Design and Simulation of go-back-n scheme based protocols in Networks on Chip

A research thesis submitted to the Institute of Management Sciences for the degree of Bachelors in Computer Science

Shahzad Haider

Department of Computer Science
Institute of Management Sciences
Peshawar, Pakistan

February, 2010

Design and Simulation of go-back-n scheme based protocols in Networks on Chip

This research thesis is submitted to the Department of Computer Science, Institute of Management Sciences as a partial requirement for the fullfilment of Bachelor degree in Computer Science.

Research Scholar:	____________________
	Shahzad Haider
Supervisor:	____________________
	Dr. Muhammad Ali
Head of Computer Science Department:	____________________
	Dr. Muhammad Ali

My achievement is dedicated to my parents and my uncle, who were always there to help me in achieving my goals.

Acknowledgements

First, I want to thank **Allah the Almighty** for giving me the strength and courage to accomplish my goal. Then, I offer my gratitude to the supervisor of my project, **Dr. Muhammad Ali**, for my guideline to work on state-of-the art technology.

I am thankful to **Sir. Adnan Yousaf** for providing me guideline throughout my bachelor studies. This achievement was a dream for me without his support. I pay regards to **Dr. Masoom Alam** and other teachers from **Security Engineering Research Group** for providing me place to work on my research.

I would want to thank those who made this report writing possible for me, by being around and providing me with their support. My friends **Mudassir Ali Khan**, **Emad Saifullah Khan** and **Abbas-ul-Hassan**who assisted me in completing this research; and to my friend **Mariya Zahid**, who always encouraged me in writing this thesis.

Indeed, I would like to thank all the teachers, classmates and friends who helped me in overall research work.

- Shahzad Haider,

fodwarning@gmail.com

Table of Contents

List of Figures

Abstract

The evolution of Integrated Circuits started in the mid of the 20th century but it took a very good pace with passage of time. According to Moore's Law the transistors on chip doubles every 18 months but current technology is not keeping the pace. It is because of reliability, which is a bottleneck in achieving billion transistor chips. However, it has been observed that the scalability of chips is posing grave problems for the current interconnect architecture which is unable to cope with the growing number of components on a chip.

Networks-on-Chip is the implementation of TCP/IP protocols suitable for chip which decouples computation from interconnection. It seems to be perfect solution for achieving billion transistor chips, because of the packet based system introduced on-chip. In order to make communication reliable end-to-end fault tolerant protocols have been devised. The focus of this thesis lies on the designing and simulation of one of the protocols based on go-back-n scheme. The design and Simulation of the protocol is carried out in Modelsim EDA design tool.

Chapter 1

Introduction

In mid of 20th century the era of Integrated Circuits emerged at Texas Instrument and Fairchild Semiconductor for experimental basis. This IC was basically the integration of large number of tiny transistors into a single chip. This technology was adopted for military purposes and later on pushed out to commercial market. ICs became focus of all large and medium manufacturers which boosted this technology. It passes through the generations of SSI (Small Scale Integration), MSI (Medium Scale Integration) and LSI (Large Scale Integration). The final step in the development of ICs is VLSI (Very Large Scale Integration) which started from 1980s till 2009. It started with hundreds and thousands of transistors and led to billions of transistors on a single chip. Researchers felt need of a chip, where all the components needed for a system are combined on a single chip known as Systems-on-Chip (SoC). It is a chip where all the heterogeneous or homogeneous components are stitched on a single chip which graves some problems of reliability. Regardless of this problem, the benefit of SoC is that it consumes very less power because of availability of all the required components on a single chip. The growth in operating frequency and transistor density will make energy dissipation and heat extraction a serious issue for researchers. The decrease in the voltage on chip will affect the signal integrity. Dynamic Power Management (DPM) is the solution for these rising issues. Propagation delay resulting largely exceeding the clock period will be due to the connections between on-chip units. Fault tolerant mechanisms should be achieved to prevent hard and soft errors (permanent and transient faults respectively). In heterogeneous units in a NoCs, different units will require different voltages/frequencies which become extremely hard to achieve. In these technologies it will be harder to achieve on chip error-free communication because of reduced signal swings, increased cross talk, electromagnetic interference (EMI) by external sources, probability of synchronization failures, soft errors due to collision of thermal neutrons. "System level reliability is the probability that the system will operate correctly at time, t, as a function of time. The expected value of the reliability function is the mean time to failure (MTTF)." Reliability is affected by the downscaling of chip which results in interconnect and device failure. These SoCs had to be designed to heal itself against hard and soft errors (permanent and transient faults respectively). NoCs is designed to deal with the effect of variability because networking technology is layered and error detection, containment and correction can be done at various layers. The silicon area cost of a NoC depends directly on the type of topology. The area of routers and node interface (NI) which can be computed by gate level synthesis and layout tools sums up to the area cost of links. The dynamic (power dissipation) cost of a NoC depends on the number of NoC components that are active, independently of data in the network (static power, clock power, etc.) and the power dissipation due to data in the NoC itself. The latter depends on the NI dissipation, the number of routers data has to traverse to arrive at the destination and the dissipation in the links that are used. Topologies with a short routes and short links score well. The performance of a NoC depends on its topology. Topologies with a high performance measure (e.g., large bisection bandwidth, small diameter, etc.) tend to have a high area cost (number of routers and links). Cost and performance must be traded off against one another. If much of the application is known, the NoC topology can be tailored and optimized, for example, to avoid hotspots (local overloading of the NoC). Moreover, optimization of network parameters, such as link bandwidth and latency, switch configuration, buffering and channel surface, as well as NI design is key to an effective design[1].

