limit the required path information to be stored in every packet to a small number of bits (corresponding to only a few hops). The large distance is covered by going through intermediate temporary destinations (called Junctions) such that sub-paths (from source to a junction, junction to another junction, and junction to the destination) are always smaller than or equal to a maximum hop count. If a packet needs to go through a junction (or many junctions) the source just appends path information from source to the junction. On reaching the junction, the packet picks up path information to reach the destination (or another junction) from this junction.

Router is the most important component for design of communication back bone of a NoC system (like any other network). The design of a NoC router depends on various aspects of NoC architecture and the performance requirement. A NoC employing junction based routing will have two types of router designs: one for the Normal Router and the other for the Junctions. The Junction Router must contain path information, in tables, to reach any destination. There are various possibilities of implementing these tables, either in the router itself, in the resource network interface (RNI) or in the resource (core). A packet reaching a junction may or may not use path information stored in the Junction. There are three distinct cases:

i. If the destination of the packet is the resource connected to the Junction itself, then it should be routed to the resource through RNI.

ii. If the destination is not very far and the packet header has enough information to reach the destination, then the router forwards the packet just by looking at the relevant field in the header.

iii. If the destination is far, then the junction will be the intermediate destination. This will be clear from the relevant field in the header. In this case, Junction modifies the path information in the packet header for onward journey to the destination, if possible otherwise to another junction as intermediate destination.

This chapter deals with the different router organizations having different Path Table storage options and design of the suitable hardware router. There are several Path Table storage options which give the different router organizations.
First we will compare these organizations and then proposed a suitable design for this thesis.

### 4.1 Path Table Storage Options

Path Table storage plays an important role in router design. A good decision makes the whole system design more efficient, less costly and less complex. Although it is difficult to achieve best design but a better option gives better results.

There are four different Path Table storage options.

i. Separate Path Table in router for each Input Port and one Path Table in RNI

ii. One common Path Table for all Input Ports in router and one Path Table in RNI.

iii. One common Path Table in RNI of connected core.

iv. One Path Table in router for all Input Ports and core

#### 4.1.1 Path Table in router for each Input Port and one Path Table in RNI

In this design option, four Path Tables are stored inside the router. As we have described earlier that Path Tables have the complete path information to reach any destination. Therefore Head flits coming from different routers, which are situated at different directions (like North, South, West and East), to the four Input Ports of the junction router get path information from their own Path Table while Core or Resource gets its path information from the Path Table stored inside the RNI. Router organization for the first design option is shown in Figure 4.1.

Head flit gets its path information from the table stored in the RNI before sending into the network. As describe earlier that two kinds of routers need for Junction Based routing, one is called normal router and other is called junction router. In this design option, only junction router needs Path Table. There is no need to store Path Table inside the normal router. If the destination is far away from the source then flits are sent to the destination through the junction. At junction, Head flit needs to change its path information. Since every Input Port has its own Path Table so every port will work independently from other ports. Due to large traffic there is always congestion problem in the network. If there are limited Path Tables then every packet will try to access the same table which results in low flit speed. Since every Input Port has its own Path Table there is no congestion among flits which results in high flit speed.

The main advantage of this organization over other organizations is that each Path Table has to deal with only one input port which makes the design simple. Since every Path Table has to store complete path information about the network it
means large memories are required to store four path tables which makes the router most costly.

![Router Design for Junction Based Routing (JBR)](image)

Figure 4.1: Separate Path Table in router for each Input Port and one Path Table in RNI

### 4.1.2 Common Path Table for all Input Ports in router and one Path Table in RNI

In this design option, one Path Table is stored inside the router. Head flits coming from different routers to the four Input Ports of the junction router get the path information from common Path Table. One Path Table is stored in RNI to the connected Resource or core. Resource gets its path information from the table stored inside the RNI. Router organization for second option is shown in the Figure 4.2.

If there is only one request from any of the four Input Port, there will be no conflict to access the path information of desire destination from the Path Table but if there are more than one request from Input Ports at the same time then conflict occurs between different ports. This conflict can be solved by assigning different priorities to each Input Port. High priority request will access the Path Table before lower priorities requests. The condition is worst when all four Input Ports trying to access the Path Table at the same time which affects the speed of the router. This router organization has less speed as compared to the first option because of the congestion problem. Input port connected to RNI will not use the router Path Table because RNI already has Path Table for core. Advantage of using Path Table in RNI is that router has less number of Input Ports to deal with. Design complexity is increased due to handling the common Path Table for four
Input Ports but this organization is more cost effective as compared to first organization.

![Diagram](image_url)

**Figure 4.2:** Common Path Table for all Input Ports in router and one Path Table in RNI

### 4.1.3 Common Path Table in RNI

In this option, a common Path Table is stored inside the RNI of connected core for all four Input Ports as well as for Resource or core. No Path Table is stored inside the router. All four Input Ports and connected core get path information from the common table stored in RNI. The router organization for this third design option is shown in the Figure 4.3.

In this option, both normal router as well as junction router does not have any Path Table. If any of the Input Port wants to access the Path Table, it has to go in RNI first to access the table. This will make the router slow but if requests are more than one to access the Path Table then the speed of the router becomes even slower. Speed of the router becomes worst when all Input Ports and connected core try to access the Path Table at the same time. This router organization is the slowest as compared to other organizations. Since router contains no Path Table therefore it is least costly as compared to the other router. This design makes the router less complex as compared to other options but junction router needs extra logic to solve the contention problem.
4.1.4 Only One Path Table in router

In this design option, a common Path Table is stored inside the router for all four Input Ports as well as for Resource. No Path Table is stored inside the RNI. The router organization for fourth design option is shown in the Figure 4.4.
Core and all four Input Ports coming from different directions take path information from the Path Table inside the router. Since Resource also has to access the Path Table from the router which gives the advantage of making a common design for normal router as well as for junction router. Router organization is almost similar to the second option described in section 4.1.2. The only difference is that core has to send the Head flit to the router to get path information from Path Table. It means all five Input Ports are trying to access the Path Table at the same time and its speed become slightly less than second design option. The speed of the router is good as compared to third option because there is no need to go inside the RNI. Although router design is complex but one of the advantages of this design is that the overall system cost is very less. This matches also our objective of the thesis to design a JBR router with lowest cost and less complexity but having normal speed. In this thesis our focus would be on this option.

Comparison between different organizations is shown in Table 4.1.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Router Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
</tr>
<tr>
<td>Speed</td>
<td>High</td>
</tr>
<tr>
<td>System Cost</td>
<td>High</td>
</tr>
<tr>
<td>Design Complexity</td>
<td>simple</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison between different router organizations

### 4.2 Packet Format

In RNI, a packet is divided into equal size parts called flits. The process of dividing the packet into flits is also known as flitization. Each flit has a fixed length of 34 bits. There are four types of flits.

1) Head flit
2) Body flit
3) End flit
4) Full flit
First flit is always a Head flit. Second flit is Body flit which follows the Head flit. Body flit may be more than one depending on the size of the packet. After sending all Body flits, End flit is sent in the last which shows that no more flits for the corresponding packet. Full flit is used when packet has less than one byte of payload. There is no need to send Body or End flit after sending full flit. To differentiate the type of the flit, two bits are reserved in each flit. Flit type and their corresponding encoded code are shown in the following Table 4.2.

<table>
<thead>
<tr>
<th>Flit type</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head flit</td>
<td>00</td>
</tr>
<tr>
<td>Body flit</td>
<td>01</td>
</tr>
<tr>
<td>End flit</td>
<td>10</td>
</tr>
<tr>
<td>Full flit</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4.2: Flit types and their Encoding

Now we will discuss format of each type of flit.

**4.2.1 Flit format**

**Head flit**

Head flit contains 34 bits in total. First two bits are used for flit type which must be “00” for Head flit. One bit is reserved for resource. In this thesis, we are using one Path Table in JB router. When a resource wants to get path information from the table then it has to send the Head flit to router through RNI with Resource Bit (RB) equal to ‘1’. Which means that flit will get the path information first from the Path Table inside the router before moving towards the destination. If Resource Bit (RB) is ‘0’ then its mean Head flit already has the path information. One bit is reserved for junction. When Junction Bit (JB) is ‘1’, it means Head flit or Full flit has to change its path information from junction router. If Junction Bit (JB) is ‘0’ then its mean Head flit contains the path information till destination. No need to change the path information because destination is not far away from the source. Sixteen bits are reserved for path information. Total eight hops are allowed from source to destination. If destination is more than eight hops away then it must go to the destination through junction. Eight bits are reserved for destination address. This allows handling of 16X16 mesh topology network. Rests of the six bits are used for payload. These bits can be used for sending signals. Head flit locks the corresponding path for Body and End flits before moving towards the destination. Format of Head flit is shown below in Figure 4.5.
Body Flit

Size of the Body flit is 34 bits same as the Head flit. First two bits of the Body flits are reserved for the flit type which is always encoded by “01”. Rest of the 32 bits contains the payload. There may be more than one Body flits following the Head flit if there is large amount of payload. It is also possible that there is no Body flit after Head flit due to short payload. Body flit format is shown in Figure 4.6.

