DESIGN AND PROTOTYPE OF RESOURCE NETWORK INTERFACES FOR NETWORK ON CHIP

Adnan Mahmood
Zaheer Ahmed Mohammed

Master of Science Thesis 2009
ELECTRONICS
DESIGN AND PROTOTYPE OF RESOURCE NETWORK INTERFACES FOR NETWORK ON CHIP

Adnan Mahmood
Zaheer Ahmed Mohammed

This thesis work is performed at Jönköping Institute of Technology within the subject area Electronics. The work is part of the university's two-year master's engineering degree. The authors are responsible for the given opinions, conclusions and results.

Supervisors: Rickard Holsmark and Shashi Kumar
Examiner: Shashi Kumar
Credit points: 30 points (D-level)
Date:

Archive number:

Postal Address: Box 1026
Visiting Address: Gjuterigatan 5
Telephone: 036-10 10 00
551 11 Jönköping
Network on Chip (NoC) has emerged as a competitive and efficient communication infrastructure for the core based design of System on Chip. Resource (core), router and interface between router and core are the three main parts of a NoC. Each core communicates with the network through the interface, also called Resource Network Interface (RNI). One approach to speed up the design at NoC based systems is to develop standardized RNI. Design of RNI depends to some extent on the type of routing technique used in NoC. Control of route decision base the categorization of source and distributed routing algorithms. In source routing a complete path to the destination is provided in the packet header at the source, whereas in distributed routing, the path is dynamically computed in routers as the packet moves through the network. Buffering, flitization, deflitzation and transfer of data from core to router and vice versa, are common responsibilities of RNI in both types of routing. In source routing, RNI has an extra functionality of storing complete paths to all destinations in tables, extracting path to reach a desired destination and adding it in the header flit. In this thesis, we have made an effort towards designing and prototyping a standardized and efficient RNI for both source and distributed routing. VHDL is used as a design language and prototyping of both types RNI has been carried out on Altera DE2 FPGA board. Testing of RNI was conducted by using Nios II soft core. Simulation results show that the best case flit latency, for both types RNI is 4 clock cycles. RNI design is also resource efficient because it consumes only 2% of the available resources on the target platform.

**Key Words**

Network on Chip (NoC)

System on Chip (SoC)

Resource Network Interface (RNI)

On Chip Communication

Distributed Routing

Source Routing

Altera FPGA

Nios II Core

Quartus II Design Tool

VHDL
Acknowledgement

First of all we are thankful to Almighty Allah, the Most Beneficent, the Most Merciful whose blessings provided us the strength to successfully complete our Masters Thesis.

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

We thank Rickard Holsmark for his supervision and support. We appreciate his time and cooperation that helped us in enhancing our thesis design.

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

We would also like to thank Lennart Lindh, Acting Professor at Embedded Systems department for helping us in resolving tools related issues.

We would like to thank Saad Mubeen, a member of our research group. We appreciate the discussions which we had with him during our thesis work.

Last but not least, we are very thankful to our parents for their motivation during our difficult hours and support throughout our life. Their love and supplications helped us in materializing our goals and in completing our thesis with success. We are also thankful to all our family members for their support and encouragement throughout our studies.
Table of Contents

1 Introduction............................................................................................................. 1
   1.1 System on Chip............................................................................................... 1
   1.2 The System on Chip Interconnect Problem ......................................................... 1
   1.3 NoC: A New Way to Design Complex Systems ..................................................... 2
   1.4 Resource Network Interface (RNI) in NoC ......................................................... 3
   1.5 Thesis Motivation and Objectives ..................................................................... 4
   1.6 Thesis Layout .................................................................................................. 4

2 Theoretical Background ......................................................................................... 5
   2.1 Options for Communication Infrastructure of SoCs ................................................ 5
       2.1.1 Direct Interconnection ................................................................................. 5
       2.1.2 Buses .......................................................................................................... 6
       2.1.3 Network on Chip ......................................................................................... 6
   2.2 Communication Networks .................................................................................. 7
       2.2.1 OSI Model .................................................................................................. 7
       2.2.2 Network Topologies .................................................................................... 8
       2.2.3 Network Terminology ................................................................................ 10
       2.2.4 Switching Techniques ................................................................................ 11
   2.3 Components of NoC .......................................................................................... 13
       2.3.1 Core or Resource ....................................................................................... 13
       2.3.2 Resource Network Interface (RNI) ............................................................. 14
       2.3.3 NoC Router ............................................................................................... 14
   2.4 Routing Methods .............................................................................................. 15
   2.5 Prototyping Electronic Systems ......................................................................... 16
       2.5.1 Programmable Logic FPGA ..................................................................... 16
       2.5.2 Development Tools .................................................................................. 20

3 RNI: Options and Hardware Design Decisions................................................. 24
   3.1 RNI Functionality ............................................................................................. 24
   3.2 Issues and Assumptions ................................................................................... 25
       3.2.1 Variable Packet Size and Flit Size ............................................................... 25
       3.2.2 Buffer Size ................................................................................................ 25
       3.2.3 Interface with Core ...................................................................................... 25
       3.2.4 Addressing of Core and Corresponding RNI .............................................. 26
       3.2.5 Interface with Router .................................................................................. 26
       3.2.6 Flitization and Deflitzation ....................................................................... 26
   3.3 Common Design Decisions for Distributed and Source Routing RNI ............... 27
       3.3.1 NoC Size .................................................................................................... 27
       3.3.2 Packet Size ................................................................................................ 27
       3.3.3 Buffer Size ................................................................................................ 27
       3.3.4 Handshaking Signals ................................................................................... 28
       3.3.5 Packet Buffering ........................................................................................ 28
       3.3.6 Interface and Communication .................................................................... 28
   3.4 Design Decisions for Distributed Routing RNI ............................................... 29
       3.4.1 Packet Format ............................................................................................. 29
       3.4.2 Flit Level Decision ...................................................................................... 30
   3.5 Design Decisions for Source Routing RNI ....................................................... 33
       3.5.1 Path Computation ....................................................................................... 33
       3.5.2 Packet Format ............................................................................................. 33
       3.5.3 Flit Level Decision ...................................................................................... 34
   3.6 Summary of Decisions ...................................................................................... 38
4 Resource Network Interface Design ........................................... 39
  4.1 RNI VARIANTS: DISTRIBUTED OR SOURCE ROUTING........................................ 39
  4.2 INTERFACE SIGNALS ......................................................................................... 39
    4.2.1 Interface between Nios System and RNI ......................................................... 39
    4.2.2 Interface between RNI and Router ................................................................. 40
  4.3 RNI FOR DISTRIBUTED ROUTING: INTERFACE PROTOCOLS................................. 40
    4.3.1 Core-RNI-Router Data Transmission .............................................................. 40
    4.3.2 Router-RNI-Core Data Transmission ............................................................ 41
  4.4 RNI DESIGN FOR DISTRIBUTED ROUTING ....................................................... 42
    4.4.1 Blocks on Core to Router Path ........................................................................ 42
    4.4.2 Blocks on Router to Core Path ....................................................................... 48
  4.5 RNI FOR SOURCE ROUTING: INTERFACE PROTOCOLS ....................................... 51
    4.5.1 Core-RNI-Router Data Transmission .............................................................. 51
    4.5.2 Router-RNI-Core Data Transmission ............................................................ 51
  4.6 RNI DESIGN FOR SOURCE ROUTING .............................................................. 51
    4.6.1 Blocks on Core to Router Path ........................................................................ 52
    4.6.2 Blocks on Router to Core Path ....................................................................... 54

5 Design Validation, Prototyping and Results ................................. 58
  5.1 DESIGN STRUCTURE ......................................................................................... 58
  5.2 TIMING CHARACTERISTICS FOR DISTRIBUTED ROUTING RNI ......................... 59
    5.2.1 Core to Router Delay ..................................................................................... 60
    5.2.2 Router to Core Delay .................................................................................... 62
    5.2.3 Core to Core Delay ....................................................................................... 63
    5.2.4 Throughput .................................................................................................... 64
  5.3 TIMING CHARACTERISTICS FOR SOURCE ROUTING RNI ................................... 65
    5.3.1 Core to Router Delay ..................................................................................... 65
    5.3.2 Router to Core Delay .................................................................................... 65
    5.3.3 Core to Core Delay ....................................................................................... 66
    5.3.4 Throughput .................................................................................................... 66
  5.4 PROTOTYPING PLATFORM ARCHITECTURE ..................................................... 67
    5.4.1 Design Configuration using SOPC Builder ...................................................... 68
    5.4.2 Integration of RNI with Nios II ...................................................................... 69
  5.5 PROTOTYPE TESTING AND RESULTS FOR DISTRIBUTED ROUTING RNI ................ 70
    5.5.1 Nios II Core to Router Delay ......................................................................... 70
    5.5.2 Router to Nios II Core Delay ......................................................................... 71
    5.5.3 Nios II Core to Nios II Core Delay ................................................................. 73
    5.5.4 Throughput .................................................................................................... 74
    5.5.5 Implementation Results .................................................................................. 74
  5.6 PROTOTYPE TESTING AND RESULTS FOR SOURCE ROUTING RNI ........................ 75
    5.6.1 Nios II Core to Router Delay ......................................................................... 75
    5.6.2 Router to Nios II Core Delay ......................................................................... 75
    5.6.3 Nios II Core to Nios II Core Delay ................................................................. 76
    5.6.4 Throughput .................................................................................................... 76
    5.6.5 Implementation Results .................................................................................. 77
  5.7 SUMMARY AND DISCUSSION ON RESULTS ....................................................... 77

6 Conclusions and Future Work ....................................................... 79
  6.1 SUMMARY .......................................................................................................... 79
  6.2 LIMITATIONS .................................................................................................... 79
  6.3 FUTURE WORK ................................................................................................... 80

7 References ......................................................................................... 81

Appendix 1 ......................................................................................... 83

Appendix 2 ......................................................................................... 85
List of Figures

FIGURE 1-1 SOC BASED ON NOC COMMUNICATION INFRASTRUCTURE ................................. 2
FIGURE 1-2 RESOURCE NETWORK INTERFACE .................................................................. 3
FIGURE 2-1 AN ILLUSTRATION OF DIRECT INTERCONNECT FOR ON-CHIP COMMUNICATION .................................................. 5
FIGURE 2-2 AN ILLUSTRATION OF BUS COMMUNICATION INFRASTRUCTURE FOR SOC ................................................................. 6
FIGURE 2-3 SOC BASED ON NOC COMMUNICATION INFRASTRUCTURE .......................... 7
FIGURE 2-4 7-LAYERS OSI MODEL .................................................................................. 8
FIGURE 2-5 EXAMPLES OF SOME NETWORK TOPOLOGIES ................................................. 9
FIGURE 2-6 COMMUNICATION UNITS .............................................................................. 10
FIGURE 2-7 WORMHOLES ROUTING IN 6X8 MESH TOPOLOGY IN NOC ............................ 12
FIGURE 2-8 4X3 NOC WITH CORE, RNI AND ROUTER ...................................................... 13
FIGURE 2-9 BLOCK DIAGRAM OF NOC ROUTER ................................................................ 14
FIGURE 2-10 3X4 MESH TOPOLOGY NOC ......................................................................... 16
FIGURE 2-11 FPGA ARCHITECTURE PRINCIPLE .............................................................. 17
FIGURE 2-12 ASIC VS FPGA ......................................................................................... 18
FIGURE 2-13 NIOS II PROCESSOR SYSTEM ....................................................................... 19
FIGURE 2-14 MULTIPLE PIO CORES .............................................................................. 20
FIGURE 2-15 DE2 DEVELOPMENT BOARD ....................................................................... 20
FIGURE 2-16 FPGA ADVANTAGE COMPILATION FLOW .................................................. 21
FIGURE 3-1 BLOCK DIAGRAM OF RNI WITH NIOS II CORE AND ROUTER ..................... 24
FIGURE 3-2 PACKET FORMAT .......................................................................................... 29
FIGURE 3-3 PACKET HEADER FORMAT FOR DISTRIBUTED ROUTING ............................. 29
FIGURE 3-4 FORMAT OF HEAD FLIT IN DISTRIBUTED ROUTING ..................................... 31
FIGURE 3-5 FORMAT OF BODY FLIT IN DISTRIBUTED ROUTING ..................................... 31
FIGURE 3-6 FORMAT OF END FLIT IN DISTRIBUTED ROUTING ....................................... 31
FIGURE 3-7 FORMAT OF PACKET HEADER CREATED AFTER DEFLITIZATION ................. 32
FIGURE 3-8 FORMAT FOR BODY AND END OF THE PACKET CREATED AFTER DEFLITIZATION ......................................................... 32
FIGURE 3-9 PACKET HEADER FORMAT FOR SOURCE ROUTING .................................... 34
FIGURE 3-10 FORMAT OF HEAD FLIT IN SOURCE ROUTING ........................................... 35
FIGURE 3-11 FORMAT OF BODY FLIT IN SOURCE ROUTING ........................................... 35
FIGURE 3-12 FORMAT OF END FLIT WITH FLIT TYPE "10" IN SOURCE ROUTING .............. 36
FIGURE 3-13 FORMAT OF END FLIT WITH FLIT TYPE "11" IN SOURCE ROUTING .............. 36
FIGURE 3-14 FORMAT OF PACKET HEADER CREATED AFTER DEFLITIZATION ................ 36
FIGURE 3-15 FORMAT FOR BODY OF THE PACKET CREATED AFTER DEFLITIZATION ......... 37
FIGURE 3-16 FORMAT FOR END OF THE PACKET CREATED AFTER DEFLITIZATION .......... 37
FIGURE 3-17 FORMAT FOR END OF THE PACKET CREATED AFTER DEFLITIZATION .......... 37
FIGURE 4-1 BLOCK DIAGRAM OF COMMUNICATION BETWEEN NIOS II CORE AND ROUTER VIA RNI ................................................ 39
List of Figures