1.1 Motivation

The step in taking towards NoC brings along strong benefits, which includes the decoupling of communication from computation and introducing the packet system (skipping the traditional bus system). Adopting NoC helps in regaining the reliability by applying fault tolerant protocols. By applying fault tolerant protocols, error correcting codes and introducing packet system in NoC will solve some of the issues emerged in achieving a billion transistor chip. The above stated facts indicate the improvement due to the design of the error control protocols in NoCs. The primary aim of this thesis is design and simulation of these fault tolerant protocols that are well suited for a diverse NoC. Finally, the solutions should be reusable and easily implementable in the current state-of-the-art of VLSI hardware.

1.2 Thesis Contribution

This thesis is aimed towards the design and simulation of End-to-End Error Control Protocols which can be used to improve the reliability of data transmission in NoCs. The first part of the thesis covers the design of these protocols and its supporting modules. These protocols are designed in Verilog using Modelsim EDA tool. All the supporting modules are also designed which works in collaboration of these protocols. The second part of the thesis shows and explain the Simulation of these protocols and supporting modules. Simulation is performed in Modelsim EDA tool; these simulations are performed by making the previously designed protocols as its ground. These protocols are responsible for the reliable delivery of packets in an NoC environment. The coding of all modules are self explanatory and can be edited for further improvements and integrations.

1.3 Thesis Organization

Chapter 2, the Literature Review describes in detail the protocols which are devised to improve the reliability of on-chip networks communication. Chapter 3 explains in detail the NoC model along with the components and schemes used by NoC. In-short this chapter provides a glimpse and a basic idea of a 2D-mesh on-chip network. In Chapter 4, the plan of taking the research is described. All the supporting modules are described. STD is converted to FSM mealy machine. Chapter 5 describes the design and simulation of supporting modules included in an NoC model. Chapter 6 explains in detail the design and simulation of actual sender and receiver protocol. Chapter 7, Concludes the research and shows research to be carried out in future.

Chapter 2

Literature Review

Few decades back the evolution of electronics started from Integrated Circuits(ICs). The evolution started from Small Scale Integration(SSI) and now it have reached till Very Large Scale Integration(VLSI) and Ultra Large Scale Integration(VLSI). These advancements in chip technology have the ability to accommodate billions of transistors on a single chip. However these billions of transistors are not used efficiently and current technology pace is slower then the prediction of Sir Moore in Moore's Law.

Researchers moved towards Systems-on-Chip in order to bring all the cores on a single chip. Uniting different cores on a single chip solves some of the issues but as technology is advancing, give birth to further problems and revival of some old problems also. Reliability was again a hot issue for SoC designer. In order to solve the problem Networks-on-Chip was introduced; whose basic motive was to decouple communication from computing. In this way researchers can concentrate on the communication paradigm of the chip and can solve the issue to faults. In NoC an altered form of TCP/IP Protocol(suitable for hardware) was adopted and ported to chip, in which all the cores on chip are treated as hosts on a Local Area Network(LAN). Why is this idea adopted? because we have a very good implementation of reliable communication around the world, The Internet. By porting the idea to chip may reduce these problems and increase reliability on-chip. Thats why NoC are some times also called on-chip networks.