End Flit

Total number of bits in End flit is 34 like Head and Body flits. First two bits are reserved for flit type which must be “10”. Rest of the 32 bits are used to carry payload. End flit is the last flit which is sent after Body flits. It is also used to unlock the corresponding path. Format of End flit is shown in Figure 4.7.

Full Flit

Full flit format is similar to the Head flit. First two bits are flit type which is encoded as “11”. The only difference of Full flit from Head flit is that there is no Body or End flits after it. It is used to carry a small amount of payload like special control signals. Format of Full flit is shown below in Figure 4.8.
### 4.3 Design Overview

In this section, the architecture of the Junction Based router will be described. This router architecture supports both kinds of routers which mean that the router can be used as a normal router as well as a junction router.

Uniformity is good for design reuse and chip layout. It can support paths computed using different routing algorithm. Different routing algorithm may require junction at different places. Figure 4.9 shows the router architecture.

The router consists of four main blocks, which are:

1. Input Handler(IH)
2. Path Table Arbiter Component (PTAC)
3. Arbiter & Control
4. Cross Bar

In Figure 4.9, there are five copies of IH component which receive incoming data from different directions and send the data to other components if they are not busy. The purpose of IH component is to store data in buffer, check the relevant field in Head flit or Full flit for new path information, modify relevant field of Head or Full flit and send flit to other components. Body or End flits are sent to other components without modification.

Flits can be received from all directions (North, South, West, East and Resource) at the same time into input buffers of each IH components. It can be from resource through RNI or from other routers. Further decision to send a flit to Path Table Arbiter Component (PTAC) or Arbiter & Control is also taken in this component.

IH (Input Handler) component is divided into three sub blocks, which are given below.

a. Input Port
b. Header Modifier
c. Junction & Resource control

Detailed discussion of each block is available in the next section.

Path Table Arbiter Component (PTAC) is the second block in the router. The purpose of PTAC is to receive destination address from IH component and return relevant path information to IH component. Request for new path information can be sent from all IH components. This component also contains the Path Table. Like IH component, Path Table Arbiter Component (PTAC) is divided into sub blocks. PTAC has two sub blocks which are given below.

a. Path Table Arbiter
b. Path Table

Detailed discussion of each block is available in the next section.

Third main block in the router is Arbiter & Control. Arbiter & Control is the most important block of JBR router. We can also describe it as the brain of the router. All the path decisions are taken inside this block. Since we are using worm-hole switching technique, the path locking decision is taken inside this block. After taking the decision, if next router or resource is ready to receive the flit then complete flit is sent to the corresponding output through Cross Bar.

Cross Bar is the fourth block of the router. The purpose of Cross Bar is to establish connections between input to selected output by receiving signal from Arbiter & Control.

The router has been designed using VHDL as modeling language. IH component, PTA component, Cross Bar and Arbiter & Control are the main blocks which are designed in VHDL. IH_N, IH_S, IH_W, IH_E, IH_R are the same components connected with different inputs. So we are taking advantage of VHDL to create
hierarchy of components and interconnect them. Single clock signal is used for the complete design. There are several reasons to design different blocks. Firstly, it is easy to use copies of components and then integrate them. Secondly, it is easy to understand the functionality of each component. Thirdly, each component is implemented using one or several processes in VHDL and because of process concurrency each process in a component executes concurrently. Therefore all inputs of a router are managed at the same time without wasting any clock cycle.

4.4 Router Design Details

As described in the previous section that the main router is divided into blocks and then some of blocks are divided into sub blocks. In this section detailed discussion of each block and sub-block will be described. Each block or component has been designed in VHDL using Gasiler’s two process method which has been described earlier in second chapter.

4.4.1 Input Handler (IH)

One of the blocks in router is named as Input Handler. As described earlier, the purpose of IH is to store data in buffer, modify data or change the path information by looking at the relevant field in the Head flit, and send the modified data (Head flit or Full Flit) or unmodified data (Body or End Flit) to other blocks. Flit can be from Resource through RNI or from other routers. Decision to send flit into Path Table Arbiter Component (PTAC) or Arbiter & Control is taken in this block.

IH component is divided into three sub blocks as described below.

a. Input Port
b. Header Modifier
c. Junction & Resource Control

4.4.1.1 Input Port

Management of input buffer and interfacing is performed in this block. The Input Port performs the following functionality.

- To transfer the flits between two routers or resources by using handshaking protocol.
- To receive the flits and store them in input buffers
- To send the flits to the next blocks
- To send signal to the other routers or to the resource about the status of the input buffer (empty or full).
- To manage the input buffer
Figure 4.10 shows the block diagram of Input Port.

Input Port has five inputs and three outputs. Inputs are clock, Rst, Put, Get, and Data_in while outputs are RTR, Req, and Data_out. Data_in and Data_out both are 34 bits wide which are used for data transferring.

As describe above that one of the functionality of the input buffer is handshaking. Put and RTR signals are used for this purpose. Detail of handshaking protocol has been described before in second chapter. RTR (ready to receive) signal is ‘1’ when the buffer has space to receive flit or when the buffer is not full. RTR signal is ‘0’ when the buffer is full which means there is no space in buffer. When the previous router or resource wants to send data then it will send high pulse on Put signal. The process of writing data to buffer or memory is synchronous which means data is stored on rising edge of clock but reading data from buffer is asynchronous means data can be read regardless of the clock edge. When the first flit is stored in the buffer, Req signal becomes high. It means flit is available for the Header Modifier block that can be read through Data_out. Req signal remains high as long as there is flit in buffer. It becomes low when there is no flit in buffer. To read the next flit, a high pulse is sent on Get signal. There are two different kinds of input buffer.

- Memory based buffer
- Register based buffer

Although there are many different implementations possible for storing packets (flits) in the buffer. In this project, input buffer has been implemented using circular memory buffer. The reason is that Memory Buffer takes less hardware as compared to a set of registers. Although register based buffer are faster than memory based buffer but they are costly. Secondly Memory Buffers are easily scalable as compared to register based buffers. Memory Buffer and register based buffer are shown in Figure 4.11 and 4.12 respectively but our focus will be on Memory based buffer.
In circular memory buffer, there are two address registers which are named as Head and Tail. Head is used to read data and Tail is used to write data into buffer. When there is Put signal then data is written at the address pointed by the Tail. As soon as data is written in the buffer it is available to read. When other blocks want to read next data they will generate Get signal. In the beginning, Head and Tail addresses are same which means that buffer is empty. Memory buffer is also called circular buffer because after writing in all locations it will start from first location of the memory. If Tail value is continuously increasing and at a certain point, value of Tail register becomes equal to the value of Head register then it means buffer is full. Similarly if Head value is continuously increasing and at a certain point, value of Head becomes equal to the value of Tail then it means buffer is empty and there is no data in buffer. In the beginning, Req signal is low but as soon as there is data in memory, Req signal becomes high which is sent to the other blocks like Header Modifier and Junction & Resource Control. Depth of input buffer used is 8 and each location is 34 bits wide.
4.4.1.2 **Header Modifier**

Header Modifier is the second sub block in IH component. Main functionality of Header Modifier is described below.

- To receive flit from Input Port
- To take decision by checking flit type, Junction Bit (JB) and Resource Bit (RB)
- To modify Head flit
- To send Head, Body or End flit to Cross Bar
- To send flit type and direction bits to Arbiter & Control

Block diagram of Header Modifier is shown in Figure 4.13.

![Block diagram of Header Modifier](image)

It has seven inputs and five outputs. Inputs are clock, Rst, Req_buf2mod, Data_buf2mod, Data_jun2mod, Req_jun2mod and Get_arb2mod while outputs are Get_mod2buf, Accept, Data_mod2bar, Req_mod2arb and Data_mod2arb. 

*Data_buf2mod, Data_jun2mod and Data_mod2bar* are 34 bits wide and are used for data transformation while *Data_mod2arb* is 4 bits wide.