FIGURE 4-2 HANDSHAKING PROTOCOL FOR PACKET TRANSFER FROM CORE TO ROUTER ................................................. 41
FIGURE 4-3 HANDSHAKING PROTOCOL FOR PACKET TRANSFER FROM ROUTER TO CORE ........................................ 41
FIGURE 4-4 INTERNAL STRUCTURE OF RNI FOR DISTRIBUTED ROUTING .......................................................... 42
FIGURE 4-5 C2R-BUFFER FOR DISTRIBUTED ROUTING RNI .................................................................................. 43
FIGURE 4-6 FLITIZER BLOCK FOR DISTRIBUTED ROUTING RNI ........................................................................... 44
FIGURE 4-7 C2R-CONTROLLER FOR DISTRIBUTED ROUTING RNI ......................................................................... 44
FIGURE 4-8 ILLUSTRATION OF FSM FOR C2R-CONTROLLER .................................................................................. 47
FIGURE 4-9 R2C-BUFFER FOR DISTRIBUTED ROUTING RNI ................................................................................... 48
FIGURE 4-10 DEFLITIZER FOR DISTRIBUTED ROUTING RNI .................................................................................... 48
FIGURE 4-11 R2C-CONTROLLER FOR DISTRIBUTED ROUTING RNI ......................................................................... 49
FIGURE 4-12 INTERNAL STRUCTURE OF RNI FOR SOURCE ROUTING ................................................................. 51
FIGURE 4-13 PATH INFORMATION TABLE .................................................................................................................. 52
FIGURE 4-14 FLITIZER FOR SOURCE ROUTING RNI ............................................................................................... 53
FIGURE 4-15 C2R-CONTROLLER FOR SOURCE ROUTING RNI ................................................................................ 53
FIGURE 4-16 R2C-BUFFER FOR SOURCE ROUTING ................................................................................................... 54
FIGURE 4-17 DEFLITIZER FOR SOURCE ROUTING RNI ............................................................................................ 55
FIGURE 4-18 R2C-CONTROLLER FOR SOURCE ROUTING RNI ................................................................................ 56
FIGURE 5-1 VHDL BLOCK MODEL OF CORE, RNI AND ROUTER .................................................................................. 58
FIGURE 5-2 VHDL BLOCK MODEL OF DISTRIBUTED ROUTING RNI ........................................................................ 59
FIGURE 5-3 ILLUSTRATION OF COMMUNICATION FROM CORE TO ROUTER VIA RNI ............................................. 60
FIGURE 5-4 THE COMMUNICATION PROCESS BETWEEN CORE AND C2R-BUFFER OF RNI ................................. 60
FIGURE 5-5 THE COMMUNICATION PROCESS BETWEEN C2R-BUFFER AND ROUTER ......................................... 61
FIGURE 5-6 MINIMUM LATENCY ............................................................................................................................... 61
FIGURE 5-7 SIMULATION RESULT ............................................................................................................................ 62
FIGURE 5-8 ILLUSTRATION OF COMMUNICATION FROM ROUTER TO CORE VIA RNI ............................................. 62
FIGURE 5-9 ILLUSTRATION OF COMMUNICATION FROM CORE TO CORE VIA RNI .................................................. 63
FIGURE 5-10 HARDWARE ARCHITECTURE OF THE SYSTEM ......................................................................................... 67
FIGURE 5-11 SYSTEM COMPONENTS CONFIGURATION IN SOPC BUILDER .............................................................. 68
FIGURE 5-12 ILLUSTRATION OF CREATING SYMBOL FILE OF VHDL FILE IN QUARTUS II ................................. 69
FIGURE 5-13 NIOS II INTEGRATION WITH RNI .......................................................................................................... 69
FIGURE 5-14 INTEGRATION OF NIOS II WITH ROUTER VIA RNI ............................................................................... 70
FIGURE 5-15 INTEGRATION OF ROUTER WITH NIOS II VIA RNI ............................................................................ 72
FIGURE 5-16 INTEGRATION OF CORE TO CORE VIA RNI ......................................................................................... 73
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Flit Type and Codes for Distributed Routing RNI</td>
<td>30</td>
</tr>
<tr>
<td>3-2</td>
<td>Flit Type and Codes for Source Routing RNI</td>
<td>34</td>
</tr>
<tr>
<td>3-3</td>
<td>Payload Size and Code bits of End Flit for Source Routing RNI</td>
<td>35</td>
</tr>
<tr>
<td>3-4</td>
<td>Summary of Design Decisions for both Source and Distributed Routing RNI</td>
<td>38</td>
</tr>
<tr>
<td>5-1</td>
<td>Implementation Results for Distributed Routing RNI</td>
<td>74</td>
</tr>
<tr>
<td>5-2</td>
<td>Implementation Results for Source Routing RNI</td>
<td>77</td>
</tr>
<tr>
<td>5-3</td>
<td>Summary of Implementation Results</td>
<td>77</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
<td></td>
</tr>
<tr>
<td>C2R</td>
<td>Core to Router</td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
<td></td>
</tr>
<tr>
<td>Flit</td>
<td>Flow Control Digit</td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
<td></td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
<td></td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
<td></td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Access Group</td>
<td></td>
</tr>
<tr>
<td>LE</td>
<td>Logic Elements</td>
<td></td>
</tr>
<tr>
<td>NoC</td>
<td>Network on Chip</td>
<td></td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnections</td>
<td></td>
</tr>
<tr>
<td>Phit</td>
<td>Physical Transfer Digit</td>
<td></td>
</tr>
<tr>
<td>PIO</td>
<td>Parallel Input/Output</td>
<td></td>
</tr>
<tr>
<td>R2C</td>
<td>Router to Core</td>
<td></td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computer</td>
<td></td>
</tr>
<tr>
<td>RNI</td>
<td>Resource Network Interface</td>
<td></td>
</tr>
<tr>
<td>RTR</td>
<td>Ready to Receive</td>
<td></td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
<td></td>
</tr>
<tr>
<td>SOPC</td>
<td>System on-a-programmable-chip</td>
<td></td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
<td></td>
</tr>
<tr>
<td>VCT</td>
<td>Virtual Cut Through</td>
<td></td>
</tr>
<tr>
<td>VHDL</td>
<td>Very high speed integrated circuit Hardware Description Language</td>
<td></td>
</tr>
<tr>
<td>WR</td>
<td>Write</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

1.1 System on Chip

In the past four decades, the extreme and rapid improvement in silicon technology gives a new way to semiconductor industry to outperform itself in the electronics world. The transistors growth has significantly increased on a chip. In advanced Integrated Circuits (ICs), one billion transistors can be put on the same chip. Due to this improvement in manufacturing capabilities and higher capacity, it is possible to connect several components with each other on a single chip. This makes the designs more complex. To reduce this complexity, System on Chip (SoC) provides a platform where a large number of cores (Intellectual property, simple cores) can be interconnected on a single chip.

Core Based Design

A core is a pre-designed block, normally implementing a special function that can be used for the design of a system on chip. System on Chip (SoC) is built by integrating the cores or IP-cores [17]. Commonly used cores are Nios II, FFT, DSP, Audio Controller, Encryption Module, Application Specific Processor, Video Controller etc. Several chips manufacturing companies purchase pre-designed cores from specialized companies. The two major FPGA manufacturing companies, Altera and Xilinx provide many free cores, including Nios II and Microblaze.

1.2 The System on Chip Interconnect Problem

In SoC, generally busses are the means to connect different cores [23]. A bus provides a shared medium to communicate between all cores. The scalability is not high and a large numbers of cores cannot be connected to the bus. Multiple cores cannot communicate at the same time on the shared medium. It may be hard to accomplish the communication requirements of the different cores using shared buses. Therefore point to point communication and hierarchical busses are also be used to interconnect the cores in SoC [24].

The communication requirement of a large number of cores in a SoC is high. A large bundle of wires will be required to build a new system with more cores. It is difficult to accommodate several cores in a specific area and reuse the system.
1.3 NoC: A New Way to Design Complex Systems

The functionality and complexity of modern electronic systems are increasing in SoC. The common techniques used for communication infrastructure of SoC are not sufficient due to their poor scalability and reusability.

Network on Chip (NoC) is a new approach for communication among different cores in a System on Chip (SoC) [3]. The basic concept of NoC is similar to common computer networks. The router interconnects each core with the other cores in the NoC. The layered based communication protocols are used in NoC. The reusability of resources is high and it provides high scalability and flexibility to the SoC designer [1].

A simplified SoC with NoC infrastructure is shown in Figure 1-1. The cores are connected with router through Resource Network Interface (RNI). The function of RNI is similar to the function of the network card in a PC for connecting it to the internet [7]. The router is considered as the backbone of the NoC system. The router is connected with other four neighbouring routers and one end with RNI. The job of a router is to receive packets from core through RNI and send it to the destination core within the NoC.

Different research groups have proposed a large number of different NoC architectures [1] [6]. Topology, core selection and routing algorithm are the three most important design issues of a NoC. Selection of a topology is important in NoC design as the design of a router depends on it. Different topologies are present and the commonly used topologies are mesh, star, bus and ring. Core selection is another important issue in a NoC design. Usually the topologies used in the NoC have fixed sized tiles for cores. Therefore, the selected core must fit in that fixed sized tile. The network design can be heavily
optimized if the used cores are equal in size [27].

The communication performance of a NoC mainly depends on the routing method used. The routing methods can be classified into distributed routing and source routing. In distributed routing the header carries the destination address and some control bits. The router selects the route path either by looking up the routing tables or executing the routing function in hardware [20]. In source routing the information about the route path from source to destination is written in packet header at the source end. The routing tables with routing information are placed inside each source. The sender resource selects route path from the tables and places this information in the packet header. When packet reaches at each router, the route path is read from packet header and forward to corresponding neighbour router until it reaches at destination.

1.4 Resource Network Interface (RNI) in NoC

Core is connected to router through RNI. Design of RNI needs to consider the I/O structure of the core and protocols used in the network at physical, data link and network layers. A core communicates within network through RNI using packets. RNI functionality can be considered to have two parts, the resource dependent and resource independent part as shown in Figure 1-2.

![Figure 1-2 Resource Network Interface](image_url)

The resource independent part is generally designed in such a way that the connected router cannot differentiate between RNI from another router. RNI seems to be another router connected with the router. The resource independent part handles interface to the router. Resource dependent part is connected with core and it deals with the control signals, data and address bus width. If the set of cores have the same interface and control signals, then resource dependent part is common for this set. Otherwise it is different for each core. In case of source routing, RNI also contains the routing tables and is responsible for adding the complete path information in the head flit.
1.5 Thesis Motivation and Objectives

NoC researchers have focussed more on issues related to router design, communication infrastructure, low power and fault tolerance. Comparatively less focus has been put on the design of network interfaces. So we decided to design and prototype a generic Resource Network Interface (RNI) for source and distributed routing on Altera FPGA board. In order to achieve this goal, the first step was to perform general and analytical analysis of RNI (functionality), source and distributed routing. To perform prototyping, the second step was to learn Altera FPGA board and development software. The next step was to develop the specifications for RNI design which are compatible with the available Nios II core. After that, a RNI connecting a Nios II processor to the NoC router will be developed in VHDL. The performance of the design was evaluated through VHDL simulation. The last step was to prototype the designed RNI on an Altera FPGA board and performs analysis of speed and cost of the FPGA.

1.6 Thesis Layout

Chapter two describes some general network concepts, the main components of NoC and the routing methods. It also discusses the description about the hardware and development tools that will be used to prototype the RNI.

Chapter three discusses the functionality of the RNI, some issues and assumption related to design. The overall design decisions at all stages of the design for both source and distributed routing RNI are presented.

Chapter four describes the complete design and internal structures of RNI for both source and distributed routing.

Chapter five describes the prototyping of designed RNI on Altera FPGA board. The obtained results are presented and discussed.

Chapter six concludes by listing summary of contributions and the proposals for some future works in this area.
2 Theoretical Background

This chapter describes the theoretical background to understand the problem addressed and our contributions. It starts with discussion on various communication infrastructure options of SoCs. The chapter also gives a brief description about communication network, network terminology, switching techniques and routing methods. A brief introduction about Network on Chip (NoC) concept is also given. The some details about programmable hardware and tools for prototyping will be discussed at the end of this chapter.

2.1 Options for Communication Infrastructure of SoCs

2.1.1 Direct Interconnection

The direct interconnection is one method used for the communication among cores in SoC. In this case, the cores are directly connected to each other through dedicated wires. Main drawbacks of this type of interconnections are that, as the number of cores increases in the network it requires a lot of wires, I/O pins and large routing area [3].

The system with direct interconnection is not scalable. It is very difficult to reuse this system and the utilization of routing resources is also very low. Due to electromagnetic field across the wires the noise level is high. This can affect the quality of signal for on-chip communication infrastructure. An illustration of the direct interconnection is shown in Figure 2-1.

![Figure 2-1 An Illustration of Direct Interconnect for On-Chip Communication](image)
2.1.2 Buses

Most current SoCs use bus based interconnect for communication between the cores. A bus offers shared medium between all cores. Every core in the system is connected with the bus through an interface. Arbiter is the main control unit of the bus. When a core wants to send data to another core through bus, the arbiter adds some control signals and sends it to the destination core.

The number of pins for the cores and wires are reduced in bus based systems as compared to direct connections. Bus has good performance for small systems. The scalability and performance is affected when the system becomes large. This is because more cores are connected through the same bus and share the same communication bandwidth. A bus based communication infrastructure is shown in Figure 2-2.

![Figure 2-2 An Illustration of Bus Communication Infrastructure for SoC](image)

2.1.3 Network on Chip

Network on chip (NoC) is a new design paradigm to overcome the interconnect problems in System on Chip (SoC). In NoC paradigm, a network oriented approach is used in which the cores are connected to each other through a network of routers. NoC provides high performance, high scalability, and reusability when compared to buses and direct interconnections. A SoC based on NoC communication infrastructure is shown in Figure 2-3.
2.2 Communication Networks

2.2.1 OSI Model

Open System Interconnection (OSI) is presented by International Organization for Standardization (ISO) as a reference model [8]. It is composed of seven layers. From top to bottom are the Application, Presentation, Session, Transport, Network, Data link and Physical layer. Each layer is specifying a particular function. There is virtual communication link between the corresponding layers in the source as well as destination as shown in Figure 2-4. It helps the two connected systems to communicate at corresponding layer. Each layer in the model is self-contained and can implement tasks independently. Generally a subset of the seven OSI layers is used for on-chip communication in SoCs.

Function of protocol layers in SoC context

On chip-communication can be expressed in terms of three lower layers of OSI model i.e. physical, data link and network layer. The Transport and Application layers issues are also addressed by researchers but they will not be considered in this thesis. The functionality of three lower layers for on-chip communication is described below.

- Physical Layer

The electrical specifications are defined in physical layer. The physical layer is responsible for control signals, clock signals for every connection, layout of...
Theoretical Background

pins, length of wire, number of wires etc.

- **Data link Layer**
  This layer is responsible for the reliable communication of data across the physical link. Data link layer deals with the number of bits, error correction and detection techniques.

- **Network Layer**
  Network layer determines the routing of packets from source to destination through network switches. The function of this layer includes routing decisions and packet buffering. This layer also handles the quality of service in terms of jitter, delay [8], packet priority etc.

![7-Layers OSI model](image)

*Figure 2-4 7-Layers OSI model*

### 2.2.2 Network Topologies

Different nodes can be connected with each other in different ways depending on the network topology used. Some of network topologies are described below and are illustrated in *Figure 2-5.*
Theoretical Background

2D Mesh Topology

In mesh topology, each node has physical connection with four other nodes in the network. If one node is faulty in mesh topology, then there is less chance of total network breakdown. Mesh topology is divided into two types those are full mesh topology and partial mesh topology. In full mesh topology data is transmitted directly from one node to all other nodes in the network, whereas in case of partial mesh topology the nodes are not directly connected with each other. Full mesh is costly when compared to partial mesh.

Star Topology

In star topology, the nodes are connected with each other through a central switch. All switching activities are controlled by this switch. Data must pass through central switch before reaching destination node. If the central switch fails in star network, then all nodes are immediately isolated.

Ring Topology

Every node is connected with two neighbour nodes forming a shape of ring. When each node in a ring receives the data, it sends to the neighbour nodes in such a way that each node holds the data.
2.2.3 Network Terminology

Some network terminologies which are commonly used are described below.

**Message**

The message is actual data to be transferred from source core to destination core in NoC. The size of message may be fixed or variable and it depends on the application.

**Packet**

The message can be broken into several packets. Packet is a small formatted block that can be transmitted from a source core to destination core. The packet can move independently in the network. Every packet consists of control information and data (payload). The packet header carries the control information. The size of packet may be fixed or variable.

**Flow Control Digit (Flit)**

A packet may be broken down into several flits. The size of flit consist usually one or several bytes. It fits the storage resources in switches in the network.

**Physical Transfer Digit (Phit)**

It is the smallest physical unit of information at physical layer. It consists of a constant number of bits. It is transferred as a unit across a channel from one router to the next router. Size of phit may or may not be equal to the size of flit. In this thesis we are assuming, the size of a flit is equal to the size of a phit and will be discussed in chapter 3.