There are some problems which resists the chip from utilizing billions transis-

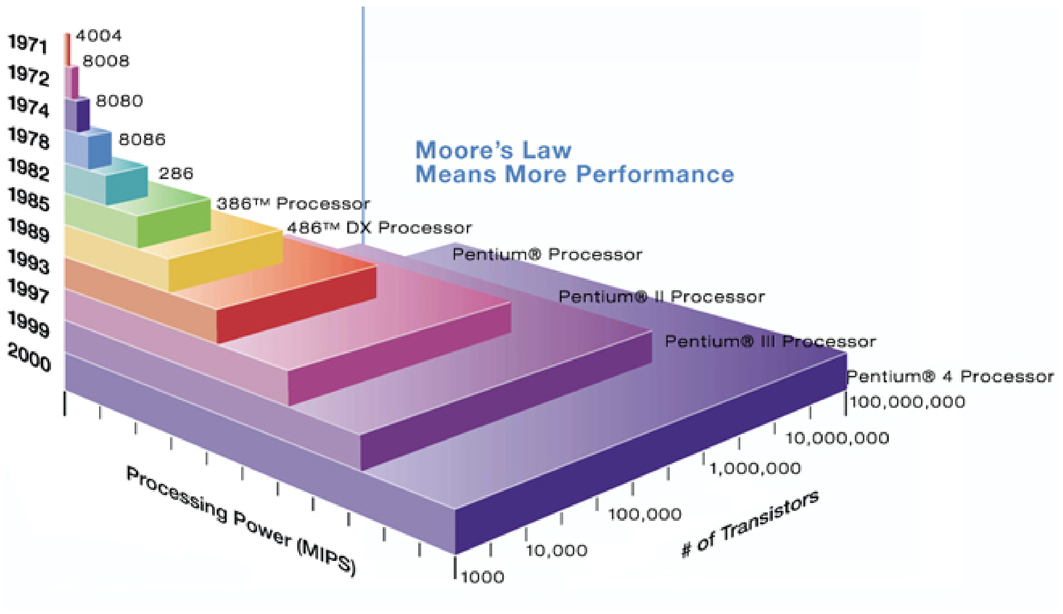


Figure 2.1: Moore's Law

tors on a single chip. Deep inside the silicon chip arose some problems which have been in focus of scientists and engineers for last few years. These problems include different issues in which Reliability is also considered. These billion transistor single chips are not reliable due to faults which are categorized as Transient Faults(or Soft error) and Permanent Faults(or Hard Error)[2].

2.1 Fault Tolerance

As mentioned above the problem faced by designers and researchers in achieving billions transistor chip are faults occurred due to hard and soft errors. Soft errors (or Transient Errors) are those errors which corrupt the packet and result in loss or misrouting of that packet these errors are not permanent and can be fixed. Hard errors(or Permanent Errors) are those errors which is nearly impossible to fix, in these errors a router may malfunction resulting in the blockage of packets. M. Ali designed protocols to fix these problems. The focus of this thesis is to design and simulate the protocols for Transient Faults. The protocols devised for these problems are End-to-End Error Control Protocol using Go-back-n and End-to-End Error Control Protocol using Selective

Repeat Protocol[3,4]. End-to-End Error Control Protocol will be covered in this thesis.

2.2 Protocol based on go-back-n scheme

This protocol is implemented in end-to-end systems where intermediate routers are relieved of storing and checking for alteration of packet. The aim of this protocol is to resend lossy or corrupt packets. A single bit change in packet header or payload may change the whole scenario; it may misroute or corrupt the packet. In either of the cases the packet will be retransmitted as the protocol is based on the acknowledgment.