As soon as there is data in input buffer it is sent to the Header Modifier. A *Req_buf2mod* signal is generated after data is saved in buffer. When Header Modifier receives *Req_buf2mod* signal then first it will check flit type, Junction Bit (JB), Resource Bit (RB) and router type and take decision according to received flit. Router type is a Generic parameter set by the designer to make the router working as Junction or Normal router. When router type is ‘1’ then router works as Junction and when router type is ‘0’ then it works as Normal router. Checking of flit can be categorized into three parts.
i. Flit type is Head or Full and any of JB or RB is ‘1’ and router type is either ‘0’ or ‘1’

ii. Flit type is Head or Full and both JB or RB are ‘0’ and router type is either ‘0’ or ‘1’

iii. Flit type is Body or End

First case can be elaborated in this way.

a. When flit type is “00” (Head) or “11” (Full) and any one of the Junction or Resource Bit (RB) is ‘1’ and router type is ‘1’ (Junction).

b. When flit type is “00” (Head) or “11” (Full) and Resource Bit (RB) is ‘1’ and router type is ‘0’ (Normal).

Header Modifier will go into waiting state until Req_jun2mod signal as well as Head or Full flit with new path information is sent from Junction & Resource Control block. After receiving Req_jun2mod from Junction & Resource block, new Head or Full flit is sent to the Cross Bar and flit type and direction bits are sent to Arbiter & Control. At the same time Req_mod2arb signal is sent to Arbiter & Control so that it can take decision according to the direction bits.

In case of second statement it can be elaborated as.

a. Flit type is “00” (Head) or “11” (Full) and both Junction and Resource bits are ‘0’ and router type is ‘1’ (Junction).

b. When flit type is “00” (Head) or “11” (Full) and Resource Bit (RB) and router type both are ‘0’.

Path information in the Head flit is modified in such a way that first two bits of path information are sent behind the last bit of path information. And then this modified flit is sent to Cross Bar and flit type and first two bits of path information, also called direction bits, are sent to Arbiter & Control block. At the same time Req_mod2arb signal is sent to Arbiter & Control so that it can take decision according to the direction bits. The reason of modifying path information is that first two bits of path information are used by the previous router to take decision about direction. That’s why change is compulsory in next router for a new direction.

In the case of third statement, when flit type is “01” (Body) or “10” (End) there is no need to change anything. Flit type bits are sent to the Arbiter & Control and Req_mod2arb signal is generated, while on the other hand complete Body or End flit is sent to Cross Bar.

After sending flit type and direction bits to Arbiter & Control, Header Modifier goes to waiting state until there is a GetARB2mod signal from Arbiter & Control. After receiving GetARB2mod signal from Arbiter & Control, an Accept signal is sent to Junction and Resource Control and Getmod2buf signal is sent to Input Port by Header Modifier. Accept signal is sent to Junction & Resource Control block to leave its waiting state and deal with next flit if there is any request from Input Port.
Port. Similarly if there is any flit in buffer a Req_buf2mod signal is sent to Header Modifier by Input Port.

4.4.1.3 Junction & Resource Control

Junction & Resource control is the third sub block in IH component. Main functionality of Junction & Resource Control is described below.

- To receive flit from Input Port
- To send destination address to PTAC and receive new path information
- To send flit with new path information to Header Modified

Junction & Resource Control block is shown in Figure 4.14.

This block has seven inputs and four outputs. Inputs are Clock, Rst, Req_buf2jun, Data_buf2jun, Get_PTA2jun, New_pathinfo, Accept_mod2jun while outputs are Req_jun2PTA, Desadd_jun2PTA, Req_jun2mod and Data_jun2mod. Data_buf2jun and Data_jun2mod both are 34 bits wide and used for data transformation. Desadd_jun2PTA is 8 bits wide while New_pathinfo is 17 bits wide.

When there is Req_buf2jun signal from Input Port to Junction & Resource Control block then it will check the flit type, Junction Bit (JB), Resource Bit (RB) and router type and take decision according to received flit. Router type is a Generic parameter set by the designer to make the router working as Junction or Normal router as described before. Checking of flit can be categorized into two parts.
Router Design for Junction Based Routing (JBR)

i. Flit type is Head or Full and any of JB or RB is ‘1’ and router type is either ‘0’ or ‘1’

ii. Flit type is Body or End

In case of first statement

a. When flit type is “00” (Head) or “11” (Full) and any one of the Junction or Resource Bit (RB) is ‘1’ and router type is ‘1’ (Junction).

b. When flit type is “00” (Head) or “11” (Full) and Resource Bit (RB) is ‘1’ and router type is ‘0’ (Normal).

Junction & Resource Control block sends the destination address to Path Table Arbiter. A Req_jun2pta is also sent to the PTA. After sending request to PTA, J&R Control block goes to waiting state until there is a Get_pta2jun signal from PTA. Before sending the Get_pta2jun signal from PTA, new path information will also send to J&R Control block. After getting the signal, new path information is replaced in Head flit or Full flit. This new Head or Full flit is then sent to Header Modifier and a Req_jun2mod signal is also generated. After sending the signal, Junction & Resource Control block goes to waiting state until there is an Accept signal from Header Modifier. Junction & Resource Control block is sent into waiting state because it should not start to deal with next flit until previous flit is sent to the next router or resource. When J&R Control block receive Accept signal from Header Modifier it goes to its initial state to deal with next flit if available.

In case of Body or End flit, J&R Control block remains in waiting state until there is Head or Full flit from the Input port.

4.4.2 Path Table Arbiter Component (PTAC)

Path Table Arbiter Component is the second block of router. The purpose of Path Table Arbiter Component is to receive address from IH component and return relevant path information to IH component. Request for new path information can be sent from all IH components. Each request is priority based. Like IH component, Path Table Arbiter Component (PTAC) is divided into two sub blocks which are given below.

a. Path Table Arbiter (PTA)

b. Path Table (PT)

4.4.2.1 Path Table Arbiter

PTA block is the first sub block in PTAC block. Main functionality of PTA is described below.

- To handle all requests for new path information
- To send back new path information to the relevant Input Handler

Block diagram of PTA is shown in Figure 4.15
This block has thirteen inputs and seven outputs. Two inputs are clock and RST. Rest of the inputs coming from different IH components (North, South, West, East and Resource) are \( \text{Req}_{\text{jun2pta}}_{-}N, \text{Req}_{\text{jun2pta}}_{-}S, \text{Req}_{\text{jun2pta}}_{-}W, \text{Req}_{\text{jun2pta}}_{-}E, \text{Req}_{\text{jun2pta}}_{-}R, \text{Dadd}_{\text{jun2pta}}_{-}N, \text{Dadd}_{\text{jun2pta}}_{-}S, \text{Dadd}_{\text{jun2pta}}_{-}W, \text{Dadd}_{\text{jun2pta}}_{-}E \) and \( \text{Dadd}_{\text{jun2pta}}_{-}R \). One input coming from Path Table is \( \text{Readover}_{\text{PT2PTA}} \). Two outputs going towards Path Table are \( \text{Read}_{\text{PTA2PT}} \) and \( \text{Desadd}_{\text{PTA2PT}} \). Five outputs going towards five different IH components are \( \text{Get}_{\text{pta2jun}}_{-}N, \text{Get}_{\text{pta2jun}}_{-}S, \text{Get}_{\text{pta2jun}}_{-}W, \text{Get}_{\text{pta2jun}}_{-}E \) and \( \text{Get}_{\text{pta2jun}}_{-}R \). Width of each \( \text{Dadd}_{\text{jun2pta}} \) and \( \text{Desadd}_{\text{PTA2PT}} \) is 8 bits.

![Path Table Arbiter Diagram](image)

**Figure 4.15:** Path Table Arbiter

In Path Table Arbiter each \( \text{Req}_{\text{jun2pta}} \) signal is scanned one by one. If there is \( \text{Req}_{\text{jun2pta}} \) signal from any of the Junction & Resource Control block (like north, south, west, east or resource) then it will handle that request. PTA block will send the received destination address to Path Table and also generate a \( \text{Read} \) signal then it goes to waiting state until it receives the \( \text{Read}_{\text{over}} \) signal from Path Table block. After receiving \( \text{Read}_{\text{over}} \) signal, a \( \text{Get}_{\text{pta2jun}} \) signal is generated by PTA block which is sent to Junction & Resource Control block. After this PTA block goes to its initial state and again starts scanning the incoming request from Junction & Resource Control block of any direction. Requests are scanned according to the priority given to the direction. North is given highest priority, south is given 2\textsuperscript{nd} priority, west is given 3\textsuperscript{rd} priority, east is given 4\textsuperscript{th} priority and resource is given the lowest priority. The reason of giving the lowest priority to resource is to avoid the entrance of new flit if the network is busy.

If there are more than one \( \text{Req}_{\text{jun2pta}} \) signals then each request signal will be handled according to the priority. For example if there is one \( \text{Req}_{\text{jun2pta}}_{-}N \) signal from north and one \( \text{Req}_{\text{jun2pta}}_{-}W \) signal from west then north request will be handled first before west. If there are requests from all five directions then each request will be handled according to the given priority.