*Figure 2-6 Communication Units*
2.2.4 Switching Techniques

Switching techniques determine how network resources are allocated to a message on a route between a source and destination node. These techniques allow each switch (many times called router) to send data properly without any loss in the network. If proper switching techniques are not used, the performance of the whole network is affected. Some switching techniques are described below according to [2].

Circuit Switching

In circuit switching a physical path is established from source to destination during communication. Data is transmitted across network until it reaches at destination. An acknowledgement signal is sent to source end from destination end. The whole path is now reserved from source to destination for the transmission of rest of data. When the transmission is completed then the path is free for next data.

Packet Switching

In packet switching, the message is broken into packets and then transmitted from the source to the destination across the network. Every packet has some control and routing information. The packets are transmitted over a shared network [21]. Each packet of message is individually routed from source to destination. As the packet makes progress using the intermediate routers one link at a time is allocated till destination.

- **Store and forward**

  In store and forward switching technique, the whole packet is transmitted and stored in intermediate router’s buffer. Then the packet is forwarded to the selected neighbour router. Packet can only be forwarded after the complete reception. This technique requires large buffer size to store the complete packet in the router. The disadvantages of this technique are the higher delay and requirement of large buffer.

- **Virtual cut through**

  In virtual cut through (VCT) switching technique, the packet is divided into flits. If there is space in the buffer of next router on the path, then head flit is sent and other flit follows it. If buffer in the next router is full, then all the flits wait in the router. This allows the head flit to keep moving if there is a space in router buffers without waiting for other flits of the packet. It still required require buffer space to store the complete packet.
• **Wormhole**

In Wormhole switching technique, the packet is divided into flits of different types i.e., Head, Body and End flits. The first flit is head flit which contains the routing information. The head flit is followed by the body flits which carry the payload. end flit is the last flit in the group of flits corresponding to a particular packet. It carries the payload and also packet end information.

As the head flit advances along the route path it locks the path for remaining flits. The path is unlocked when the end flit is transmitted. The head flit moves along specified route path, so the flow of remaining flits will be in a pipelined fashion. When the head flit can not proceed, the other flits also halt until the head flit is able to proceed.

Consider a 6X8 NoC. The source “A” connected to (2, 2) router wants to send 4 flits packet to destination “B” connected to (5, 6) router. First the head flit is transmitted from source “A” to neighbour router. Depending on the routing information and algorithms, the head flit travels through the routers and lock the path for rest of flits till destination B. The remaining flits must follow the dedicated path (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 6), (4, 6), (5, 6) to reach at destination B. The flits movement looks like a worm as shown in *Figure 2-7*. The wormhole switching technique required small size of buffer as compared to store and forward switching technique. Since routing of flits is in a pipelined fashion, so the latency of the packet is smaller.

*Figure 2-7 Wormholes Routing in 6X8 Mesh Topology in NoC*
2.3 Components of NoC

A NoC can be described in terms of Resource (Core), Resource Network Interface (RNI) and Router (Switch). Figure 2-8 illustrates a 3X4 mesh topology NoC and its components.

![Figure 2-8 4X3 NoC with Core, RNI and Router](image)

2.3.1 Core or Resource

A core is a pre-designed block, normally implementing special function that can be used for the design of a system on chip. Some commonly used cores are Nios II, FFT, DSP, Audio controller, Encryption module, I/O controller, Memories, Power Module, Application specific processor, Video controller etc. The Pre-designed cores are available from several specialized companies. Many chip manufacturing companies buy the design of these cores and fabricate their chip using these cores.

2.3.1.1 Core Categorization

Depending on the flexibility/configurability, the cores can be classified into three categories.

- **Hard Core**

A hard core consists of the physical arrangement information of the design. The hard core can interconnect with soft core along with other peripherals on the same chip [25]. The hard cores are highly optimized for area and performance. The hard cores are technology fixed. The flexibility is minimum because of fixed design of the core. ARM, Virtex-II and Lexra are provided as hard cores by many core providing companies.
• **Soft Core**

A soft core is a functional description of an IP Core [26]. The soft core is generally available in hardware description language like VHDL. The soft cores are reconfigurable. The flexibility rate is higher than hard cores. It is easy to modify the function and structure of soft cores in RTL description. Altera’s Nios II and Xilinx’s Microblaze are examples of commercial soft cores.

• **Firm Core**

A firm core contains the gate-level netlist information. It can be fixed at any place between soft and hard core. The flexibility of firm cores is higher than hard cores but less than soft cores.

### 2.3.2 Resource Network Interface (RNI)

Core is connected to router through RNI. The function of RNI is similar to the function of the network card in a PC for connecting it to the internet [3] [7]. The internal design of an RNI can be portioned into two parts i.e. resource dependent and resource independent as shown in *Figure 1-2*.

### 2.3.3 NoC Router

The idea of NoC router is taken from computer network router. Router has very important role and is considered to be the backbone of NoC. Router forwards the packet from one of its input ports to any one/more of its output ports. Different researches have proposed different architectures for NoC router [1] [6]. A typical NoC route for a 2D mesh topology NoC is shown in *Figure 2-9*.

![Figure 2-9 Block Diagram of NoC Router](image-url)
Each router is connected with other four neighbour routers and one end is connected with resource through RNI. In packet switching networks the core sends packet to the router. Packet arrives at the input port of the router. The buffer stores the packet before sending it to the output port of the router. Each output port of router has a multiplexer. It receives the packets from different queues and then selects one packet to send it to the input port of next router. Switch control is the main important component of the router. The routing protocols and algorithms are implemented in the switch control block.

## 2.4 Routing Methods

Routing methods are used to determine the route followed by the message in the network. Communication performance of a NoC mainly depends on the routing methods used in the network. The routing methods can be classified into distributed and source routing.

### Distributed Routing

In distributed routing the routing functions are implemented in each router of the network. Packet header is very compact. It carries the destination address and some control bits. Each router contains information about the neighbour routers. When the packet arrives at the input port of the router, the route path is selected either by looking up the routing tables or executing the routing function in hardware [20]. As the router used for distributed routing requires additional hardware for execution of routing logic and extra memory unit to store the routing tables, so the design of the router can be complex. Distributed routing is favourable for regular topologies so that same implementation can be used for all routers [19].

### Source Routing

In source routing the information about the route from source to destination is written in the packet header at the source end. The source makes all routing decisions before packet transmitted into the network. The routing tables are placed inside the RNI. The tables are filled with routing information. The sender RNI selects route path from the tables and placed this information in packet header. The packet is transmitted in the network through RNI. When packet reaches at an intermediate router, the route path is read from packet header and forwarded to corresponding neighbour router till it reaches the destination.

Consider a 3X4 mesh topology of NoC as shown in Figure 2-10. DSP core wants to send a packet to I/O-unit core. Consider the X-Y routing algorithm is applied. The packet travels between (1, 2) and (2, 3) routers. The complete route path for the packet is (1, 2), (1, 3), (2, 3). Source routing requires less routing time as the packet header contains the route information. Therefore, the router can immediately select the output port to forward the packet. The router design for the source routing is simple as compared to distributed routing, because it does not require an extra memory for the routing tables in the router.
Theoretical Background

2.5 Prototyping Electronic Systems

This section describes some details about the reconfigurable component technologies. Nios II processor core and some other soft cores and the development tools that have been used in the prototyping in the thesis are described in following subsections.

2.5.1 Programmable Logic FPGA

FPGA is acronym for Field Programmable Gate Array. FPGA is a semiconductor device that can be configured using programming procedure. FPGA consists of programmable logic blocks called logic elements (LE). The physical connections are established to these blocks through programmable interconnects shown in Figure 2-11. The logic blocks can be configured to perform the simple logic gates AND, OR, XOR etc or complex combinational functions such Adder, Multiplexer, decoder etc. The architecture of FPGA logic block is different within different families even for the same company. The basic architecture for logic block consists of 4-input look-up table (LUT) along with flip flop, signal routing MUX. LUT is a simple storage element and can implement Boolean function. The size of LUTs can affect the resource utilization within block [10]. Mostly 4-input LUT is used for logic blocks. But in recent years the manufactures are using 6-input LUT for better performance [9].

The three common technologies are used in FPGA are SRAM based technology, flash-based technology and anti-fuse technology. SRAM based technology FPGAs can be configured very quickly and can be reconfigured a
Theoretical Background

lot of times. But SRAM is volatile. If power supply is disconnected during configuration or running mode, these devices will lose stored configuration (program). So an additional non-volatile memory such as flash memory or EEPROM is required outside to FPGA circuit to store the program. The program is loaded from flash memory chip to FPGA when the power is applied.

The Flash based technology FPGA does not require an additional memory. The flash cells are configured to store the programme. Due to non-volatile property it holds the program even after the failure of the power. The speed of flash based FPGA is slower than SRAM based FPGA. Anti-fuse technology FPGAs can be configured only once. This technology has non-volatile memory to store data configuration. The speed of circuit using anti-fuse based FPGAs is faster than SRAM based FPGAs.

Advantages

The major advantage of FPGA is the ability of re-programming and rapid prototyping. The designer can implement and prototype his design on FPGA more easily and quickly instead of manufacturing a new chip for the same design. The Non-Recurring Engineering (NRE) cost of FPGA is lower than Application Specific Integrated Circuit (ASIC) shown in Figure 2-12.

![Figure 2-11 FPGA Architecture Principle](image)

Xilinx and Altera are two major manufacturer companies of the FPGA. Other FPGA manufacturers are Atmel, Actel, Aerflex UTMC, Lattice semiconductor,
NEC, Quick Logic. Altera has launched different FPGA families such as Stratix II, Stratix III, Arria GX, Arria II GX, Cyclone II and Cyclone III.

![Figure 2-12 ASIC vs FPGA](image)

2.5.1.1 Altera Cyclone II FPGA

Cyclone II is low cost FPGA device family. It contains 68,416 logic elements (LE), 622 useable I/O pins and 1.1 M-bit of embedded memory, Phase-Locked Loops (PLLs) and Multipliers. Cyclone II built on TSMC’s (Taiwan Semiconductor Manufacturing Company) 90-nm process technology using 300-mm wafers [11]. It supports Nios II processor. Multiple Nios II processors can also be used in a design with this family device. Large complex digital systems can also be implemented. Multiple communication modes such as JTAG, passive serial and active serial are supported. The driving voltage for Cyclone II is multiple (3.3V, 2.5V and 1.8V). Cyclone II also supports power-on-reset. In power-on-reset (POR) the reset signal is generated through the FPGA interface when the power is on. Cyclone II FPFA has been used in Automobile, Communication Technology and Video Processing Systems.

2.5.1.2 Nios II Soft Processor

Nios II is a user configurable 32-bit RISC soft core processor [5]. Nios II processor is relatively easy to use and offers great flexibility to the designer, thus making it one of the most popular embedded systems. The complex processor systems and digital systems can be designed with Nios II core.

Nios II processor has 32 bit instruction set, 32 general purpose registers, Software Development Environment (IDE) etc. Nios II processor system is just like microcontroller or other processor system. Nios II provides set of peripherals such as on chip memory, Parallel Input/Output (PIO) ports, SDRAM etc. Custom processor based system can be designed in SOPC (System on-a-programmable-chip) Builder tool using Nios II with peripherals. Architecture of Nios II system is shown in Figure 2-13. Nios II is connected with other peripherals through Avalon bus. It is configured through JTAG debug module in the system. Three versions of Nios II processor are available.
• **Nios II/f - Fast**

The Nios II/f is designed for fast performance. This core provides more features such as optional divide circuitry, hardware multiply circuitry etc.

• **Nios II/s – Standard**

The Nios II/s core can be used to design small and simple systems. This core provides hardware multiply, divide options, 5-stage pipeline etc.

• **Nios II/e – Economy**

Nios II/e core is the lowest cost Nios core processor. This core does not provide exception handling and pipelined memory access.

![Figure 2-13 Nios II Processor System [5]](image)

**2.5.1.3 Parallel Input/Output (PIO)**

Nios II core can be connected with an external device to send and receive signals through Parallel Input/Output (PIO) [13]. PIO is interface with Nios II through Avalon bus as shown in Figure 2-14. PIO is standard interface for Nios and general purpose I/O ports. Nios II core addressed PIO through Avalon bus. PIO sends configured data to an external device outside FPGA. The width of PIO I/O ports is up to 32 bits. The configurations for PIO core can be input only, output only, both input and output and bidirectional mode with tristate control. PIO can be configured with Nios system and other devices in SOPC builder tool. Multiple PIOS can also be used with Nios processor system.
2.5.2 Development Tools

2.5.2.1 DE2 Development Board

The DE2 Development and Educational Board is a platform for building Nios based system. The board is composed of Cyclone II 2c35 FPGA which contains 33,216 Logic Elements (LEs) [12]. The board provides multiple in-built and on board clocks, LEDs, 7-Segment Display, 16x2 LCD display, the standard interface (JTAG, Serial, Ethernet, and USB blaster) and additional user interface. The operating voltage for the board is 9V DC. The layout of board is shown in Figure 2-15.
2.5.2.2  FPGA Advantage Design Tool

FPGA Advantage is an advanced design tool for FPGA based design provided by Mentor Graphics [22]. FPGA Advantage design flow is shown in Figure 2-16.

![FPGA Advantage Compilation Flow](image)

- HDL designer tool is used to create the design module.
- The behaviour of design in the context of real time signals is checked in simulation process. ModelSim is used to simulate the design.
- In synthesis the design is compiled and at end the design is transferred into FPGA.

2.5.2.3  Quartus II FPGA Design Tool

A processor system can be designed and prototyped on FPGA through development board. The implementation of processor system in System on-a-programmable-chip (SOPC) is done through Quartus II [14]. SOPC is also a development tool and part of Quartus II. The design flow of Quartus II is shown in Figure 2-17.

- At the design entry, the design modules are created in the SOPC builder tool. The Quartus II supports VHDL (.vhd), Verilog HDL (.v) and Block diagram file design entry methods.
- In Synthesis step, the design is compiled. Compilation checks the error in the design file and at the end gives a summary of compilation results.
- In Fitting step, the design is fitted into the target device. Fitter places and routes the design which also involves pins assignment.
The theoretical background includes:

- The device programming image of the design is created in the Assembler module.
- The design is analyzed and reports timing information in Timing Analysis module.
- The behaviour of design in the context of real time signals is checked in Simulation step.
- The design files are downloaded into FPGA through communication link between PC and FPGA board such as USB blaster cable.

**Figure 2-17 Quartus II Compilation Flow**

### 2.5.2.4 SOPC Builder

SOPC (System On-a-Programmable-Chip) Builder is the system development tool [14]. A system with Nios II processor and other components can be created in SOPC Builder. The system without processor can also be created in SOPC Builder. It is divided into two parts, one is graphical user interface (GUI) and the second is generating program. In GUI part, the components are configured and added. The “Generate Program” creates the program files for the design. SOPC Builder is part of Quartus II tool.
2.5.2.5 ModelSim

ModelSim is a powerful simulator for both ASIC and FPGA designs [22]. Multi language designs can be verified in ModelSim. The VHDL or Verilog designs can be simulated in ModelSim. ModelSim is used to perform the timing or functional simulation of Quartus II generated designs. Mentor Graphics tool also uses ModelSim.