Packets are of two types; control packet and data packets. The idea here to communicate using packets in NoC is that Local Processing Element(LPE) have a message which is divided into packets (equal size of packets). These packets are then transmitted over the on-chip network. A general packet have two parts; Packet header and Payload. Header carries control information while payload carries the actual data. In an NoC control packet, there is no payload but only header; while a data packet have header as well as payload. The packet header carries information about the sequence number (pno), type of packet (ptype), source address (SID), destination address (RID) and Time since the packet was created (TC).

In a typical NoC, routers are connected to its adjacent routers in a 2D mesh topology. Router are known as nodes in on-chip networks. Each router is connected to its own Local Processing Element(LPE). LPE is totally concerned with computation thats why it will not be focus of this thesis and Router is concerned with the communication. This router have four input and output ports for transferring data packets. Four ports of input and four of output are connected to router in its geographical directions I.e., East, West, North and South; while the other input and output port is connected to LPE. An arbiter inside the router is responsible for forwarding the packets to its desired destination. Packet received at any input port can be forwarded to any output of the router. This uses relative XY switching policy in-order to switch packets towards its destination. The destination address is extracted from the packet by arbiter; this address is in the form of x,y format. X shows the location of the router in row and Y shows the location of router in column. When the address is extracted the arbiter forward the packet to desired output port through which the packet can reach its destination. When the address is matched at any router, the packet is forwarded to LPE of that router.

2.2.1 Sender Protocol

This protocol is End-to-End Error Control using go-back-n scheme; so the packets will be checked for inconsistencies on end systems or on destinations. In order to send packet from source to destination the sender of the source router will communicate with the receiver of the destination router. In order to send packets the sender protocol will have to build a connection with receiver of the destination router; To do so, the sender will build a handshaking communication with receiver. When the sender has data to send it will send a control packet of REQEST to receiver, which includes Request(SID, pno). Here, SID is the source address and pno is the sequence number of set of packets. This packet tells the receiver that sender have some packets to transmit, whose address is SID and packet sequence is pno which is to be sent. The receiver on receiving this request packet will respond with and ACKNOWLEDGEMENT packet telling sender that receiver is ready to receive these packets. If receiver is busy, it simply ignores the request. The sender on receiving the ACK packet starts the process of sending packets but if the receiver ignores the request, sender resends after a timeout. When all the packets are send the sender simply sends a TERMINATE packet and the connection is terminated.

In this protocol, every packed is not acknowledged but a sequence of packets are send and NACK are generated if a packet was not received successfully; ACK is used to inform that the set of packets are received successfully and send further packets if available. Initially, the sender is in WAIT state; because there are no packets to be sent. When buffer have packets to be sent, the sender automatically switches to SEND state, initialize buffers and set X=0, P=0. After successful establishment of handshaking communication, the sender starts sending all the packets available in buffer in its sequence. During this process if a NACK was received, the sequence of sending packets is discarded and retransmission of packets starts from the sequence of NACK