The hierarchical FSM diagram for scanning each J&R Control block is shown in Figure 4.16. Each state is responsible for scanning J&R Control block of each direction. The flaw of this design is that it takes too much time to handle more
than one request even only one request can take more time in worst case though. To handle this flaw, an improved design is proposed which will be discussed later in this thesis.

![Hierarchical FSM](image)

**Figure 4.16: Hierarchical FSM**

### 4.4.2.2 Path Table

Path Table block is the second sub block in PTAC block. Main functionality of Path Table block is described below.

- To read address and send data of the specific address to IH block.

Diagram of Path Table is shown in Figure 4.17.

![Table component](image)

**Figure 4.17: Table component**
This block has four inputs and three outputs. Inputs are clock, Rst, Read and Dess_add while outputs are Data_out and Read_over. Dess_add is 8 bits wide while Data_out is 17 bits wide.

The purpose of this block is to read data at the specific address and sends it to relevant Junction and Resource Control block. As soon as there is Read signal from PTA, path information of specified destination address is sent to Junction & Resource Control block. After this, an acknowledge is sent back to the PTA in the form of Read_over signal.

### 4.4.3 Arbiter & Control

Arbiter & Control is the third block of router. Main functionality of Arbiter & Control is described below.

- To handle all requests for output direction.
- To take decision for output direction by decoding the Head flit.
- To lock and unlock the path.
- To check the space in the buffer of next router.
- To send signals for selecting the output direction.
- To send signal to the next router or resource to save data in buffer.

Block diagram of Arbiter & Control is shown in Figure 4.18.

![Figure 4.18: Arbiter & Control](image-url)

This block has 17 inputs and 25 outputs. Inputs are Clock, Rst, Req_mod2arb_n, Req_mod2arb_s, Req_mod2arb_w, Req_mod2arb_e, Req_mod2arb_r, Data_mod2arb_n, Data_mod2arb_s, Data_mod2arb_w, Data_mod2arb_e, Data_mod2arb_r, RTR_n, RTR_s, RTR_w, RTR_e and RTR_r while outputs are Sel_n, Sel_s, Sel_w, Sel_e, Sel_r, Put_n, Put_s, Put_w, Put_e, Put_r,
Get_arb2mod_n, Get_arb2mod_s, Get_arb2mod_w, Get_arb2mod_e, Get_arb2mod_r. Each Data_mod2arb is 4 bits wide while each Sel is 3 bits wide.

This is the main block of the router. In other words it is called the brain of the router. All decisions regarding directions are taken inside this block. Arbiter & Control has five Req_mod2arb signals and five data lines coming from five different IH components. Five Get_arb2mod signals are sent to the IH components for acknowledge. To communicate with the next router or resource, Arbiter & Control has five Put signal going towards four different routers and one resource and five RTR signals coming from four different routers and one resource. To select output port five different selection lines are sent to Cross Bar.

If there is request from Header Modifier block of any direction then the first task of Arbiter & Control is to check the flit type from data send by Header Modifier. Then Arbiter & Control divide it into three different tasks and deals with each flit type differently. Like Head flit, Body flit, End flit or Full flit.

When the flit type is Head flit then Arbiter & Control will check its direction by decoding the direction bits received from Header Modifier. Decoding depends on which direction it is coming from. If data is coming from north then “00” means east, “01” means south, “10” means resource and “11” means west direction. All possible direction decoding is shown in the Figure 4.19.

Since we are using worm hole routing technique, path locking is also the responsibility of this block. It means after taking decision from Head flit, the path will be locked for Body and End flit. When End flit traverses through the path then path is unlocked for the next coming flits. When Full flit traverses through the router, then no need to lock the path.

After decoding, it is checked that the desire direction should be unlock and the buffer of the next router should have space to receive the next flit. It can be checked through RTR signal coming from next router or resource to the Arbiter & Control block which is ‘1’ when there is any space left in buffer and it is ‘0’ if the buffer is full. After checking all these constraints, signals are sent to the Cross Bar through selection lines for particular direction and the desire direction is locked.

For Body or End flit type, there is no need to select direction because path has already been selected by the Head flit. The only thing to check is that buffer of next router or resource should have space before sending the next flit which can be checked through RTR signal. If flit type is Full flit then direction and all other constraints will be checked like Head flit, the only difference is that path will not be locked for Full flit.

After taking path decisions and checking all other constraints, path is lock or unlock and next buffer is full or empty, a Put signal is sent to the relevant router or resource followed by the hand shaking protocol. A Get_arb2mod signal is also sent back to the relevant Header Modifier indicating that data have been sent to the next router or resource and Arbiter & Control is ready to deal with next data.
Like Path Table Arbiter, each request signal is scanned one by one. If there is more than one request then each request will be handled according to the priority. Scan of each request results in more delay. Latency will be increased if any flit needs to access the Path Table. Five flits coming from different directions having different output directions cannot traverse from the router at the same time. This is a big problem in this design. To overcome this problem, an improved design is proposed in which flits having different destination output are handled at the same time. An improved router design is discussed in detail in chapter 5.

Figure 4.19: Direction Decoding
4.4.4 Cross Bar

Cross Bar is the fourth block of router. Main functionality of Cross Bar is described below

- Traverse data from one path to selected path.

Cross Bar has ten inputs and five outputs. Inputs are Data_in_n, Data_in_s, Data_in_w, Data_in_e, Data_in_r, Sel_n, Sel_s, Sel_w, Sel_e, Sel_r and outputs are Data_out_n, Data_out_s, Data_out_w, Data_out_e, Data_out_r.

Cross Bar has five inputs and five outputs for data transfer. Each input and output is 34 bits wide. There are also five input selection lines. Each line is 3 bits wide. Since there are five input data lines, three bits are enough to take decision which input data line shall be connected to selected output data line. Five input selection lines are used to take decision for five outputs data line. All decisions are taken inside the Arbiter & Control and selection signals are sent to the Cross Bar.

For example, if “001” is sent on the north selection lines it means data line from north is connected to the data line to north direction. If “010” is sent on the north selection lines it means data line from south is connected the data line to north direction. If “011” is sent on the north selection lines it means data line from west is connected the data line to north direction and so on. Similarly if three bits signals are sent to the south selection line then incoming data line will be connected to the south data line depending on the combination choose for incoming data line. Combination for incoming data lines are given in the following Table 4.3.

<table>
<thead>
<tr>
<th>Code</th>
<th>Incoming Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>North</td>
</tr>
<tr>
<td>010</td>
<td>South</td>
</tr>
<tr>
<td>011</td>
<td>West</td>
</tr>
<tr>
<td>100</td>
<td>East</td>
</tr>
<tr>
<td>101</td>
<td>Resource</td>
</tr>
<tr>
<td>Others</td>
<td>No connection</td>
</tr>
</tbody>
</table>

Table 4.3: Direction codes

Figure 4.20 shows the block diagram of Cross Bar.
After designing each component separately, they are integrated to get a complete router design. Detailed design is shown in Figure 4.21.
5 VHDL Modeling, Simulation and Results

The Router has been designed using hardware description language named as VHDL. VHDL stands for Very high speed integrated circuit Hardware Description Language which is used for the modeling of digital systems at structural, RTL or behavioral level. We have divided the router into different components (blocks) as described earlier. These components are designed in VHDL and all designed components are integrated in the end. The reason for dividing the design into components is that it is easy to design small components and due to the concurrent nature of VHDL, each component works independently from others.

Router design is simulated by creating a dummy environment around its surroundings. After designing the router in VHDL, a test bench is written for testing the design. Test bench works as a dummy core that is sending different flits to the router at different rate. At one end of the router, dummy core is connected which sends the flits in different direction of the router. At the second end of the router, another router is cascaded that works as a dummy core or dummy router to receive the flits from the router.

![Router Delay Diagram](image)

Figure 5.1: Router delay from input port to output port

Coding and simulation of the router design has been implemented by using Mentor Graphics as a VHDL tool. For this purpose ModelSim-Altera 6.5b has been used. The purpose of the simulation is to determine the router delay in terms of clock cycles. There are various cases of testing router delay.

**Test Cases**

1. **With zero loads:** Only one input port receives flits from core or router. There are two categories.
• When Junction Bit (JB) is ‘1’.
• When Junction Bit (JB) is ‘0’.

2. With load (priority based): More than one input port receive different flits from core or routers. There are again two cases.
   a. Without output port conflict. It can be categories into two parts.
      • When Junction Bit (JB) is ‘1’.
      • When Junction Bit (JB) is ‘0’.
   b. With output port conflict. It can be categories into two parts.
      • When Junction Bit (JB) is ‘1’.
      • When Junction Bit (JB) is ‘0’.