2.5.2.6 Nios II Integrated Development Environment (IDE)

Nios II IDE is a graphical software development tool for programming the Nios II processor [15]. IDE provides Eclipse tool environment. After writing a program for Nios II, it is built and run. The JTAG download cable is the only communication medium for Nios II processor. The projects are also import from Nios command shell in IDE. The C/C++ and assembly languages are used to program Nios II processor in IDE.
3 RNI: Options and Hardware Design Decisions

This chapter starts with the functionality of RNI and proceeds with some issues and assumptions related to the design. Finally, the design decisions of RNI for both source and distributed routing are discussed. These decisions are related to NoC size, Buffer size, Packet size and Flit formats etc.

3.1 RNI Functionality

The main goal of the thesis is to design and develop the RNI for Altera FPGA implementation. RNI is the interface between a core and a router. The Figure 3-1 shows how a core is connected to a network router using a RNI.

![Figure 3-1 Block Diagram of RNI with Nios II core and Router](image)

The core used is Nios II processor. It is a user configurable 32-bit RISC soft core processor. The RNI has different internal blocks, namely Buffers, Flitizer, Deflitizer and Controller. Controller is the main block of the RNI, which controls all the control signals and flow of packets from core to router and from router to core. When core wants to send a packet to another core in the network, it first sends the packet to the RNI which will store it temporarily in its buffer. When the router is ready to receive the packet, RNI flitizes the packet and forwards it to the router. Similarly when RNI receives the packet from router, it deflitizes the received packet and stores it in the buffer. Whenever core is ready to receive the packet, RNI transfers the stored packet from its buffer to the core. Some control signals are used to synchronize the communication between core and RNI and between RNI and router. Wormhole switching technique is used in packet transferring from RNI to router and from router to RNI.
3.2 Issues and Assumptions

3.2.1 Variable Packet Size and Flit Size

Packets can be of different sizes. The size of the packet depends on the type of message that core wants to send. The packet that is sent from source core to the destination core will be first stored temporarily in the RNI buffer. To store the complete packet, RNI should have buffer size that is equal to the maximum size of a packet. It is difficult to use large size of buffers because of cost and other issues.

The maximum packet size should be known before forwarding packet from core to RNI, so that the same size of buffer can used in the RNI. Therefore it is assumed that packet size will be variable but the maximum size is fixed. The maximum size of a packet is assumed to be 512-bits (64 bytes) divided into 16 flits of 32-bits each.

3.2.2 Buffer Size

Buffers in RNI are used to store the packets temporarily while they are transferred from the source core to the destination core and while receiving data coming from another core. The size of the buffer should not be less than the size of one packet. Otherwise there are chances of losing data from packet when router (or core connected to RNI) is not be ready to receive the packet. Therefore the maximum size of the buffer must be at least equal to the maximum size of a packet. As the maximum size of packet is fixed i.e. 512 bits, so it is assumed that the maximum size of the buffer will also be fixed to 512 bit (sixteen 32-bit flits).

3.2.3 Interface with Core

A core is a pre-designed block, normally implementing special function that can be used for the design of a system on chip. Nios II core can be connected with external devices to send and receive data through parallel input/output (PIO). PIO is a standard interface for Nios II processor. The maximum available width of PIO port is 32-bits. Therefore Nios II core can send and receive 32-bits of data at a time.

To match with PIO width, we use flow control digit (Flit) size between core and RNI and RNI to core to be 32-bits. That means phit size is same as flit size.
3.2.4 Addressing of Core and Corresponding RNI

Every core needs a unique address in the network to communicate with other cores in the network. It is useful for reliable communication for the destination core to know the address of core which sent the message. RNI which is connected to the core shares the address with the core, because only one RNI is connected to each core in the network.

There are two options to specify the source address to the destination core. The first option is that the core itself adds its own source address in the packet when sends it to the RNI. The second option is that the RNI adds the source address during flitization in the head flit and sends it to the router. In this thesis we are considering the second option. The reason is that it reduces the overhead of the core.

3.2.5 Interface with Router

There are two options available for communication protocol between RNI and router. The first option is called Credit based scheme and the second option is Ready to Receive based scheme.

In Credit based scheme, router always sends a signal to the RNI that represents the number of available spaces in its buffer. This signal helps to implement back pressure flow control in the network.

In case of Ready to Receive based scheme, the router will send one bit Ready to Receive (RTR) signal to the RNI whenever it has free space in its buffer. Interface design is simpler in this case. We have selected the ready to receive scheme in this thesis work.

3.2.6 Flitization and Deflitization

To transfer data from source core to the destination core without any data loss, some extra control information should be added to the packet i.e., flitization. Similarly when the packet is received at destination core, the extra control information that was added to the packet should be removed to get the actual data i.e. deflitization. The flitization and deflitization processes can overload the cores. To reduce overhead on the cores, the flitization and deflitization processes will be done in the RNI.
3.3 Common Design Decisions for Distributed and Source Routing RNI

3.3.1 NoC Size

NoC size is the main parameter that should be decided first before the design of network. NoC size will decide the required number of bits to represent the core address in the packet. With current technology upto 50 core NoCs are possible.

We assume that the maximum possible size of NoC will be 8X8 i.e., 64 cores, then minimum 6-bits are required to represent the destination address of the core, which can be used to identify at most 64 locations (cores). So at least 12-bits are required in the packet header to present the destination and source address of the core.

The maximum size of NoC for source routing can only be 7X7. As the path encoding scheme for source routing developed by [16] [18] requires (2 * maximum path length) bits to encode path information. In 7X7 NoC, as the maximum path length is 13, so it requires 26-bits for path encoding. The remaining 6-bits are used to carry Source ID.

3.3.2 Packet Size

To specify the maximum buffer size that should be used, the maximum size of the packet should be fixed. So in this thesis, we are assuming that the maximum size of the packet will be 512 bits or 16 flits of 32-bit each.

A packet can have minimum of 1 flit and maximum of 16 flits when distributed routing is used. In case of source routing, minimum size of packet should be 2 flits while maximum size will be 16 flits. In source routing the minimum size of packet is 64-bits, this is because the first flit of packet just contains control bits and does not contain any payload (data). So to send even one byte of payload, one body flit has to be sent.

3.3.3 Buffer Size

In both distributed and source routing RNI two different buffers are used. The sizes of two buffers of distributed routing RNI are same and are equal to 512-bits. But in case of source routing RNI, the sizes of two buffers differs from each other. The size of one buffer is equal to 512-bits and the size of the other is equal to 544-bits.
3.3.4 Handshaking Signals

Two separate 1-bit wires namely RTR (Ready to Receive) and WR (Write) are used for handshaking signals. The width of these wires is same at both the ends of RNI.

3.3.5 Packet Buffering

Packet buffering is the process of temporarily storing the packets in the intermediate blocks while they are moving from source to destination.

In RNI there are two different buffers to temporarily store the packets during their transmission from source to destination. The first buffer is used to temporarily store the packets that are to be transmitted from core to router, whereas the second buffer is used to temporarily store the packets that are moving from router to core.

In the case of data transfer from core to router, first the RNI will receives the whole packet from core into its buffer and it does not receive any further packet until it transfers already received whole packet to the connected router i.e. single packet buffering. After transferring the whole packet to the router, it can start receiving the next packet from core.

Similarly in the case of data transfer from router to core, the RNI will receive the whole packet from router into its buffer and it will not receive the next packet from router until it transfers the whole received packet to the core. Once it transfers the whole received packet to the core, it again starts receiving the next packet from router.

3.3.6 Interface and Communication

3.3.6.1 Interface and Communication between Core and RNI

The data bus width from Nios II core to RNI and from RNI to Nios II is 32-bits, whereas the width for each control signal is 1-bit.

The communication between Nios II core and the RNI will be half duplex. This means, the RNI can either receive the packet from Nios II core or it can send the packet to Nios II core.

3.3.6.2 Interface and Communication between RNI and Router

The bus width for data from the RNI to router and from the router to RNI is 34-bits and the width for each handshaking signal is 1-bit. The communication between RNI and router will be full duplex. This means the RNI can send and receive packet to the router at the same time.
3.4 Design Decisions for Distributed Routing RNI

3.4.1 Packet Format

The maximum size of a packet is fixed i.e 512 bits. A packet can be divided into three types of flits, namely HEADER, BODY and END of the packet. The packet header contains the first 32 bits of the packet while the rest is payload. The end 32-bits of payload are called as end of the packet. The bits between header and end bits are called body of the packet. The packet format is shown in Figure 3-2.

![Figure 3-2 Packet Format](image)

**Packet Header Format**

The size of the packet header is 32-bits. As the maximum size of NoC is 8X8, minimum 6-bits are required to represent the node address in the network. In the header flit the first 6-bits represent the destination address of the core in the network. The next 6-bits represent the packet size. Packet size helps in tracking the arrival of the whole packet. The next 4-bits carry the “Packet Sequence Number”. The packet sequence number is used to rearrange the packet in correct order at the destination. The next 8-bits are unused for future use. The rest 8-bits are payload. The packet header format is shown in Figure 3-3.

![Figure 3-3 Packet Header Format for Distributed Routing](image)
3.4.2 Flit Level Decision

3.4.2.1 Flit Size

After receiving a packet from the core, the RNI converts it into flits. This process is called Flitization. A packet can have minimum 1 flit and maximum 16 flits. The size of a flit is kept fixed and is equal to 34-bits.

3.4.2.2 Flit Types

First two bits of each flit constitute “Flit Type” as shown in Figure 3-4. A flit can be of three types and those are HEAD, BODY and END flit type. Each type of flit is encoded by the scheme is given in Table 3-1.

<table>
<thead>
<tr>
<th>Flit type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Flit with full Payload</td>
<td>00</td>
</tr>
<tr>
<td>Head Flit</td>
<td>01</td>
</tr>
<tr>
<td>Body Flit</td>
<td>10</td>
</tr>
<tr>
<td>End Flit</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3-1 Flit Type and Codes for Distributed Routing RNI

3.4.2.3 Flit Format after Flitization

Head Flit

Head flit is the first flit of a packet that enters into the network through resource network interface. In distributed routing, this flit carries first 24-bits as control information and next 2-bits are unused while rest of the 8 bits are payload. Head flit is used for locking the path for the following body flits and end flit while traversing through the network.

Two types of codes are used to represent the type of head flit i.e. “00” and “01”. “00” is used when the original packet from the core was only 32-bits including packet header. Hence, in that case there will be only one flit that corresponds to the original packet and there will be no body and end flits. When flit type “01” is used the original packet was more than 32 bits. In that case, packet can have both body and end flits or just end flit which depends on the size of the packet.
Format of head flit is shown in Figure 3-4. Apart from Flit Type, the head flit also contains Destination Address, Packet Size, Packet Sequence Number and Source Address. These terms have already been discussed in the earlier sections.

<table>
<thead>
<tr>
<th>2-bits</th>
<th>6-bits</th>
<th>6-bits</th>
<th>4-bits</th>
<th>6-bits</th>
<th>2-bits</th>
<th>8-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flit Type</td>
<td>Destination Address</td>
<td>Packet Size</td>
<td>Packet Sequence Number</td>
<td>Source Address</td>
<td>Unused</td>
<td>Payload</td>
</tr>
</tbody>
</table>

**Figure 3-4 Format of Head Flit in Distributed Routing**

**Body Flit**

Body flit follows the head flit. It always carries the payload. After flitization, a packet may correspond to minimum of zero body flit and maximum of 14 body flits depending upon the payload contained in the original packet. Body flit is represented by flit type equal to “10”. Format of a body flit is shown in the following Figure 3-5.

<table>
<thead>
<tr>
<th>2-bits</th>
<th>32-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Payload</td>
</tr>
</tbody>
</table>

**Figure 3-5 Format of Body Flit in Distributed Routing**

**End Flit**

End flit is the last flit in the group of flits corresponding to a particular packet. End flit follows the last body flit. It unlocks the path for the packet to which it
belongs. It should be noted that the path was locked by the head flit of the same group of flits. End flit also carries the payload. End flit has the same flit format as that of a body flit except the code of Flit type and is represented by flit type equal to “11”. Format of end flit is shown in the above Figure 3-6.

### 3.4.2.4 Flit Format after Deflitization

The process of converting the flits into a packet is known as deflitization. The deflitization process is started after receiving the 34-bits head flit from the router and continues till it receives the end flit in the RNI. Deflitization is needed for all cores in the network, because the cores only recognize the packets and not the flits.

**Head Flit**

When RNI receives the 34-bit head flit from router, it removes the “Flit Type” and “Destination Address” bits from the flit. The “Source Address” bits are moved to left most position and represent the first 6-bits of the packet header. The “Packet Size” and “Packet Sequence Number” bits are also shifted to LSB side by 2-bits. The next 8-bits are unused. Rest of 8-bits are payload. The created 32-bit packet header is stored in the RNI buffer. The created packet header format is shown in Figure 3-7.

**Body Flit and End Flit**

Both body and end flits are deflitized by removing the “Flit Type” bits and the rest 32-bits payload is transferred to the buffer in RNI. The format for both body and end of the packet is same and shown in Figure 3-8.
3.5 Design Decisions for Source Routing RNI

3.5.1 Path Computation

What is Path Computation?

Path computation is to find the complete route information from a source core to a destination core. In source routing, the route path from every core to the other cores in the network should be computed. For example, in an 8X8 NoC every core will have the complete route path information of other 63 cores as destinations.

Choice of Path Computation

There are two ways to carry out path computation. First method is to compute paths and store them in each resource in the form of tables. The second method is to compute the paths at runtime using an algorithm in each resource. The second method is costlier and slow.

In this thesis, the table based path computation method has been selected because a Matlab based Path Computation Tool for source routing “MatPC Tool” has already been designed by our research group [16] [18]. So it is a good idea to use an existing tool to compute paths offline for source routing and store them in the memory of every source core in the network.

3.5.2 Packet Format

The maximum size of a packet is fixed. Like distributed routing a packet can be divide into three parts and those are HEAD, BODY and END of the packet. The packet header contains first 32 bits of the packet while the rest is payload. The end 32-bits of payload are called end of the packet. The bits between header and end bits are called BODY of the packet. The packet format is same as that of distributed routing RNI and is shown in Figure 3-2.

The structure of packet header is different from distributed routing RNI. The size of packet header is fixed and is equal to 32-bits. Since the maximum network size is 7X7, 6-bits are required to represent address of the resource. The first 6-bits of the header represent the destination address of the resource. The next followed 6-bits are reserved for the packet size. The rest of 20 bits are unused and left for the future use. The header packet format is shown in Figure 3-9.
3.5.3 Flit Level Decision

3.5.3.1 Flit Size

A packet can have minimum of 2 flits and maximum of 16 flits same as in distributed routing. The size of a flit is kept fixed and is equal to 34-bits.

3.5.3.2 Flit Types

First two bits of each flit constitute “Flit Type” as shown in Figure 3-10. A flit can be of three types and those are HEAD, BODY and END flit type. The flit types and corresponding codes are shown in Table 3-2.

Table 3-2

<table>
<thead>
<tr>
<th>Flit type</th>
<th>Code</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 with full payload</td>
<td>10</td>
</tr>
<tr>
<td>End Flit with payload less than 4 bytes</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3-2 Flit Type and Codes for Source Routing RNI

3.5.3.3 Flit Format after Flitization

Head flit

In source routing, head flit always carries control information and there will be no payload in it. As opposed to distributed routing, we have used only one code for head flit. This code constitutes the start of flit and is always equal to “00”. The next 6-bits carry the source address. Remaining 26-bits carry the complete route information. The format for the head flit is shown in the following Figure 3-10.
Body flit

In source routing, a packet may correspond to a minimum of 1 body flit and maximum of 14 body flits depending upon the payload contained in the original packet. Body flit is represented by flit type equal to “01”. Format of a body flit is shown in Figure 3-11.