There are lots of cases to test the design. It is very difficult to test each and every case due to limitation of short available time. So we have taken some important cases for the simulation of our design. Since router scans each input port one by one therefore flits are injected at different rates to check the worst case of latency.

5.1 Test Results

5.1.1 Testing with Zero load

The first test is performed by sending flits to only one input port of any direction. Suppose if flits are sent to north input port then there will be no flit at any other Input Port. We can see from Table 5.1 that clock cycle’s column is divided into three iterations and their average is calculated in the last column. Reason of three iterations is that “Arbiter & Control” block and “Path Table Arbiter” block scans the input port request from all directions one by one according to the given priority. Therefore number of clock cycles depends on the rate at which flits are inserted into the router.

Simulation results in the form of clock cycles are shown in the Table 5.1.

From the table it is noted that when the Junction Bit (JB) in the Head flit or Full flit is ‘0’ then the maximum (Max) number of clock cycles taken by the flit is 12 and minimum (Min) is 8. In case of Body or End flit, they take almost same number of clock cycles as the Head flit takes. When the Junction Bit (JB) in the Head flit or Full flit is ‘1’ then the Max clock cycles taken by the flit is 20 and Min is 14. In case of Body or End flit, they take almost the same number of clock cycles as the Head flit takes when the Junction Bit (JB) is ‘0’. It is because Head flit has to take the new path information from the Path Table first and then path is locked after passing the Head flit from the router. Body or End flit just follows the path locked by the Head flit.
<table>
<thead>
<tr>
<th>Direction</th>
<th>Flit Type</th>
<th>JB</th>
<th>Number of clock cycles</th>
<th>Average clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Iteration1</td>
<td>Iteration2</td>
</tr>
<tr>
<td>N→E</td>
<td>Head</td>
<td>0</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>S→R</td>
<td>Head</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>R→E</td>
<td>Head</td>
<td>0</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>W→S</td>
<td>Full</td>
<td>0</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>W→N</td>
<td>Full</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>R→S</td>
<td>Head</td>
<td>1</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>E→N</td>
<td>Full</td>
<td>1</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>E→W</td>
<td>Full</td>
<td>1</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>R→S</td>
<td>Full</td>
<td>1</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>R→E</td>
<td>Full</td>
<td>1</td>
<td>18</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 5.1: Simulation results (without load)

5.1.2 Testing with load and no output port conflict

This test is performed by sending flits to more than one Input Port but different flits access different output ports. Suppose if one flit is sent to north input port
and another flit is sent to south input port then there will be no chance of getting same output port by these flits. In another way both have different destination output ports.

Simulation results in the form of clock cycles are shown in the Table 5.2 below.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Flit Type</th>
<th>JB</th>
<th>Number of clock cycles</th>
<th>Average clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Iteration1</td>
<td>Iteration2</td>
</tr>
<tr>
<td>N→E</td>
<td>Head</td>
<td>0</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td>0</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td>0</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>S→N</td>
<td>Head</td>
<td>0</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td>0</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td>0</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>S→R</td>
<td>Full</td>
<td>0</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>E→N</td>
<td>Full</td>
<td>0</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>1</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>N→S</td>
<td>Full</td>
<td>1</td>
<td>24</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 5.2: Simulation results (without output port conflict)

From the table it is noted that when the Junction Bit (JB) in the Head flit or Full flit is ‘0’ then the Max clock cycles taken by the flit is 20 and Min is 10. In case of Body or End flit, they take almost same number of clock cycles as the Head flit takes. When the Junction Bit (JB) in the Head flit or Full flit is ‘1’ then the Max clock cycles taken by the flit is 28 and Min is 18. In case of Body or End flit, they take almost same number of clock cycles as the Head flit takes when the Junction Bit (JB) is ‘0’. It is because Head flit has to take the new path information from the Path Table first and then path is locked after passing the Head flit from the router. Body or End flit just follows the path locked by the Head flit. If all input ports receive Head or Full flit, with Junction Bit (JB) equal to ‘1’, then number of clock cycles increases for low priority flits.
5.1.3 Testing with output port conflict

This test is performed by sending flits to more than one input port of any direction and more than one flit access the same output port as destination. Suppose two or more than two flits entering input ports north and south have the same destination port east. This means they have the same destination output port.

Simulation results in the form of clock cycles are shown in the Table 5.3 below.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Flit Type</th>
<th>JB</th>
<th>Number of clock cycles</th>
<th>Average clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Iteration1</td>
<td>Iteration2</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>0</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>N→R</td>
<td>Full</td>
<td>0</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>N→R</td>
<td>Full</td>
<td>1</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>1</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>N→E</td>
<td>Head</td>
<td>0</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>N→E</td>
<td>Body</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>N→E</td>
<td>End</td>
<td>0</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>S→E</td>
<td>Head</td>
<td>40</td>
<td>42</td>
<td>38</td>
</tr>
<tr>
<td>S→E</td>
<td>Body</td>
<td>50</td>
<td>52</td>
<td>48</td>
</tr>
<tr>
<td>S→E</td>
<td>End</td>
<td>60</td>
<td>62</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 5.3: Simulation results (with load and output port conflict)

From the table it is noted that when the Junction Bit (JB) in the Full flit is ‘0’ and more than one Full flit coming from different input port to the same output port then high priority input port will get the output port. So number of clock cycles taken by the high priority flit is less than the other. The Max clock cycles taken by the flit are 20 and Min is 14. In case of Head Body or End flit, the low priority flits have to wait until high priority Head, Body and End flits pass through the router and unlock the path for other flits. If number of Body flits increased then time taken by the low priority flits will also increase.
The same situation will happen when the Junction Bit (JB) in Head flit or Full flit is ‘1’. The only difference is that due to change of path information in Head flit or Full flit it will take more time as compared to normal routing.

5.2 Improved Design

Arbiter & Control block and Path Table Arbiter block take time to scan each request from different directions. These blocks scan the requests one by one based on priorities given to each request from different directions. Like, north has the highest priority, 2nd priority is given to south, 3rd priority is given to west, 4th priority is given to east and 5th priority is given to resource. There are two flaws in this design. Firstly, if there is request from low priority Input Port then it has to wait until process scans all high priority requests. Secondly, if there is request from more than one Input Port then all low priority requests have to wait until high priority request finish its task of sending data to output port.

If the flits are not sent to the same output port then there is a possibility to send flits at the same time. For example, if flit coming from north wants to go to south, south wants to west, west wants to east, east want to resource and resource wants to send flit to the north direction then it is possible to send all of them at the same time rather than wasting time by sending one flit at a time. Although in router architecture there is one common Path Table which can handle one request at one time but still time can be saved by checking all the requests at the same time. If there are more than one request for the same resource (like Path Table or destination direction) then we have to assign priorities to avoid congestion problem so that only one request can access common resource.

To overcome this flaw a new design is proposed to handle different requests. Using the idea of iterative circuits we can handle all requests at the same time. Basic idea of using iterative circuit is to reduce design complexity. In real life combinational circuits have large number of inputs and outputs. Large number of input makes the circuit design more complex. In iterative circuit we divide the function into sub functions. Every sub function has less number of inputs which make it easy to design a circuit. In iterative circuit, all sub functions are replica of each other. Due to using the same copy of sub function, circuit is named as Iterative Circuit. There is a tradeoff between speed and cost in Iterative Circuit. Using iterative circuit increases the area of the design, results in more costly. On the other hand it increases the speed of the design.

In Arbiter & Control there are 35 inputs coming from different blocks and some internal feedback. With this high number of inputs, design complexity increases and having low speed. Using the idea of Iterative Circuit, Arbiter & Control block is divided into sub functions. Each sub function has seven inputs.

Path Table Arbiter block is also based on the same design as described before. The only difference is that block has different kinds of inputs and outputs and only one request can access Path Table. Still latency can be reduced by checking all requests at the same time. The advantage of using this design is that when there is
no high priority request then low priority request can be handled as soon as possible. There is no need to wait for scanning of higher request.

An Iterative circuit design for Arbiter & Control is shown in Figure 5.2. If the lock_in signal for any Arbiter & Control block is ‘0’ then any low priority request can use the output port.

![Iterative circuit design](image)

**Figure 5.2**: Iterative circuit implementation in Arbiter & Control

### 5.3 Results

Improved design has been tested in the same way as the first design as was described in the previous section. Table has been divided into three categories as described below.

#### 5.3.1 Testing with Zero load

The first test is performed by sending flits to only one input port of any direction. Simulation results in the form of clock cycles are shown in the Table 5.4 below.

From the table it is noted that when the Junction Bit (JB) in the Head flit or Full flit is ‘0’ then the average clock cycles taken by the flit is 2. In case of Body or End flit, they also take 2 clock cycles as the Head flit takes. When the Junction Bit (JB) in the Head flit or Full flit is ‘1’ then the average clock cycles taken by the flit is 4. It is because Head flit has to take the new path information from the Path Table first and then path is locked after passing the Head flit from the router. In case of Body or End flit, they take same number of clock cycles as the Body or End flit takes when the Junction Bit (JB) is ‘0’. Body or End flit just follows the path locked by the Head flit.
### VHDL Modeling, Simulation and Results

#### Without load (zero load)

<table>
<thead>
<tr>
<th>Direction</th>
<th>Flit Type</th>
<th>JB</th>
<th>Number of clock cycles</th>
<th>Average clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Iteration1</td>
<td>Iteration2</td>
</tr>
<tr>
<td>N→S</td>
<td>Head</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S→R</td>
<td>Head</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>R→E</td>
<td>Head</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>W→S</td>
<td>Full</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>W→N</td>
<td>Full</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>E→N</td>
<td>Full</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>E→W</td>
<td>Full</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>R→S</td>
<td>Full</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>R→E</td>
<td>Full</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.4: Improved results (without load)

#### 5.3.2 Testing with no output port conflict

This test is performed by sending flits to more than one input port of any direction but no flit want to access the same output port. Suppose if one flit is sent to west input port and another flit is sent to east input port then there will be no chance of getting same output port by these flits. In another way both have different destination output port.

Simulation results in the form of clock cycles are shown in the Table 5.5 below.
### Table 5.5: Improved results (with load and no output port conflict)

<table>
<thead>
<tr>
<th>Direction</th>
<th>Flit Type</th>
<th>JB</th>
<th>Number of clock cycles</th>
<th>Average clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Iteration1</td>
<td>Iteration2</td>
</tr>
<tr>
<td>N→E</td>
<td>Head</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S→N</td>
<td>Head</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Body</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S→R</td>
<td>Full</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>E→N</td>
<td>Full</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>N→W</td>
<td>Full</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>N→S</td>
<td>Full</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>S→W</td>
<td>Full</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>W→E</td>
<td>Full</td>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>R→N</td>
<td>Full</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

From the table it is noted that when the Junction Bit (JB) in the Head flit or Full flit is ‘0’ then the average clock cycles taken by the flit is 2. In case of Body or End flit, they take same number of clock cycles as the Head flit takes. When the Junction Bit (JB) in the Head flit or Full flit is ‘1’ then the average clock cycles taken by the high priority flit is 4 and low priority flit is 5. Number of clock cycles will increase by one for each low priority flit because only one flit can access the Path Table at a time. It is because Head flit has to take the new path information from the Path Table first and then path is locked after passing the Head flit from the router. In case of Body or End flit, they take same number of clock cycles as
the Body or End flit takes when the Junction Bit (JB) is ‘0’. Body or End flit just follows the path locked by the Head flit.

5.3.3 Testing with output port conflict

This test is performed by sending flits to more than one input port of any direction and flits from different input ports want to access the same output port. Suppose two or more than two flits entering Input Port resource and Input Port east have the same destination port north. In another way both have same destination output port.

Simulation results in the form of clock cycles are shown in the Table 5.6 below.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Flit Type</th>
<th>JB</th>
<th>Number of clock cycles</th>
<th>Average clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Iteration 1</td>
<td>Iteration 2</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>N→R</td>
<td>Full</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>N→W</td>
<td>Full</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S→W</td>
<td>Full</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>E→W</td>
<td>Full</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>R→W</td>
<td>Full</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>E→R</td>
<td>Full</td>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>N→R</td>
<td>Full</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>N→S</td>
<td>Full</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>W→S</td>
<td>Full</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>E→S</td>
<td>Full</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>R→S</td>
<td>Full</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5.6: Improved results (with load and output port conflict)
From the table it is noted that when the Junction Bit (JB) in the Full flit is ‘0’ and more than one Full flit coming from different input port to the same output port then high priority input port will get the output port. So number of clock cycles taken by the high priority flit is less than the other. The average clock cycles taken by the high priority flit is 2 while for low priority flit is 4. Number of clock cycles will increase by 2 for each low priority flit. In case of Head, Body or End flit, the low priority flits have to wait until high priority Head, Body and End flits pass through the router and unlock the path for other flits. If number of Body flits increased then time taken by the low priority flits will also increase.

![Timing diagram of flit traversing from the router](image)

**Figure 5.3:** Simulation results when JB or RB is ‘0’

The same situation will occur when the Junction Bit (JB) in Head flit or Full flit is ‘1’. The only difference is that due to change of path information in Head flit or Full flit it will take more time as compared to normal routing.

Timing diagram of flit traversing from the router is shown in Figure 5.3 when Junction Bit (JB) or Resource Bit (RB) is ‘0’. In Figure 5.3, there are five different flits at different direction of the router and each flit destination is different from
other. In other way there is no output port conflict between different flits. All Head flits are received at the output port at the same time. We can see clearly from the diagram that each flit takes two clock cycles from input port to output port. One clock cycle is used to save the flit into buffer and one clock is used to take decision of output port.

Figure 5.4: Simulation results when JB or RB is ‘1’

Timing diagram of flit traversing from the router is shown in Figure 5.4 when Junction Bit (JB) or Resource Bit (RB) is ‘1’. In this figure there are five different flits at different direction of the router and each flit destination is different from other. In this case there is no output port conflict between different flits. All Head flits are received at the output port at different time. The reason is that every Head flit has to get path information from Path Table before moving to the output port. Only one flit can access Path Table at a time so each flit will get path information
according to the given priority. Low priority flit will take the highest time to go to
the output port. We can see clearly from the diagram that first flit takes four clock
cycles from input port to output port. One clock cycle is used to save the flit into
buffer, two clock cycles are used to get path information and one clock is used to
take decision of output port. Second flit will take five clock cycles. One clock
cycle is used to save the flit into buffer, three clock cycles are used to get path
information because of high priority flit and one clock is used to take decision of
output port. Third flit will take six clock cycles and so on.
6 FPGA Prototyping

This chapter deals with the router prototyping on Altera FPGA board. To test the working of the router, it is connected with the Nios II and flits are generated through software. PC is connected with Altera FPGA board through JTAG UART (Joint Test Access Group Universal Asynchronous Receiver Transmitter). Quartus II is used to compile the design and program the FPGA chip. NIOS II IDE (Integrated Development Environment) is used to program the memory and observe the output of the design.

Communication between Router and Nios II CPU through Avalon bus and sending different kind of flits (Head, Body, End, Full flit) to the router through PIO is shown in the Figure 6.1.

![Figure 6.1: Hardware architecture of the system](image)

Hardware architecture has been designed using SOPC (System on Programmable Chip) Builder. SOPC Builder is the part of Quartus II tool. SOPC Builder has some predefined components. These components are called Intellectual Property (IP) because of their copyright status which is protected through a patent [13]. These IP components for FPGA systems include CPUs, PCI interfaces, PIOs, protocols, graphic processors etc. SOPC library also includes some of these IP components. One of these components is NIOS II CPU which has been used in this project. Some of other components used in the prototype are JTAG UART, On-chip RAM, SRAM (External) and PIOs. An internal clock also known as system clock of 50 MHz is used to run the system.
Figure 6.2: Quartus II Architecture
SOPC Builder software is used to connect these components and generates a complete system. This system includes the following files.

- Verilog HDL or VHDL files of all components.
- A Block Symbol File (.bsf) for the system.
- A project file (.ptf) for the SOPC Builder.

This System Block with a name of NiosII is added into the Qartus II project as shown in the Figure 6.2.

Router design files (VHDL files) are imported into Quartus II project and a block symbol file (.bsf) is created by using some options. This router block symbol file is then added into the same file with the NiosII block file. Pins of both blocks are connected through wires. Connection between NiosII and router is shown in Figure 6.2. Both NiosII and router are connected with 50 MHz on board system clock. In Figure 6.2, NiosII is dealing with only one direction of the router because in NoC one resource is connected to only one router but the testing has been performed with all direction of the router. Quartus architecture of complete testing design is shown in Appendix I in Figure 9.1.

A flow chart of complete design process and its prototyping is shown in the Figure 6.3.

### 6.1 Prototype Result

To test the router functionality on FPGA board, different flits are sent to different directions. Flits are sent by the NiosII towards router and the output of the router is given back to the same NiosII CPU. Since router has buffers to store flits therefore first the flits are sent to the router by NiosII is temporarily stored in buffers of router. When NiosII is ready to receive flits by giving some handshaking signals to the router then router sends these flits towards their destination directions. In our design decision, we have fixed flit size of 34 bits but due to PIO limitations we are using flit size of 32 bits in the prototype. PIO is IP component included in SOPC Builder which is using total of 32 bits for data transfer. Since we are connecting router to NiosII through PIO therefore for testing purpose we have changed flit size to 32 bits.