End flit

End flit can have one of the two codes “10” and “11” in the first 2-bits of the flit i.e. Flit Type. If the value is “10” the end flit has 32-bits payload. If the end flit has value “11” the payload is 8 to 24 bits and that is determined by the 2-bits “Payload Size” next to the Flit Type. “Payload Size” represents the number payload bytes present in the end flit. This scheme is used for RNI to know the exact payload size of a packet sent by source core. The remaining 6-bits are unused. The representation of Payload Size value bits is shown in Table 3-3. Format for the end flit with Flit Type “10” is shown in Figure 3-12 and with Flit Type “11” is shown in Figure 3-13.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Payload Size Code bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte Payload</td>
<td>01</td>
</tr>
<tr>
<td>2 Byte Payload</td>
<td>10</td>
</tr>
<tr>
<td>3 Byte Payload</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3-3 Payload Size and Code bits of End Flit for Source Routing RNI
3.5.3.4 Flit Format after Deflitization

The deflitization process starts after receiving the 34-bits head flit from the router and continues till it receives the end flit into the RNI.

Head Flit

The packet header for source routing RNI is created by removing the “Flit Type” and “Route Path Information” bits from the head flit. The “Source Address” bits are moved to LSB position which represents first six bits of the packet header. The calculated packet size will be added next to the source address bits. The method of calculating the packet size will be discussed in the chapter 4. The rest 20-bits are unused. The created packet header format is shown in Figure 3-14.

![Figure 3-14 Format of Packet Header Created after Deflitization]
**Body Flit**

Body flits are deflitized by removing the “Flit Type” bits and the rest 32-bits payload is transferred to the destination core. The format for body of the packet is shown in *Figure 3-15*.

![Figure 3-15 Format for Body of the Packet Created after Deflitization](Image)

**End Flit**

As there are two types of end flits, the deflitization process is different for both end flits. When “Flit Type” is “10” then the “Flit Type” bits are removed and rest 32-bits of payload is transferred to the destination core. The format for end of the packet when flit type is “10” is shown in *Figure 3-16*.

When “Flit Type” is “11”, the “Flit Type” and “Payload Size” bits are removed. The payload bits are moved to LSB position. The size of payload can be 1, 2 or 3 bytes which depends on the payload size bits. The rest of the bits are unused. The format for end of packet is shown in *Figure 3-17*.

![Figure 3-16 Format for End of the Packet Created after Deflitization](Image)

![Figure 3-17 Format for End of the Packet Created after Deflitization](Image)
3.6 Summary of Decisions

The design decisions at all levels for both distributed and source routing RNI are summarised in Table 3-4.

Table 3-4 Summary of Design Decisions for both Source and Distributed Routing RNI

<table>
<thead>
<tr>
<th>Routing Method</th>
<th>Maximum NoC Size</th>
<th>Maximum Packet Size (Bits)</th>
<th>Buffer Size (Bits)</th>
<th>Flit Size (Bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Buffer 1</td>
<td>Buffer 2</td>
</tr>
<tr>
<td>Distributed</td>
<td>8X8</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Source</td>
<td>7X7</td>
<td>512</td>
<td>512</td>
<td>544</td>
</tr>
</tbody>
</table>

Table 3-4 Summary of Design Decisions for both Source and Distributed Routing RNI
4 Resource Network Interface Design

The purpose of this chapter is to explain the design and internal structure of RNI for both source routing and distributed routing. This chapter will focus on top level model of RNI block, its internal blocks, interface signals and interface protocols.

4.1 RNI Variants: Distributed or Source Routing

Two RNI designs will be described, one for distributed routing and the other for source routing. In distributed routing RNI, destination address is added in the packet header. Instead in source routing RNI, the complete route information is added in the packet header.

4.2 Interface Signals

The connection between Nios II core, RNI and router is shown in Figure 4-1. The functionality of the RNI is described in Section 3.1. The detail of each interface is discussed in the following subsections.

4.2.1 Interface between Nios System and RNI

Nios System consists of Nios II core, Avalon bus and PIOs. Nios II core and PIOs are connected through an Avalon Bus. Avalon bus is provided by Altera for connecting on chip processors with other peripherals. The PIOs which receives data from resource are connected at slave port of Avalon bus and the PIOs which send data to resource are connected at master port of Avalon bus. The system is generated and configured using System-on-Programmable-Chip
Resource Network Interface Design

(SOPC) Builder tool. This tool automatically connects the system components with the Avalon bus.

RNI is connected with Nios II core through PIOs. The size of PIOs is different for each interface. The two 32-bit PIOs are used for data transfer and the rest four 1-bit PIOs are used for control signals, namely RTR (Ready to Receive) and WR (Write). The direction of each PIO is shown in Figure 4-1.

4.2.2 Interface between RNI and Router

Two different sizes of busses are used to connect RNI and router. Two of them are 34-bit wide and the remaining four are 1-bit wide each. All busses are unidirectional as shown in Figure 4-1. The two 34-bit buses are used to send and receive the data and four 1-bit wires are used for control signals.

4.3 RNI for Distributed Routing: Interface Protocols

4.3.1 Core-RNI-Router Data Transmission

Two separate 1-bit wires are used for handshaking signals namely, RTR (Ready to Receive) and WR (Write) as shown in Figure 4-1. The functionality of these handshaking signals is same on both sides of RNI. The handshaking protocols are discussed with respect to RNI. The handshaking protocol used for the packet transfer from the core to router via RNI is explained below and is shown in Figure 4-2.

Initially RNI sends the RTR signal high to the core. When core receives the RTR signal high, it sends a WR signal high and a chunk (i.e. 32-bits) of the packet to RNI. When RNI receives the WR signal high, it reads the packet chunk. After reading the data, RNI sends RTR signal low to core and waits for WR signal low, i.e. one chunk of the packet has been transferred from core to RNI. For the next chunk of the packet, RNI again sends the RTR high to core. This process continues until RNI has received the complete packet. This type of handshaking communication is used because core and RNI may not be synchronized and may work at different clock speeds.

Similarly when RNI receives an RTR signal high from the router, it sends the WR signal high and sends a flit (i.e. 34-bits) to router and then waits for RTR signal low from the router. When the RNI receives RTR signal low, it sends the WR signal low to router, i.e. one flit has been transferred from the RNI to router. To transfer next flits, RNI again waits for RTR signal high from the router. This process continues until RNI transfers all the flits to router.
4.3.2 Router-RNI-Core Data Transmission

The handshaking protocol used for packet transfer from the router to core is explained below and is shown in Figure 4-3.

The RNI sends a RTR signal high to router. When RNI receives the WR signal high from the router, it reads the flit. Then it sends a RTR low to router and
waits for WR signal low, i.e. RNI has been received one flit from the router. When RNI receives WR signal low, it again sends a RTR signal high to the router for the next flit. This process continues until RNI receives the end flit.

When RNI receives RTR signal high from the core, it sends WR signal high and a flit to the core and waits for RTR signal low. When RNI receives RTR signal low, it sends WR signal low to core. It means one flit has been transferred from RNI to core. This process continues until all the flits have been transferred from RNI to core.

### 4.4 RNI Design for Distributed Routing

The detailed internal structure of RNI for distributed routing is shown in *Figure 4-4*. The RNI has six internal blocks and those are C2R-Buffer, Flitizer, C2R-Controller, R2C-Buffer, Deflitizer and R2C-Controller. Each block is performing their defined specific job. The functionality of each block and control signals are discussed in the following subsections.

![Figure 4-4 Internal Structure of RNI for Distributed Routing](image)

#### 4.4.1 Blocks on Core to Router Path

RNI has different blocks and control signals as shown in *Figure 4-4*. In order to keep the core simple and not to be overloaded, flitization and deflitization processes are done in RNI. The detailed functionality of each component is discussed in the following subsections.
4.4.1.1 C2R-Buffer

C2R-Buffer is a FIFO (First-In-First-Out) buffer which is connected to the input port of RNI from the core side. It has sixteen locations to store a packet and the size of each location is equal to 32-bits i.e. a chunk size of the packet, see Figure 4-5. A chunk of the packet is defined as a part of the packet and the size of a chunk is equal to 32-bits. Whenever it receives the “Write Enable” signal from the C2R-Controller it stores the data coming from the core at the specified location. The location will be specified by the signal “Add Write” from C2R-Controller. It sends the stored chunk of the packet to flitizer block whenever it receives the “Read Enable” signal from C2R-Controller. It sends the chunk from the address location which is specified by the signal “Add Read”. This buffer can simultaneously store the chunks of the packet coming from core and can send the stored packet to flitzer block.

![Figure 4-5 C2R-Buffer for Distributed Routing RNI](image)

4.4.1.2 Flitizer

The process of converting a packet into flits is called flitization. The signals entering and leaving flitizer block are shown in Figure 4-6. Flitization process starts when it receives the “Flitizer Enable” signal from the C2R-Controller. It reads the 32-bit of a packet i.e. a chunk from the C2R-Buffer. If the “Flit Type” signal value received from the C2R-Controller is either “00” or “01”, then the chunk of the packet which is read from the C2R-Buffer is assumed to be the header of the packet. If the “Flit Type” value is “00”, it means the packet contains only one flit. In that case, body and end flits are not present in the packet. If the “Flit Type” value is “01”, it means the packet contains more than one flit. In flitization process, it adds 2-bit flit type in the “Flit Type” field and 6-bit source address in the header of the packet i.e. from bit numbers 18 to 23, creates a 34-bit head flit. The format of the head flit was shown in Figure 3-5. When it receives the signal “Flit Type” as “10” or “11” then it assumes that the incoming packet from the C2R-Buffer is body or end of the packet respectively. In these cases it just adds the “Flit Type” to the flit at the field specified for flit type i.e. from bits 0 to 1, creates a 34-bit body and end flits. The formats of body and end flits were shown in Figure 3-6 and Figure 3-7 respectively. After flitization process, the flits are forwarded to router.
4.4.1.3 C2R-Controller

C2R-Controller is the main block of the RNI that generates all control signals for proper flow of packets from the core to router. Internally C2R-Controller has different subcomponents and each is performing their specific tasks. In C2R-Controller different control signals apart from data are entering and leaving. These subcomponents and control signals are shown in the following Figure 4-7.

C1 : 6 – Bit Counter

This 6-bit counter is used to count the total number of payload bytes of the packet coming from the core to RNI. Initially it sets to “000000”. When packet header arrives from the core, the corresponding bits in the packet header which represents the size of payload bytes will be stored in this counter. A chunk of the packet is defined as a part of the packet and the size of a chunk is equal to...
32-bits. The counter value of C1 is decremented by 4 whenever C2R-Buffer receives a new chunk of the packet. If the counter value is less than 4, then the controller identifies that the last chunk of the current packet is left to receive into the C2R-Buffer and hence, the counter will be reset i.e. equal to “000000” after receiving the last chunk of the packet.

**C2 : 4 – Bit Counter**

This 4-bit counter is used to locate the address of C2R-Buffer to store the receiving packet from the core. Initially counter value is sets to “0000”. The counter value of C1 is incremented by 1, whenever a new chunk of the packet is stored in the C2R-Buffer. When complete packet is received, the controller stops sending RTR signal to core.

**C3 : 6 – Bit Counter**

This counter counts the total number of payload bytes (Packet Size) that has been transferred to router from the C2R-Buffer. Initially the counter value is sets to “000000”. Similar to C1 counter, when packet header is received from the core, the corresponding bits of size will be stored in the counter. The counter value is decremented by 4, whenever a chunk of the packet is forwarded from C2R-Buffer to flitizer block. If counter value becomes less than 4 then the controller identifies that it has left with only one chunk of the packet to send to the router and hence, the counter value will be reset i.e. “000000” after sending the last chunk of the packet.

**C4 : 4 – Bit Counter**

This 4-bit counter is used to locate the address of the C2R-Buffer from where the chunk of the packet has to be transferred to flitizer. Whenever a chunk of the packet is transferred from C2R-Buffer to flitizer, the counter value will be incremented by 1. After forwarding all the chunks of the a packet to flitizer, the counter value will be reset to “0000”.

**F1 : FSM**

This block is responsible for controlling the whole communications from the core to router in RNI. It first sends the RTR signal high to core and waits for the WR signal high from the core. When F1 receives the WR signal high it sends the “Write-En” and “Add-Write” signals to C2R-Buffer to store the coming chunk of the packet from the core. The “Add-Write” signal is the value of counter C2. The “Write-En” signal enables the writing and “Add-Write” signal specifies the location where to write in the C2R-Buffer. Then F1 sends RTR signal low to core and waits for the WR signal low from the core. When F1 receives the WR signal low, then it checks the values of C1 and C2. If the value of C1 becomes “000000” or C2 becomes “1111” i.e. the RNI has received the complete packet from the core. Then F1 stops sending the RTR signal high to core. Otherwise it sends the RTR signal high to core in order receive the next chunk of the packet. This process continue until the RNI
receives the complete packet from the core.

Once the header of the packet is received and stored in the C2R-Buffer from the core then F1 checks the RTR signal from the router. If the RTR signal from the router is high then it checks the counter values of C3 and C4. If the counter values C3 and C4 are not equal to “000000” and “1111” respectively. Then F1 sends “Read-En” and “Add-Read” signals to C2R-Buffer, “Flitizer-En” and “Flit Type” signals to flitizer and WR signal high to router. After sending flit to router, it waits for RTR signal low from the router. When F1 receives RTR signal low from the router, it sends WR signal low to router. Every time the RTR signal from router will be checked only after receiving the next chunk of the packet from the core i.e. the incoming packet chunks from the core will keep stores in the C2R-Buffer and simultaneously checks for the RTR signal from the router to send the stored flit. If the RTR from router is high then the packet will be flitized and sent to router. This process will be continued till the RNI sends the complete packet to router. If the counter value of C3 becomes “000000” or C4 becomes “1111” i.e. RNI has sent complete packet to router then F1 sends the RTR signal high to core to receive the next packet.

The “Flit Type” information that is sent to flitizer block will be decided based on the counter values of C3 i.e. “Packet Size” and C4 i.e. “Add-Read”. If the packet is only single byte of payload, then the “Flit Type” of the head flit will be “00” otherwise the “Flit Type” will be “01”. The received packet from the core is single byte of payload or not will be recognized by checking the counter values of C3 and C4. If the counter value of C3 is “000000” and C4 is “0000” the packet not single byte payload packet, otherwise not single byte payload packet. The “Flit Type” for the body flits will be “10”. Similarly body flits will be recognized by checking counter values of C3 and C4 i.e. the counter value of C3 is greater than 4 and counter C4 value is less than “1111”. For the end flit, the “Flit Type” will be “11”. The end flit will be recognized by checking the same C3 and C4 counter values i.e. either the counter C3 value becomes less than 4 or counter C4 value becomes “1111”.

An FSM corresponding to above behaviour was implemented and is shown in Figure 4-8. In FSM only the transition conditions and mealy outputs are shown and Moore outputs are not shown. The transition between the states depends on the handshaking signal and the packet size.
Figure 4-8 Illustration of FSM for C2R-Controller
4.4.2 Blocks on Router to Core Path

4.4.2.1 R2C-Buffer

R2C-Buffer is a FIFO buffer connected to the output port of the RNI towards the core side. It has sixteen locations to store the packet and the size of each location is equal to 32-bits i.e. the size of a flit after deflitization. The detailed block of R2C-Buffer is shown in Figure 4-9. Whenever it receives the “Write Enable” signal from the R2C-Controller, it stores the flit coming from the deflitizer block at a specified address location. The address location is specified by the signal “Add-Write” from the R2C-Controller. Whenever it receives “Read Enable” signal, it sends the stored flit from the specified address location of buffer to core. The address location from where the flit to be read is specified by the signal “Add Read”. R2C-Buffer can store the flits coming from deflitizer and can send the stored flts to core at the same time.