As described before in previous chapter that there are a lot of cases to test a router completely. This is not possible to do within a short period of time. During the router prototyping, it is also difficult to create a big network on Altera FPGA chip due to a limited number of logic elements (LEs). Therefore, we used only one resource to connect to router to test the functionality. Different kind of flits are sent on different directions of the router e.g. at one direction full flit is sent with JB is ‘0’, one full flit is sent with Junction Bit (JB) is ‘1’. At one direction Head, Body and End flits are sent. For Body flit data is taken from the keyboard. After sending different kind of flits, NiosII CPU is started to receive flits from the router. C-code for sending and receiving flits is given in Appendix I.
Figure 6.3: Flow chart diagram of the design and test process
Compilation report shows that the max frequency which can be used for the system clock is 53.7 MHz because buffers are implemented using on chip memory which is slower than the registers. Resource utilization taken by each of the component of the design and taken by the complete design is shown in Table 6.1.

<table>
<thead>
<tr>
<th>Components</th>
<th>Total Logic Elements</th>
<th>Total comb. functions</th>
<th>Total Registers</th>
<th>Total Memory Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input port</td>
<td>58(&lt;1%)</td>
<td>55(&lt;1%)</td>
<td>51(&lt;1%)</td>
<td>272(&lt;1%)</td>
</tr>
<tr>
<td>Header Modifier</td>
<td>99(&lt;1%)</td>
<td>99(&lt;1%)</td>
<td>40(&lt;1%)</td>
<td>0</td>
</tr>
<tr>
<td>Junction &amp; Resource Control</td>
<td>97(&lt;1%)</td>
<td>97(&lt;1%)</td>
<td>38(&lt;1%)</td>
<td>0</td>
</tr>
<tr>
<td>Path Table Arbiter</td>
<td>104(&lt;1%)</td>
<td>104(&lt;1%)</td>
<td>46(&lt;1%)</td>
<td>0</td>
</tr>
<tr>
<td>Path Table</td>
<td>2(&lt;1%)</td>
<td>1(&lt;1%)</td>
<td>1(&lt;1%)</td>
<td>4352(&lt;1%)</td>
</tr>
<tr>
<td>Cross Bar</td>
<td>703(&lt;2%)</td>
<td>703(&lt;2%)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arbiter &amp; Control</td>
<td>310(&lt;1%)</td>
<td>309(&lt;1%)</td>
<td>61(&lt;1%)</td>
<td>0</td>
</tr>
<tr>
<td>IPC</td>
<td>228(&lt;1%)</td>
<td>224(&lt;1%)</td>
<td>129(&lt;1%)</td>
<td>272(&lt;1%)</td>
</tr>
<tr>
<td>PTAC</td>
<td>109(&lt;1%)</td>
<td>106(&lt;1%)</td>
<td>47(&lt;1%)</td>
<td>4352(&lt;1%)</td>
</tr>
<tr>
<td><strong>Router</strong></td>
<td><strong>2259(7%)</strong></td>
<td><strong>2244(7%)</strong></td>
<td><strong>754(2%)</strong></td>
<td><strong>5712(1%)</strong></td>
</tr>
<tr>
<td>NIOS II</td>
<td>2485(7%)</td>
<td>2048(6%)</td>
<td>1641(5%)</td>
<td>9476(2%)</td>
</tr>
<tr>
<td>Router + NIOSII</td>
<td>4608(14%)</td>
<td>4314(13%)</td>
<td>2386(7%)</td>
<td>15108(3%)</td>
</tr>
</tbody>
</table>

Table 6.1: FPGA implementation results of JBR Router
Discussion about Results

Prototype test has been performed on Altera cyclone II FPGA (EP2C35) chip having 33,216 logic elements (Leds). Cyclone II family belongs to low cost FPGAs which are easily available in universities. From Table 6.1 it is clear that Router takes 7% of the FPGA chip while Nios II also takes 7% of the total chip. Totally seven (7) routers and resources like Nios II can be integrated on one FPGA chip. It means a network of 2x2 or 3x2 is possible to integrate on FPGA chip. The maximum capacity in cyclone II family (EP2C70) is 68,416 LEs. A network of 4x3 or 3x3 can be integrated on that FPGA chip. Altera also have FPGA family whose chip capacity is more than cyclone II. The following table gives capacity of different Altera families.

<table>
<thead>
<tr>
<th>Low cost FPGA</th>
<th>Midrange FPGA</th>
<th>High End FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone V(SCEA9)</td>
<td>Arria V(5AGXB7)</td>
<td>Stratix V(5SGXB6)</td>
</tr>
<tr>
<td>301,000 (LEs)</td>
<td>503,500</td>
<td>597,000</td>
</tr>
</tbody>
</table>

Table 6.2: A Comparison between different Altera families

It is clear from the table that the maximum capacity of the FPGA is for the Stratix family which has total of 597,000 LEs. It means an 11x11 NoC can be integrated on a single FPGA chip. Network capacity like 16x16 can be achieved by connecting multiple FPGA chips on a single board.
Conclusions

7 Conclusions

Source routing is a suitable routing technique in NoC and has many advantages over distributed routing. But for large networks it becomes inefficient in term of overhead in packet header. Junction Based routing (JBR) is a new efficient technique for large networks because it reduces overhead in packet header of source routing. This thesis was to develop a router to support JBR and compare it with normal NoC router for the latency and cost. A prototype of the design has been implemented in FPGA to find the resource requirement of our design.

7.1 Summary of Contribution

To implement this new routing technique, a Path Table is needed to store the path information for Junction Based Source routing in junction routers. There is only a fixed length path information in Head flit to route towards destination. If destination is far from the source then it has to change its path information from Junction router.

7.1.1 Path Table Options

There are various options of Path Table organization. First option is to store Path Table in router for each Input Port and one Path Table in RNI. Second option is to store common Path Table for all Input Ports in router and one Path Table in RNI. Third option is to store common Path Table in RNI of connected core for all Input Ports and core and final option is to store one Path Table in router for all Input Ports and core. After comparing all options it is concluded that to store one Path Table in router for all Input Ports and the core is most cost efficient. This design decision has been followed throughout the thesis.

7.1.2 Router Design, Modeling and Prototyping

To simplify the design, first router was divided into small components like Input Port, Header Modifier, Junction & Resource Control, Path Table Arbiter, Cross Bar and Arbiter & Control. Then signals are defined for each component and in the end all components are integrated to make a complete router design.

All components of Router were designed by using hardware description language VHDL and then each component was simulated separately in ModelSim. In the end, all components were integrated to make the JBR router and simulated it to find the latency of the router.

Through simulation based analysis we found that the first JBR router design was very slow with a large packet latency. An improved design was developed and implemented to speed up router and reduce router latency. The idea of iterative circuit was used to deal with all input ports in parallel to improve the speed of the router. Simulation results show that in case of normal router, each flit takes 2 clock cycles to traverse from the router, while in case of Junction router, Head or Full flit will take 4 clock cycles if it needs to access the Path Table. Otherwise all type of flits take 2 clock cycle to traverse from the router.
Conclusions

Finally, design has been prototyped on Altera FPGA board to check the resource utilization and operating clock speed.

7.2 Limitation and Future Work

There are some limitations of our current work and there exists many opportunities for future work to extend this thesis.

7.2.1 Path Table Compression

One of the main problems in router is to store common Path Table for all Input Ports and the core. Area of the Path Table increases with increase in the network size. Huge memory is needed to store such a big Path Table. Path Table compression is an issue which would be an interesting research for the future.

7.2.2 Virtual Channel in Router

One of the problems in router is that a lot of flits can be blocked in buffer. This is because the Head flit is waiting for an output port which is occupied by some other flit. It would be a good idea to introduce virtual channels which can separate flits according to the different destinations. This will be helpful for the decreasing packet latency.

7.2.3 Enhanced Pipelining Technique

Normal router and Junction router take more than one clock cycle to traverse a flit. Junction router has high latency as compared to the Normal router. It would be very interesting and challenging task to improve the design so that the latency is equal to one clock cycle. Perhaps enhanced pipelining techniques can be used for this purpose.

7.2.4 Multiple FPGA Board for Prototyping

In this thesis design has been prototyped using only one single FPGA chip. Since available FPGAs have limited number of resources, it is not possible to prototype a big network on such a small chip. It would be a good idea to integrate multiple FPGA on a single board to implement a bigger NoC.
 References

8 References


9 Appendix I

Testing of Router Prototype:

We have connected the NiosII Core with router in such a way that NiosII is connected with five different Input Ports of the router to send data as well as handshaking signals. Same Nios II CPU is connected to the output port of router to receive data by using some handshaking signals. The purpose is to send different kinds of flit types at different Input Ports and receive them from the other end of the router. Since Altera FPGA board is attached to the PC through JTAG UART therefore it is easy to print the output of the router on the console window of Nios II IDE.