![Figure 4-9 R2C-Buffer for Distributed Routing RNI](image)

4.4.2.2 Deflitizer

This block performs the deflitization process. Deflitization is a process of reconverting the flits into a packet. The signals entering and leaving the deflitizer block are shown in Figure 4-10.

![Figure 4-10 Deflitizer for Distributed Routing RNI](image)

This block starts deflitization process whenever it receives the “Deflitizer Enable” signal from the R2C-Controller and then it reads the 34-bit flit from the router port. It first checks the flit type bits. If the “Flit Type” is head flit type, then it simply removes the “Flit Type” and “Destination Address” bits from the flit and shifts the “Source Address” bits to the “Destination Address”
field and creates a 32-bit packet header. The created packet header exactly matches to the one that was created by the source, only “Destination Address” bits are replaced by “Source Address” bits and rest of the header bits remains the same. As soon as the deflitization process is completed, the created packet header will be sent to R2C-Buffer. When a body and end flit arrives, it just removes the “Flit Type” bits from the body and end flit and the remaining 32-bits of payload are transferred to R2C-Buffer.

4.4.2.3 R2C-Controller

The R2C-Controller block is responsible for controlling the communications from router to core. The R2C-Controller has different sub-blocks those are counters and FSM block.. These subcomponents and control signals are shown in the following Figure 4-11.

![Figure 4-11 R2C-Controller for Distributed Routing RNI](image)

**C5 : 4 - Bit Counter**

It is a 4-bit counter and is used to the locate address of the R2C-Buffer where the flits coming from the router have to be stored. The counter value of C5 is incremented by 1, whenever a flit is stored in the R2C-Buffer.

**C6 : 6 – Bit Counter**

This 6-bit counter is used to count the number of payload bytes transferred to core. The functionality of this counter is similar to the counter C3 as discussed in Section 4.4.1.3. When head flit is received, the corresponding bits of size will be stored in this counter. The counter value will be decremented by 4 on each transfer of one flit to core. When the value of the counter becomes
“000000” then all the flits has been sent to core.

**C7 : 4–Bit Counter**

The value of this counter is used as an address of R2C-Buffer from where the flits need to be read and transferred to resource. It is similar to counter C4 as shown in *Figure 4-7*. The counter value increases by 1 on every transfer of flit from the R2C-Buffer to core.

**F2 : FSM**

The block F2 is the main component that controls all control signals and transfer of flits from router to core. Initially it sends an RTR signal to router to receive the flits. When the WR signal high from router is received. Then it sends “Deflitizer-En” signal to deflitizer block, “Write-En” and “Add-Write” signals to R2C-Buffer and reads the coming flit. The “Add-Write” signal is the value of the counter C5. The FSM block first checks the “Flit Type” bits of the receiving flit. If the received flit is head flit then it reads the “Flit Size” bits from the flit and stores in the counter C6. After reading the flit, it sends RTR signal low to router and waits for WR signal low from the router. Before making request for next flit to router, it always checks flit type bits for end flit type and counter C5 value for “1111”. If either of the condition is true then it stops sending RTR signal high to router. Otherwise it sends a request to router to receive next flit. This process will be continued until it receives end flit.

While it is storing flits from the router into the R2C-Buffer, it also checks for RTR signal high from the core. If it receives the RTR signal high from core then it sends “Read-En” and “Add-Read” signals to R2C-Buffer and WR signal high to core. The “Add-Read” signal is the value of counter C7 that specifies the address location on buffer from where the flit to be read. After sending flit it waits for RTR low signal. Once it receives RTR signal low it sends WR signal low to core. It always checks the counter values of C6 and C7 before sending flits to core. If either counter C6 value becomes “000000” or counter C7 value reaches to “1111”. Then it stops sending flits to core and sends RTR signal high to router to receive next flits. An FSM corresponding to above behaviour was implemented and is similar to the FSM of C2R-Controller and is shown in *Figure 4-8*. 
4.5 RNI for Source Routing: Interface Protocols

4.5.1 Core-RNI-Router Data Transmission

Two separate 1-bit signals are used for handshaking in source routing RNI as shown in Figure 4-1. In source routing, the communication protocol used for transferring the packets from the core to router is same as that used for distributed routing RNI, as discussed in Section 4.3.1. The handshaking protocol for packet transfer from core to router is shown in Figure 4-2.

4.5.2 Router-RNI-Core Data Transmission

The protocol used for packet transfer from router to core is similar to the protocol that is used for distributed routing RNI as discussed in Section 4.3.2. But the RNI will not start the process of sending the received flits to core until it receives the complete packet from the router. The detail of the handshaking protocol is shown in Figure 4-3.

4.6 RNI Design for Source Routing

The block diagram of RNI for source routing is shown in Figure 4-12. The internal structure of RNI for source routing is same as for distributed routing except one extra block i.e. Path Information Table and the positions of the R2C-Buffer and Deflitizer are interchanged. The detail of each block is discussed in the following sections.

![Figure 4-12 Internal Structure of RNI for Source Routing](image-url)
4.6.1 Blocks on Core to Router Path

4.6.1.1 C2R-Buffer

The function and structure of C2R-Buffer for source routing is same as that of C2R-Buffer for distributed routing as discussed in Section 4.4.1.1.

4.6.1.2 Path Information Table

Path information table contains complete route information from each source to all other destination resources in the network. There is one entry in the table for each of the destinations. Therefore, the size of the table is equal to the size of NoC. For example, if the address of the destination resource is “101010” then the route information of this destination resource is available at location “101010” in path the information table. This path information table should be stored in RNI before system start up. The signals entering and leaving the path information table are shown in Figure 4-13.

4.6.1.3 Flitizer

The signals entering and leaving flitizer block are shown in Figure 4-14. The process of flitization starts when it receives “Flitizer Enable” signal from the C2R-Controller. When flitizer receives the signal “Flit Type” as “00” from the C2R-Controller, it assumes that the incoming 32-bits are the header of the packet. This extracts the complete route information from the “Path Information Table” corresponding to the destination address indicated in “Destination Address” field in the header of the packet. This adds the “Flit Type” bits at the field specified for flit type. It adds the 6-bits of “Source Address” next to flit type field. In the remaining 26-bits it adds the complete “Route Path Information” which was extracted from the path information table and creates a 34-bit of head flit as shown in Figure 3-8.

When it receives “01” as “Flit Type” from the C2R-Controller, it assumes that the incoming bits are body of the packet. In that case, it adds “01” in flit type field and the remaining 32-bits are payload. The format of the body flit is as shown in Figure 3-9.

If it receives either “10” or “11” as “Flit Type”, it assumes that the incoming bits are end of the packet. When “10”, it adds “10” at the field specified for the
flit type and the remaining 32-bits are payload. The format of the end flit with flit type “10” is as shown in Figure 3-10. When “11”, it adds “Flit Type” at the field specified for the flit type and the “Payload Size” bits next to the flit type field bits. The remaining bits in the end flit are payload. The format of the end flit with flit type “11” is as shown in Figure 3-11. It sends the flits to the router as soon as they flitized.

![Figure 3-10](image)

**Figure 3-10 Flitizer for Source Routing RNI**

### 4.6.1.4 C2R-Controller

The functioning of C2R-Controller in source routing RNI is almost same as that of C2R-Controller of distributed routing RNI. The functioning of counters C8, C9, C10 and C11 is same as that of counters C1, C2, C3 and C4 respectively in distributed routing RNI as discussed in Section 4.4.1.3. The difference is only in FSM block. The internal subcomponents and control signals of C2R-Controller is shown in Figure 4-15.

![Figure 4-15](image)

**Figure 4-15 C2R-Controller for Source Routing RNI**

---

53
In source routing, the process of transferring packet from core to RNI is same as that of distributed routing RNI. The processes of packet transferring from RNI to router is also same as that of process used for distributed routing RNI except the difference is in deciding the flit type value.

When header packet is flitizing it sends “00” as “Flit Type”. When body of the packet is flitizing, it sends “Flit Type” as “01”. While end of the packet flitizing, it checks the counter value of C10. If the counter value of C10 is 4, then it sends the end “Flit Type” as “10”. If the counter value of C10 is less than 4. Then it sends the end “Flit Type” as “11” and also sends the “Payload Size”. The “Payload Size” value will depend on the counter value of C10. For example, if the counter value is 3 i.e. 3-bytes of payload is available in the end of the packet, then it sends “Payload Size” bits as “11”. Similarly, if the counter value C10 is 2 then it sends “10” and if the counter value is 1 then it sends “01” as “Payload Size” bits to flitizer. We never have the end flit with zero byte payload. An FSM corresponding to above behaviour was implemented and is similar to the FSM of C2R-Controller and is shown in Figure 4-8.

### 4.6.2 Blocks on Router to Core Path

#### 4.6.2.1 R2C-Buffer

The R2C-Buffer is connected towards the router side in RNI. It has sixteen locations to store the flits from the router. The size of each location is equal to a flit size i.e. 34-bits as shown in Figure 4-16. It stores the flit, whenever it receives “Write Enable” signal from the R2C-Controller. The received flit will be stored at the location specified by the signal “Add Write”. In source routing, buffer will not transfer the stored flits to deflitizer block until it receives the complete packet from the router. Whenever the “Read Enable” signal arrives, it transfers the flit to deflitizer. The location in the C2R-Buffer from where the flit need be transferred to deflitizer will be specified by the signal “Add Read”.

![Figure 4-16 R2C-Buffer for Source Routing](image)
4.6.2.2 Deflitzer

The deflitzer block for source routing RNI is shown in Figure 4-17. The deflitzation process starts whenever it receives “Deflitzer Enable” signal from R2C-Controller and it reads the 34-bit flit from the R2C-Buffer. First it checks the “Flit Type” field bits, if it is a head flit type then it removes the “Flit Type” and “Complete Route Information” fields from the head flit and adds 6-bit “Packet Size” next to the “Source Address” field. The created 32-bit packet header is in the same format as that of sent by the source core, except the “Destination Address” bits are replaced by the “Source Address” bits. If the “Flit Type” is of body flit type then it removes the “Flit Type” bits and sends the remaining 32-bits of payload to core. Similarly if the “Flit Type” is end flit type then it checks for the type of end flit. If the end flit type is “10” then it removes the “Flit Type” bits and sends the remaining 32-bits payload to core. If the end flit type is “11” then it removes the “Flit Type” bits and reads the next 2-bits i.e. “Payload Size” bits. Depending on the “Payload Size” bits it sends 1 to 3 bytes of payload to core.

![Deflitzer for Source Routing RNI](image)

4.6.2.3 R2C-Controller

The functioning of R2C-Controller for source routing RNI is completely different to that of distributed routing RNI. R2C-Controller has different sub-blocks and their detailed functionalities are described in below subsections. The internal subcomponents and control signals of R2C-Controller is shown in Figure 4-18.

C12 : 4–Bit Counter

The functioning of this counter is same as that of counter C5 and is discussed in Section 4.4.2.3.

C13 : 6–Bit Counter

This 6-bit counter is used to count the size of the payload, which are receiving from the router. When R2C-Buffer receives the head and body flits from the router, then the counter value of C13 increases by 4. When an end flit arrives, the increment of counter value depends on the value of “Payload Size” bits and
hence increment by 1, 2, 3 or 4 is possible.

**C14 : 4-Bit Counter**

The functionality of this counter is similar to counter C7 and is discussed in *Section 4.4.2.3.*

![Figure 4-18 R2C-Controller for Source Routing RNI](image)

**F4 : FSM**

The block F4 is the main component of R2C-Controller, which controls all control signals and transfer of flits from router to core. Initially it sends RTR signal high to router to receive the flits. When it receives WR signal high from router then it sends “Write-En” and “Add-Write” signals to R2C-Buffer and stores the coming flit into its R2C-Buffer. The signal “Add-Write” specifies the location in buffer to store the flit. The “Add-Write” signal value is the value of counter C12. It also checks the “Flit Type” bits, whether it is an end flit or not. If it is not an end flit then it increases the counter value of C13. Then it sends RTR signal low to router and waits for WR signal low. When it receives WR signal low then it again sends RTR signal high to router to receive the next flit. This process will be continued until it receives end flit. Once it receives end flit it stops requesting for next flits to router until it transfers all the received flits which are stored in the R2C-Buffer to core.

While receiving the flits from router it also checks the RTR signal from the core. If it receives RTR signal high from core, it sends “Deflitizer En” signal to deflitizer block, “Read-En” and “Add-Read” signals to the R2C-Buffer and sends WR signal high to core. When the head flit is deflitizing, it sends the counter value of C13 to deflitizer as “Packet Size”. In deflitizer, this “Packet Size” will be added next to the “Source Address” in the header of the packet. Then it waits for RTR low signal from the core. When it receives RTR low
from the core it sends WR signal low to core. before sending next flits to core It always checks whether the counter value of C14 becomes equal to the counter value of C12 or not. If the counter value of C14 becomes equal to the counter value of C12 i.e. it has send all the flits to core then it stops sending flits to core and starts requesting for the next flits to router. An FSM corresponding to above behaviour was implemented and is similar to the FSM shown in Figure 4-8.
5 Design Validation, Prototyping and Results

5.1 Design Structure

The RNI is designed using VHDL language. VHDL (Very high speed integrated circuit Hardware Description Language) is a language commonly used for describing the structural, physical and behavioural characteristics of the digital systems. The VHDL block model of complete system design is shown in Figure 5-1.

![VHDL Block Model of Core, RNI and Router](image)

The dummy core and dummy router are hardware models, which are designed using VHDL. The purpose of dummy core is to generate and receives packets. The dummy core model resembles the behaviour of a real resource. In dummy core a memory is created and a packet is stored in that memory. Whenever RNI is ready to receive the packet, it accesses the packet from the memory and sends to the RNI. Control signals are used for the packet transactions. The RNI unit is internally divided into several processes i.e. two separate state machines for controlling the communication flow from the core to router and from the router to core. The dummy router is designed to receive the packets from the RNI and to generate the packets for RNI. To generate the packets, a memory is created in the router and a packet is stored in that memory. Whenever RNI is ready to receive the packet, it sends the packet from memory. The VHDL top-
level block model is same for both source and distributed routing RNI. The design of RNI has been described in detail in chapter 4. The VHDL block model for distributed routing RNI is shown in Figure 5-2 and similar VHDL block model is also designed for source routing RNI.

![Figure 5-2 VHDL Block Model of Distributed Routing RNI](image)

### 5.2 Timing Characteristics for Distributed Routing RNI

The VHDL model of RNI has been tested separately for both distributed and source routing. FPGA Advantage 7.2 from Mentor Graphics was the tool used to accomplish the RNI design. ModelSim simulator was used to simulate the code. The main objective of the VHDL model is to determine the delay/execution time in terms of clock cycles for various blocks.

The RNI design for distributed routing is tested and verified in three phases. In first phase, the communication from core to router was tested. In second phase, the communication from router to core was tested. In the third and last phase, the communication from core to core was tested. In this test phase, the output of the RNI is given back to RNI as feedback input. The details of steps, setup and results of each phase are presented in the following subsections.
5.2.1 Core to Router Delay

The interface between core and router via RNI is shown in Figure 5-3. The dummy core works as a source core. It generates packets and control signals. Similarly, the dummy router generates control signals and receives the packets from the RNI.