It is difficult to describe all cases for router testing but only a few test combinations have been applied in this appendix to check the working of the router. At North Input Port two Full flits are sent. One of the flits has the JB equal to ‘1’ and other has JB equal to ‘0’. One full flit with RB equal to ‘1’ has been sent to the Resource Input Port. Please see section 4.2 for flit format. One Full flit with JB or RB equal to ‘0’ is sent to the East input of the router. The Head, Body and End flit is sent to North direction in such a way that after sending Head flit, Nios II needs three input character from the keyboard. Nios II makes three different Body flits of receive characters and sends them to the router. After that End flit is sent to the router.

All flits are received at South output port because decision is taken by looking in path information bits of all Head or Full flits. Nios II receives all flits from south direction depending on the priority of the flit. Quartus architecture for router prototyping is shown in Figure 9.1.

C-Code for router test:

```c
#include<io.h>
#include<system.h>
#include<stdio.h>
#include<stddef.h>

int main(void)
{
    int rtr_n_out=0, put_n_out, rtr_s_out=0, put_s_out, rtr_w_out=0,
    put_w_out=0, rtr_e_out=0, put_e_out, rtr_r_out=0, put_r_out;
    int data_out, data_n_out, data_s_out, data_w_out, data_e_out, data_r_out;
    const int wr_1=1;
    const int wr_0=0;
    char c[20];
    int i;
    long int a=0x40000000, b=0;
    printf("let's start\n");
    IOWR(RTR_S_IN_BASE,0,wr_1);

    //****1st flit: full flit when RB is 1****/
    IOWR(DATA_R_IN_BASE,0,0xE0000010);
    IOWR(put_R_IN_BASE,0,wr_1);
    rtr_r_out=IORD(RTR_R_OUT_BASE,0);
```
while(rtr_r_out!=0)
{
  rtr_r_out=IORD(RTR_R_OUT_BASE,0);
}
printf("rtr_r_out=%d\n",rtr_r_out);//should be 0
IOWR(PUT_R_IN_BASE,0,wr_0);
while(rtr_r_out!=1)
{
  rtr_r_out=IORD(RTR_R_OUT_BASE,0);
}
printf("rtr_r_out=%d\n",rtr_r_out);//should be 1

/***2nd flit: full flit when JB is 1*****/
IOWR(DATA_N_IN_BASE,0,0xD0000020);
IOWR(PUT_N_IN_BASE,0,wr_1);
rtr_n_out=IORD(RTR_N_OUT_BASE,0);
while(rtr_n_out!=0)
{
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}
printf("rtr_n_out=%d\n",rtr_n_out);//should be 0
IOWR(PUT_N_IN_BASE,0,wr_0);
rtr_n_out=IORD(RTR_N_OUT_BASE,0);
while(rtr_n_out!=1)
{
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}
printf("rtr_n_out=%d\n",rtr_n_out);//should be 1

/***3rd flit: full flit when JB and RB both are 0*****/
IOWR(DATA_E_IN_BASE,0,0xC0000000);
IOWR(PUT_E_IN_BASE,0,wr_1);
rtr_e_out=IORD(RTR_E_OUT_BASE,0);
while(rtr_e_out!=0)
{
  rtr_e_out=IORD(RTR_E_OUT_BASE,0);
}
printf("rtr_e_out=%d\n",rtr_e_out);//should be 0
IOWR(PUT_E_IN_BASE,0,wr_0);
rtr_e_out=IORD(RTR_E_OUT_BASE,0);
while(rtr_e_out!=1)
{
  rtr_e_out=IORD(RTR_E_OUT_BASE,0);
}
printf("rtr_e_out=%d\n",rtr_e_out);//should be 1

/***4th flit: full flit when JB 1*****/
IOWR(DATA_W_IN_BASE,0,0xD0000030);
IOWR(PUT_W_IN_BASE,0,wr_1);
rtr_w_out=IORD(RTR_W_OUT_BASE,0);
while(rtr_w_out!=0)
{ 
  rtr_w_out=IORD(RTR_W_OUT_BASE,0);
}
printf("rtr_w_out=%d
",rtr_w_out);//should be 0
IOWR(PUT_W_IN_BASE,0,wr_0);
while(rtr_w_out!=1)
{
  rtr_w_out=IORD(RTR_W_OUT_BASE,0);
}
printf("rtr_w_out=%d
",rtr_w_out); //should be 1

IOWR(DATA_N_IN_BASE,0,0x01000000);
IOWR(PUT_N_IN_BASE,0,wr_1);
rtr_n_out=IORD(RTR_N_OUT_BASE,0);
while(rtr_n_out!=0)
{
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}
printf("rtr_n_out=%d
",rtr_n_out);
IOWR(PUT_N_IN_BASE,0,wr_1);
rtr_n_out=IORD(RTR_N_OUT_BASE,0);
while(rtr_n_out!=0)
{
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}
printf("rtr_n_out=%d
",rtr_n_out);
while(rtr_n_out!=1)
{
  IOWR(PUT_N_IN_BASE,0,wr_0);
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}
while(rtr_n_out!=0)
{
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}
printf("rtr_n_out=%d
",rtr_n_out); //should be 0
IOWR(PUT_N_IN_BASE,0,wr_0);
rtr_n_out=IORD(RTR_N_OUT_BASE,0);
while(rtr_n_out!=1)
{
  IOWR(PUT_N_IN_BASE,0,wr_0);
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
}

/*---------------------------*/
for(i=0;c[i]!=13;i++)
{
  b=a+c[i];
  printf("a=%x
 ",b);
  IOWR(DATA_N_IN_BASE,0,b);
  IOWR(PUT_N_IN_BASE,0,wr_1);
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
  while(rtr_n_out!=0)
  {
    IOWR(PUT_N_IN_BASE,0,wr_1);
    rtr_n_out=IORD(RTR_N_OUT_BASE,0);
  }
  printf("rtr_n_out=%d
 ",rtr_n_out);//should be 0
  IOWR(PUT_N_IN_BASE,0,wr_0);
  rtr_n_out=IORD(RTR_N_OUT_BASE,0);
  while(rtr_n_out!=1)
  {
    IOWR(PUT_N_IN_BASE,0,wr_0);
    rtr_n_out=IORD(RTR_N_OUT_BASE,0);
  }
  printf("rtr_n_out=%d
 ",rtr_n_out);
}*/
Appendices

```c
printf("rtr_n_out=%d \n", rtr_n_out); // should be 1

/***9th flit: End flit****/
IOWR(DATA_N_IN_BASE, 0, 0x80000000);
IOWR(PUT_N_IN_BASE, 0, wr_1);
rtr_n_out = IORD(RTR_N_OUT_BASE, 0);

while (rtr_n_out != 0)
{
    rtr_n_out = IORD(RTR_N_OUT_BASE, 0);
}
printf("rtr_n_out=%d \n", rtr_n_out);
IOWR(PUT_N_IN_BASE, 0, wr_0);
rtr_n_out = IORD(RTR_N_OUT_BASE, 0);

while (rtr_n_out != 1)
{
    rtr_n_out = IORD(RTR_N_OUT_BASE, 0);
}
printf("rtr_n_out=%d \n", rtr_n_out);

/***reading flits****/

printf("Start reading data\n");
while (1)
{
    IOWR(RTR_S_IN_BASE, 0, wr_1);
    put_s_out = IORD(PUT_S_OUT_BASE, 0);
    data_s_out = IORD(DATA_S_OUT_BASE, 0);
    // printf("data_out=%d \n", data_out);

    while (put_s_out != 1)
    {
        IOWR(RTR_S_IN_BASE, 0, wr_1);
        data_s_out = IORD(DATA_S_OUT_BASE, 0);
        put_s_out = IORD(PUT_S_OUT_BASE, 0);
    }
    printf("PUT_s_out=%d\ndata_s_out=%0x\n", put_s_out, data_s_out);
    IOWR(RTR_S_IN_BASE, 0, wr_0);
    put_s_out = IORD(PUT_S_OUT_BASE, 0);

    while (put_s_out != 0)
    {
        IOWR(RTR_S_IN_BASE, 0, wr_0);
        put_s_out = IORD(PUT_S_OUT_BASE, 0);
    }
    IOWR(RTR_S_IN_BASE, 0, wr_1);
}
return 0;
```
Nios II complete connection to input and output of the router through PIO:

Figure 9.1: Quartus II Architecture for router prototype