To test the design, a memory was created in dummy core which stored a test packet in that memory. The dummy core sends control signal and a part of the packet whenever the RNI is ready to receive data. The received packet is processed in the RNI and sends to the dummy router whenever it is ready to receive the packet. The number of clocks to transfer a packet from core to router is calculated at different stages and the results are presented below.

![Diagram of Communication from Core to Router via RNI](image)

**Figure 5-3 Illustration of Communication from Core to Router via RNI**

5.2.1.1 Simulation Results

**Core to RNI C2R-Buffer Delay**

The timing diagram of VHDL simulations is shown in Figure 5-4. One clock cycle is required to receive the data into C2R-Buffer from the RNI input port. The turn around time to request for the next data is 3 clock cycles, the reason that these clock cycles occurs are decided by the handshaking protocol. The turn around time is 3 clocks and one clock to store into C2R-Buffer from the RNI input port, it sums up to a period of 4 clocks to receive the next data into C2R-Buffer. Therefore, the delay in clocks cycles to receive a complete packet into C2R-buffer of RNI from the core is:

\[
\text{Delay} = 1 + 4(N-1) \text{ clocks / packet}
\]

Where \( N \) = Number of Flits. A Flit = 32-bits of a packet

![Timing Diagram of VHDL Simulations](image)

**Figure 5-4 The Communication Process between Core and C2R-Buffer of RNI**
**RNI C2R-Buffer to Router Delay**

The timing diagram of VHDL simulations for the communication between C2R-Buffer and the router is shown in *Figure 5-5*. Two clocks are required to read the data from buffer and flitization and to send to the router. Therefore the delay in clocks to transfer a complete packet from the C2R-Buffer to the router is:

\[
\text{Delay} = 2 + 4(N-1) \text{ clocks / packet}
\]

---

**Figure 5-5 The Communication Process between C2R-Buffer and Router**

**RNI Input Port to Output Port Delay**

The timing diagram for the communication from core to router is shown in *Figure 5-7*. One clock is required to receive the packet into the C2R-Buffer from the RNI input port and two clocks are required to read from the buffer and for flitization, it sums up to 3 clocks to transfer from the RNI input port to RNI output port per flit.

\[
\text{Flit Delay} = 3 \text{ clocks / flit}
\]

---

**Figure 5-6 Minimum Latency**

The time to transfer a complete packet from RNI input port to RNI output port is:

\[
\text{Packet Delay} = 3 + 4(N-1) \text{ clocks / packet}
\]
Example:

Let’s assume a packet with size equal to 512 bits (i.e. 16 Flits). The delay in clocks to send a complete packet from the C2R-Buffer to router is:

\[
\text{Packet Delay} = 3 + 4*(16 - 1) = 63 \text{ clocks / packet}
\]

5.2.2 Router to Core Delay

In second test phase, the communication from router to core is tested. The interface between router and core via RNI is shown in Figure 5-8. In dummy router, a memory is created and a packet is stored in that memory. The router accesses the packet from its memory and sends to the RNI when it is ready to receive. The behaviour of accessing the packet from memory is like receiving packet from neighbouring router in the network. The dummy core works as a destination core and receives the packets from the RNI. The simulation results are presented below.

![Diagram of Communication from Router to Core via RNI](image)
5.2.2.1 Simulation Results

Router to RNI R2C-Buffer Delay

\[ \text{Delay} = 2 + 4(N-1) \text{ clocks} / \text{packet} \]

RNI R2C-Buffer to Core Delay

\[ \text{Delay} = 1 + 4(N-1) \text{ clocks} / \text{packet} \]

RNI Input Port to Output Port Delay

\[ \begin{align*}
\text{Flit Delay} &= 3 \text{ clocks} / \text{flit} \\
\text{Packet Delay} &= 3 + 4(N-1) \text{ clocks} / \text{packet}
\end{align*} \]

Example:

Packet Size = 512 bits.

\[ \text{Packet Delay} = 3 + 4*(16 - 1) = 63 \text{ clocks} / \text{packet} \]

5.2.3 Core to Core Delay

In test 3, the complete design of RNI was tested. The output port of RNI is connected to input port of RNI at router side. The interface is shown in Figure 5-9. It is not possible to make the complete network as we have not designed the router, so we have given the output of RNI as input to itself instead of router. The feedback data is assumed as input data from the router.

To test this condition we have designed two different hard cores has designed. One core works as a source core which generates packets. The second core works as a destination core, which receives the packets from RNI. The RNI-C2R represents the RNI part that handles the transfer of packets from core to router. Similarly RNI-R2C represents the RNI part handles the transfer of packets from router to core. The simulation results are presented below.

![Figure 5-9 Illustration of Communication from Core to Core via RNI](image)
5.2.3.1 Simulation Results

Source Core to RNI-C2R Output Port Delay

Delay = 3 + 4(N – 1) clocks / packet

RNI-R2C Input Port to Destination Core Delay

Delay = 3 + 4(N – 1) clocks / packet

Source Core to Destination Core Minimum Delay

Delay = 6 + 4(N-1) + r*d clocks / packet

Where r = Number of router hops,

d = each router delay

Example:

Packet Size = 512 bits.

Assume r = 3 hops and d = 2 clocks delay

Packet Delay = 6 + 4(16-1) + 3*2 = 72 clocks / packet

5.2.4 Throughput

Throughput is defined as the total number of flits processed by RNI per second.

Throughput = 1 / 4 (Flits / Clock)

The packet throughput is:

Throughput = 1 / 4N (Packets / Clock)

Where N = Number of flits / packet

Example:

Consider a clock frequency of F = 50 MHz. Then the flit throughput can be calculated as follows:

Throughput = 1 / (4*(1/(50*10^6))) = 12.5 MFlits / Second

Assume a packet with 16 flits then the packet throughput can be calculated as follows:

Throughput = 1 / (4*16*(1/(50*10^6))) = 0.78 MPackets / Second
5.3 Timing Characteristics for Source Routing RNI

Similar to distributed routing RNI tests, the test of RNI for source routing is also done in three phases. The details of steps, setup and results of each phase is presented in the following subsections.

5.3.1 Core to Router Delay

The steps and setup for this test is same as that of test 1 in distributed routing RNI. The simulation results are presented below.

5.3.1.1 Simulation Results

Core to RNI C2R-Buffer Delay

\[\text{Delay} = 1 + 4(N-1) \text{ clocks/packet}\]

RNI C2R-Buffer to Router Delay

\[\text{Delay} = 2 + 4(N-1) \text{ clocks/packet}\]

RNI Input Port to RNI Output Port Delay

Flit Delay = 3 clocks/flit

Packet Delay = 3 + 4(N-1) clocks/packet

Example:

Packet Size = 512 bits.

Packet Delay = 3 + 4(16-1) = 63 clocks/packet

5.3.2 Router to Core Delay

Similar to test 2 in distributed routing RNI, the steps and setup of source routing RNI for the communication from router to core is same. The simulation results are presented below.

5.3.2.1 Simulation Results

Router to RNI R2C-Buffer Delay

\[\text{Delay} = 2 + 4(N-1) \text{ clocks/packet}\]

RNI R2C-Buffer to Core Delay

\[\text{Delay} = 1 + 4(N-1) \text{ clocks/packet}\]
Design Validation, Prototyping and Results

**RNI Input Port to Output Port Delay**

Packet Delay = 3 + 2 * 4(N-1) clocks / packet

**Example:**

Packet Size = 512 bits.

Packet Delay = 3 + 2 * 4(16-1) = 123 clocks / packet

**5.3.3 Core to Core Delay**

The steps and setup for the communication from core to core for source routing RNI is same as that of distributed routing RNI test 3. The simulation results are presented below.

**5.3.3.1 Simulation Results**

**Source Core to RNI-C2R Output Port Delay**

Delay = 3 + 4(N-1) clocks / packet

**RNI-R2C Input Port to Destination Core Delay**

Delay = 3 + 2 * 4(N-1) clocks / packet

**Source Core to Destination Core Minimum Delay**

Delay = 6 + 8(N-1) + r*d clocks / packet

**Example:**

Packet Size = 512 bits.

Assume r = 3 hops and d = 2 clocks delay

Delay = 6 + 8(16-1) + 3*2 = 132 clocks / packet

**5.3.4 Throughput**

The flit throughput is:

Throughput = 1 / 4 (Flits / Clock)

The packet throughput when the packet transfer from core to router is:

Throughput = 1 / 4N (Packets / Clock)
The packet throughput when the packet transfer from router to core is

\[
\text{Throughput} = \frac{1}{8N} \text{ (Packets / Clock)}
\]

**Example:**

Consider a clock frequency of \( F = 50 \text{ MHz} \) and a packet with 16 flits. Then the packet throughput for data transfer from core to router can be calculated as follows:

\[
\text{Throughput} = \frac{1}{4 \times 16 \times (1/(50 \times 10^6))} = 0.78 \text{ MPackets / Second}
\]

The packet throughput for data transfer from router to core can be calculated as follows:

\[
\text{Throughput} = \frac{1}{8 \times 16 \times (1/(50 \times 10^6))} = 0.39 \text{ MPackets / Second}
\]

### 5.4 Prototyping Platform Architecture

The fundamental components that build up a HW/SW system are shown in *Figure 5-10*.

**Figure 5-10 Hardware Architecture of the System**

Nios II core is connected with JTAG, on-chip RAM and PIOs through Avalon bus. On-chip RAM (Random Access Memory) stores the instructions of Nios II core during process. The JATG UART (Joint Test Access Group Universal Asynchronous Receiver Transmitter) is used to communicate with the host computer. RNI which is interfaced with Nios II core sends and receives the data through PIOs and the router is connected with the RNI.
5.4.1 Design Configuration using SOPC Builder

The design flow starts by making a new project in Quartus II software. SOPC Builder is also a development tool and is part of the Quartus II software. The system is generated and configured in System-on-Programmable-Chip (SOPC) Builder tool. This tool automatically connects system components with the Avalon bus. On board 50 MHz clock frequency is used to run the system. The system was generated in SOPC Builder with Nios II, memory, JTAG and PIO shown in the following Figure 5-11. The system components configuration and setting in SOPC Builder is as follow.

### Nios II

Nios II core family consists of three types of cores, fast (Nios II/f), economy (Nios II/e), and standard (Nios II/s) cores. The detail about each core has been discussed in chapter 2. We are using the “economy (Nios II/e)” because it is optimized for minimum logic usage. It uses 600-700 logical elements (LEs).

![Figure 5-11 System Components Configuration in SOPC Builder](image)

### Memory and JTAG UART

The On-chip RAM memory and JTAG UART is initialized. The size of on-chip RAM is 40 Kbytes and memory width is 32-bits.

### Parallel Input/Output (PIO)

PIO is initialized to transfer and receive the data and control signals. The parameter settings of PIO are different for each interface. The two 32-bit PIOs are configured as input and output ports for data. The four 1-bit PIOs are configured as input and output ports for control signals.
5.4.2 Integration of RNI with Nios II

The generated system (Nios II core, memory, JTAG URAT and PIO) in SOPC Builder is uploaded in the project of Quartus II software. The VHDL file of RNI is imported into the project. The symbol file of VHDL code is created with the help of “Create Symbol File for Current File” option in Quartus II as shown in Figure 5-12.

The RNI block is integrated with Nios II system block. After the pins assignment, the design is compiled. After successful compilation, the design is loaded into the FPGA using programmer. The complete integration of Nios II block with the RNI is shown in Figure 5-13.
5.5 Prototype Testing and Results for Distributed Routing RNI

The system created with Nios II core and RNI in Quartus II project is tested separately for both distributed and source routing on FPGA board. Nios II Integrated Development Environment (IDE) is used to run the software for the Nios II processor. C language code is developed in Nios II IDE for testing purpose and they are given in Appendix 1 and Appendix 2. The DE2 development board is used for prototyping. The detail about the testing steps, setup and results are presented in subsections below.

RNI with Nios II processor system is tested and verified for distributed routing in three phases. Due to unavailability of router design, we have design a dummy router for testing purpose. The description about each testing phase is discussed in the following subsections.

5.5.1 Nios II Core to Router Delay

In this test phase Nios II processor sends the packets to dummy router though RNI using the control signals. To see the packet at router output end, the output port of router is connected to seven segment display on DE2 development board through decoder block. The integration of Nios II processor block, RNI block, dummy router and decoder block are shown in Figure 5-14.

![Figure 5-14 Integration of Nios II with Router via RNI](image)

To generate the packets from the Nios II core, a C-code is written in Nios II IDE. The C-code is given in Appendix 1. An array of 16 flits is created which represent a single packet. Each value of an array represents as a single flit of
the packet. The first value in the array represents as a header of the packet and the remaining values are as body and end of the packet. The communication from core to router via RNI for variable size packets is tested, thus headers with different packet size values are sent to RNI.

The main function consists of two while loops. The first while loop is used to send the complete packet of different sizes, whereas the second while loop is used to send all flits of a packet. The number of flits is to be send is depends on the size of the packet. The design is simulated in Altera ModelSim and calculated the number of clocks required to transfer a complete packet from Nios II core to router. The obtained results are presented below.

### 5.5.1.1 Simulation Results

**Nios II Core to RNI C2R-Buffer Delay**

\[ \text{Delay} = p + 170(N-1) \text{ clocks / packet} \]

Where \( p \) = Number of clocks that Nios II core takes to start process.

**RNI C2R-Buffer to Router Delay**

\[ \text{Delay} = 2 + 170(N-1) \text{ clocks / packet} \]

**RNI Input Port to Output Port Delay**

- Flit Delay = 170 clocks / flit
  - Packet Delay = \( p + 3 + 170(N-1) \text{ clocks / packet} \)

**Example:**

Packet Size = 512 bits.

\[ \text{Packet Delay} = p + 3 + 170(16-1) = 2553 + p \text{ clocks / packet} \]

### 5.5.2 Router to Nios II Core Delay

In this test phase, dummy router sends the packets to Nios II processor though RNI. The received packet can be seen in Nios II IDE tool. The transfer of packet from RNI to Nios II core can also be seen on seven segment display by interfacing the decoder block between them. The integration of dummy router, RNI, Nios II processor and decoder block is shown in Figure 5-15.
Design Validation, Prototyping and Results

To receive the packet in Nios II core, a C code is written and is given in the Appendix 2. The main function consists a while loop, which repeats on receiving one flit. The simulation results are presented below.

5.5.2.1 Simulation Results

Router to RNI R2C-Buffer Delay

Delay = 2 + 4(N-1) clocks / packet

RNI R2C-Buffer to Nios II Core Delay

Delay = 1 + 142(N-1) clocks / packet

RNI Input Port to Output Port Delay

Flit Delay = 142 clocks / flit

Packet Delay = 3 + 142(N-1) clocks / packet

Example:

Packet Size = 512 bits.

Packet Delay = 3 + 142(16-1) = 2133 clocks / packet
5.5.3 Nios II Core to Nios II Core Delay

In this test phase Nios II processor sends and receives the packets though RNI. The integration of RNI and Nios II processor is shown in the Figure 5-16.

![Figure 5-16 Integration of Core to Core via RNI](image)

A C code was written to generate packets and to receive packets in Nios II IDE. The C code is the combination of the codes given in Appendix 1 and Appendix 2. The simulation results are presented below.

5.5.3.1 Simulation Results

**Nios II Core to RNI-C2R Output Port Delay**

\[
\text{Delay} = 3 + 170(N-1) \text{ clocks / packet}
\]

**RNI-R2C Input Port to Nios II Core Delay**

\[
\text{Delay} = 3 + 142(N-1) \text{ clocks / packet}
\]

**Nios II Core to Nios II Core Minimum Delay**

\[
\text{Packet Delay} = 6 + 170(N-1) + 142(N-1) + r*d \text{ clocks / packet}
\]

**Example:**

Packet Size = 512 bits.

Assume \( r = 3 \) hops and \( d = 2 \) clocks delay

\[
\text{Packet Delay} = 6 + 170(16-1) + 142(16-1) + 3*2 = 4692 \text{ clocks / packet}
\]
5.5.4 Throughput

The packet throughput when packets are transferred from Nios II core to router is:

\[
\text{Throughput} = \frac{1}{170(N)} \text{ (Packets / Clock)}
\]

The packet throughput when packets are transferred from router to Nios II core is:

\[
\text{Throughput} = \frac{1}{142(N)} \text{ (Packets / Clock)}
\]

Example:

Consider a clock frequency of \( F = 50 \text{ MHz} \). Assume a packet with 16 flits then the packet throughputs can be calculated as follows:

Core to Router Throughput = \( \frac{1}{(170\times16\times(1/(50\times10^6)))} \) = 18.4 K Packets / Second

Router to Core Throughput = \( \frac{1}{(142\times16\times(1/(50\times10^6)))} \) = 22 K Packets / Second

5.5.5 Implementation Results

The summary of the resources utilization of the device for all three test phases are listed in Table 5-1.

<table>
<thead>
<tr>
<th>Distributed Routing</th>
<th>Total Logic Elements</th>
<th>Total Combinational Functions</th>
<th>Dedicated Logic Registers</th>
<th>Total Memory Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>1,766/33,216 (5%)</td>
<td>1,651/33,216 (5%)</td>
<td>922/33,216 (3%)</td>
<td>272,768/483,840 (56%)</td>
</tr>
<tr>
<td>Test2</td>
<td>2,056/33,216 (6%)</td>
<td>1,920/33,216 (6%)</td>
<td>1,033/33,216 (3%)</td>
<td>338,944/483,840 (70%)</td>
</tr>
<tr>
<td>Test3</td>
<td>2,379/33,216 (7%)</td>
<td>2,189/33,216 (6%)</td>
<td>1,613/33,216 (4%)</td>
<td>339,968/483,840 (70%)</td>
</tr>
<tr>
<td>RNI</td>
<td>654/33,216 (2%)</td>
<td>615/33,216 (2%)</td>
<td>215/33,216 (&lt;1%)</td>
<td>1,536/483,840 (&lt;1%)</td>
</tr>
</tbody>
</table>

Table 5-1 Implementation Results for Distributed Routing RNI
5.6 Prototype Testing and Results for Source Routing RNI

Similar to distributed routing, the RNI for source routing is also tested with Nios II processor in three phases. The description about each testing phase is discussed in the following subsections.

5.6.1 Nios II Core to Router Delay

The procedure and setup is same as that used in distributed routing test 1. The obtained results are presented below.

5.6.1.1 Simulation Results

Nios II Core to RNI C2R-Buffer Delay

\[ \text{Delay} = p + 1 + 170(N-1) \text{ clocks / packet} \]

RNI C2R-Buffer to Router Delay

\[ \text{Delay} = 2 + 170(N-1) \text{ clocks / packet} \]

RNI Input Port to Output Port Delay

- Flit Delay = 170 clocks / flit
- Packet Delay = \( p + 3 + 170(N-1) \) clocks / packet

Example:

Packet Size = 512 bits.

\[ \text{Packet Delay} = 3 + 170(16-1) = 2553 \text{ clocks / packet} \]

5.6.2 Router to Nios II Core Delay

The procedure and setup is same as that used in distributed routing test 2. The obtained results are presented below.

5.6.2.1 Simulation Results

Router to RNI R2C-Buffer Delay

\[ \text{Delay} = 2 + 4(N-1) \text{ clocks / packet} \]

RNI R2C-Buffer to Nios II Core Delay

\[ \text{Delay} = 1 + 142(N-1) \text{ clocks / packet} \]
RNI Input Port to Output Port Delay

Packet Delay = 3 + 4(N-1) + 142(N-1) clocks / packet

Example:
Packet Size = 512 bits.

Packet Delay = 3 + 4(16-1) + 142(16-1) = 2193 clocks / packet

5.6.3 Nios II Core to Nios II Core Delay

The procedure and setup is same as that used in distributed routing test 3. The obtained results are presented below.

5.6.3.1 Simulation Results

Nios II Core to RNI-C2R Output Port Delay

Delay = 3 + 170(N-1) clocks / packet

RNI-R2C Input Port to Nios II Core Delay

Delay = 3 + 146(N-1) clocks / packet

Nios II Core to Nios II Core Minimum Delay

Packet Delay = 6 + 170(N-1) + 146(N-1) + r*k clocks / packet

Example:
Packet Size = 512 bits.

Assume r = 3 hops and k = 2 clocks delay

Packet Delay = 6 + 170(16-1) + 146(16-1) + 3*2 = 4752 clocks / packet

5.6.4 Throughput

The packet throughput when packets are transferred from Nios II core to router is:

Throughput = 1/170(N) (Packets / Clock)

The packet throughput when packets are transferred from router to Nios II core is:

Throughput = 1 / [142(N) + 4(N)] (Packets / Clock)
Example:

Consider a clock frequency of $F = 50$ MHz. Assume a packet with 16 flits. Then the packet throughputs can be calculated as follows:

Core to Router Throughput $= \frac{1}{(170*16*(1/(50*10^6)))} = 18.4$ Kpackets / Second

Router to Core Throughput $= \frac{1}{(146*16*(1/(50*10^6)))} = 21.4$ Kpackets / Second

5.6.5 Implementation Results

The summary of the resources utilization of the device for all three test phases are listed in Table 5-2.

<table>
<thead>
<tr>
<th>Source Routing</th>
<th>Total Logic Elements</th>
<th>Total Combinational Functions</th>
<th>Dedicated Logic Registers</th>
<th>Total Memory Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>2,039/33,216 (6%)</td>
<td>1,929/33,216 (6%)</td>
<td>936/33,216 (3%)</td>
<td>338,304/483,840  (70%)</td>
</tr>
<tr>
<td>Test2</td>
<td>2,024/33,216 (6%)</td>
<td>1,865/33,216 (6%)</td>
<td>1,010/33,216 (3%)</td>
<td>338,368/483,840  (70%)</td>
</tr>
<tr>
<td>Test3</td>
<td>2,420/33,216 (7%)</td>
<td>2,226/33,216 (7%)</td>
<td>1,174/33,216 (4%)</td>
<td>339,456/483,840  (70%)</td>
</tr>
<tr>
<td>RNI</td>
<td>667/33,216 (2%)</td>
<td>631/33,216 (2%)</td>
<td>203/33,216 (&lt;1%)</td>
<td>2,048/483,840    (&lt;1%)</td>
</tr>
</tbody>
</table>

Table 5-2 Implementation Results for Source Routing RNI

5.7 Summary and Discussion on Results

Summary of the results obtained for the RNI with hardware core (dummy core) and with Nios II soft core are listed in Table 5-3.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Routing Method</th>
<th>Throughput (Packets/sec) (RNI-C2R)</th>
<th>Throughput (Packets/sec) (RNI-R2C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy Core</td>
<td>Distributed</td>
<td>0.78M</td>
<td>0.78M</td>
</tr>
<tr>
<td></td>
<td>Source</td>
<td>0.78M</td>
<td>0.39M</td>
</tr>
<tr>
<td>Nios II Core</td>
<td>Distributed</td>
<td>18.4K</td>
<td>22K</td>
</tr>
<tr>
<td></td>
<td>Source</td>
<td>18.4K</td>
<td>21.4K</td>
</tr>
</tbody>
</table>

Table 5-3 Summary of Implementation Results

The maximum throughputs of distributed routing RNI design for different resources are compared as shown in Table 5-3. When RNI is connected with
the hardware core, we derived a throughput of 0.78M packets per second and when it is connected with Nios II soft core, we derived a throughput of 18.4K packets per second to send one packet (i.e. 512-bits) from core to router. Through this analysis we could state that the RNI design with hardware core is 42 times faster than RNI design with Nios II soft core to send one packet.

In case of packet transfer from router to core for distributed routing, the maximum throughput of RNI design with hardware core is 0.78M packets per second which is much higher than the RNI design with Nios II soft core whose throughput is 22K packets per second to send one packet. Similarly for source routing, the maximum throughput of RNI design with hardware core is 0.39M packets per second to send one packet which is 18 times faster than the RNI design with Nios II soft core whose throughput is 21.4K packets per second.

The throughput of Nios II soft core is much lower than the hard core, because soft cores are slow when compare to hardware core and it largely depends on the number of instructions executed in soft core to send one packet.

Implementation results of the RNI design on Altera Cyclone II FPGA shows that it consumes only 2% of the available resources and for one Nios and one RNI, it consumes 70% of the available memory. If we compare the RNI design for distributed routing with source routing, the source routing RNI design utilizes more resources and large memory. If we assume that the design of router consumes less than 30% of available memory, then it is possible to implement only one RNI. To prototype a complete NoC, FPGA with large memory capacity is required.
6 Conclusions and Future Work

This chapter summarizes the thesis contributions and outlines future directions.

6.1 Summary

Resource Network Interface (RNI) provides bridging functionality between a core and a router in the network. It serves as a specific interface between each core and the network and it is important for designing multi-core SoCs using NoC idea. Apart from buffering, it performs flitization and deflitization of packets from the core and packets from the router respectively. In case of source routing, it also accommodates tables which contain complete path information to all its destinations in the network.

In this thesis, we have designed a Resource Network Interface (RNI) for both distributed as well as for source routing specifically for Nios II core. The design of RNI is also prototyped on Altera DE2 FPGA board.

The RNI is developed for a standard I/O port interface provided by core, called Parallel Input/Output (PIO), to the on-chip network. The design of this RNI can be reused for connecting any core, which provides I/O port based interface.

One of the strengths of our RNI design is that it supports both fixed and variable packet sizes up to a maximum size. The handshaking protocol which we have used for data transfer provides reliable communication, when cores and routers are running at different clock speed as compared to RNI.

The design of RNI which we presented in this thesis offers low latency. Simulation results show that the best case flit latency for both distributed and source routing RNI is 4 clock cycles. Implementation results of this RNI design on Altera Cyclone II FPGA show that it consumes only 2% of the available logic resources.

6.2 Limitations

RNI and protocol used can handle NoC system of limited sizes for both distributed and source routing. In distributed routing this limitation is because of the address bits to be stored in the packet whereas in source routing, this limitation is due to the path overhead caused by storing routing information in the head flit. This limitation can be easily removed by using first two flits for header in the packet.

Another limitation of the thesis work is the single packet buffering i.e. only one packet can be stored in the RNI buffers at a time for both distributed and source routing. This can cause reduction in the rate of packets which can be injected by the core in the network.
This RNI design can only work with the cores which have parallel I/O (PIO) port based interface. RNI design must be modified to interface it to other types of interfaces provided by cores.

### 6.3 Future Work

The results of our thesis provide us motivation to improve RNI design in the future by adding extra features. One such improvement can be storing multiple packets in the buffer if there is sufficient space for them. This will significantly improve packet injection rate.

Currently we are testing the functionality of our RNI design by connecting Nios II core on one side and a dummy router on the other. Thus the core sends data to the router through RNI and the router sends it back to the core through RNI. Prototyping a complete NoC and testing the functionality of our RNI design in complete NoC will be very interesting.

Currently we are using the same clock for both RNI and Nios core, if commercial off-the-shell (COTS) cores are used with our RNI design then the clock of RNI and core used will be different. If different clocks are used then some issues may be encountered such as synchronization. Resolving these issues to make our RNI design usable with COTS with different clock frequencies can be an interesting future work.
7 References


References


Appendix 1

C Code for packet generation

#include<stdio.h>
#include<system.h>
#include<io.h>

int main()
{
    volatile int RTR=0;
    const int Wr_1 = 1;
    const int Wr_0 = 0;
    int pkt[16]={0xDB0,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xAA,0xBB,0xCC,0xDD,
                 0xEE,0xFF};  // A Complete Packet
    int index = 0,i,j,loop;
    i=j=loop=0;
    IOWR(WR_BASE,0,Wr_0);  // Output port (Write signal)
    IOWR(DATA_BASE,0,0);  // Output port (Data)
    while(loop < 16)
    {
        while(index <= loop)
        {
            RTR=IORD(RTR_BASE,0);
            while(RTR != 1)
            {
                IOWR(WR_BASE,0,Wr_0);
                RTR=IORD(RTR_BASE,0);     // Input port (RTR signal)
            }
            printf("RTR=%d\n",RTR);
            if (index == 0) {
        if (loop == 0)
            IOWR(DATA_BASE,0,0X180); // Header with single flit
        else if (loop == 1)
            IOWR(DATA_BASE,0,0X580); // Header with two flits
        else if (loop == 2)
            IOWR(DATA_BASE,0,0X980);
        else if (loop == 3)
            IOWR(DATA_BASE,0,0xD80);
        else if (loop == 4)
            IOWR(DATA_BASE,0,0x190);
        else if (loop == 5)
            IOWR(DATA_BASE,0,0x990);
        else if (loop == 6)
            IOWR(DATA_BASE,0,0x590);
        else if (loop == 7)
            IOWR(DATA_BASE,0,0x190);
        else if (loop == 8)
            IOWR(DATA_BASE,0,0x990);
        else if (loop == 9)
            IOWR(DATA_BASE,0,0x590);
        else if (loop == 10)
            IOWR(DATA_BASE,0,0x190);
        else if (loop == 11)
            IOWR(DATA_BASE,0,0x990);
        else if (loop == 12)
            IOWR(DATA_BASE,0,0x180);
        }
    }
}
else if (loop == 13)
    IOWR(DATA_BASE,0,0X5B0);
else if (loop == 14)
    IOWR(DATA_BASE,0,0X9B0);
else if (loop == 15)
    IOWR(DATA_BASE,0,0xDB0);  // Header with 16 flits
    }
else {
    IOWR(DATA_BASE,0,pkt[index]);
    }
   IOWR(WR_BASE,0,Wr_1);
    printf("index=%d\n",index);
    index++;
    RTR=IORD(RTR_BASE,0);
    while(RTR != 0)
    {
    IOWR(WR_BASE,0,Wr_1);
    RTR=IORD(RTR_BASE,0);
    }
    printf("RTR=%d\n",RTR);
    IOWR(WR_BASE,0,Wr_0);
    }
    index=0;
    loop++;
C Code for packet receiving

#include<stdio.h>
#include<system.h>
#include<i.o.h>

int main()
{
    int Wr,Packet;
    Wr=Packet= 0;
    while(1)
    {
        IOWR(RTR_BASE,0,1); // Output port (RTR signal)
        Wr=IORD(WR_BASE,0); // Input port (Write signal)
        while(Wr != 1)
        {
            IOWR(RTR_BASE,0,1);
            Wr=IORD(WR_BASE,0);
        }
        Packet=IORD(DATA_BASE,0);  // Input port (Data)
        IOWR(RTR_BASE,0,0);  // Input port (Data)
        Wr=IORD(WR_BASE,0);
        while(Wr != 0)
        {
            IOWR(RTR_BASE,0,0);
            Wr=IORD(WR_BASE,0);
        }
    }
}