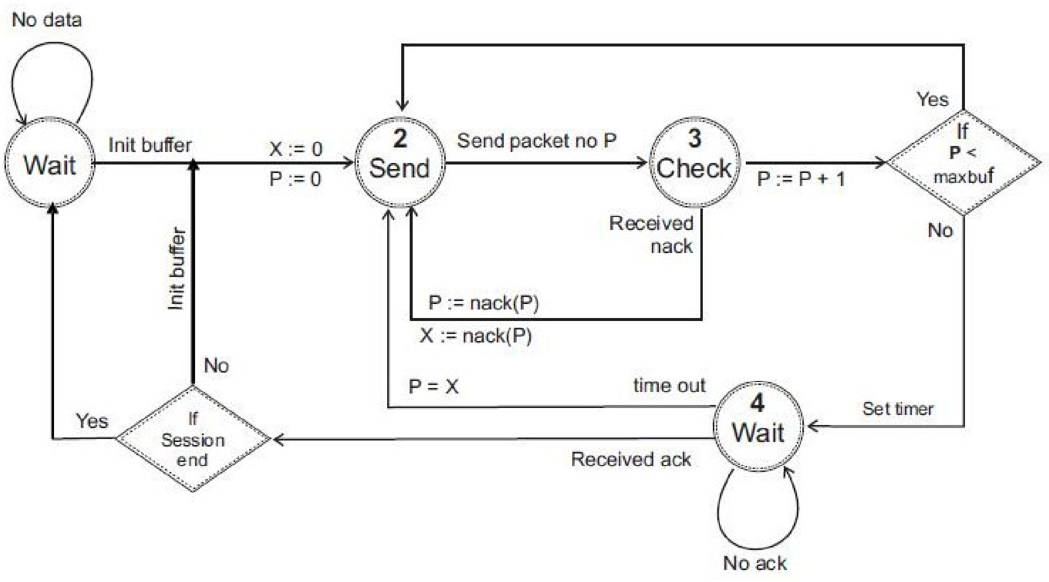


Figure 2.2: Sender Protocol

received. When all the packets are received by receiver, it generate an ACK and ask sender for the next set of packets pno+1. Thus a NACK is very useful as it saves the sender from retransmitting all the packets again and enables faster error recover. In order to detect errors in control packets and data packets, a CRM decoder is implemented which drops a faulty control/data packets. If sender has no more data to send, it sends a TERMINATE signal to inform the receiver that the session has ended. The receiver responds with an ACK(pno+1) where pno is the sequence number of the last received packet. This ACK is an acknowledgment to the sender that the receiver has terminated the communication.

2.2.2 Receiver Protocol

The receiver protocol is in LISTEN state and waits for sender to start communication. When a connection is established the receiver switches to WAIT state and it is now ready to receive packets. As it receives first packet, it maintain the sequence number and start counting the packets till the buffer is full. On receiving the packets, they are checked using CRC checker. When all the packets are received the receiver sends an ACK packet to sender in order to inform that all the packets are successfully received and switches to

WAIT state in order to receive further packets if there are any. But if a faulty packets were found or packets were lost, receiver discards the corresponding packets and sends a NACK requesting the lossy/corrupt packet. Unless the required packet of the sequence is not received the receiver will not accept other packets. When all the packets are received and sender has no further packets to be sent, it will send a TERMINATE packet. Receiver will terminate the communication as there are no further packets to be received.

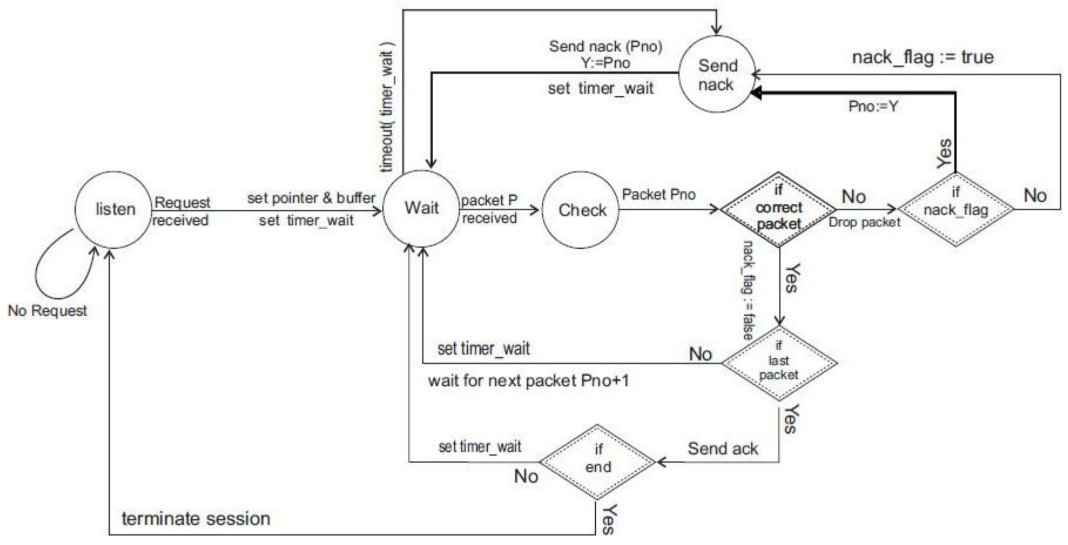


Figure 2.3: Receiver Protocol

Chapter 3

Generic NoC Model

NoC was introduced due to the increasing demand of electronic transistors on a single chip. In order to achieve such amount of transistors efficiently on a single chip was very difficult. Researchers approached apart from electronics field and looked around for a suitable solution. As we know, The Internet is the most reliable communication technology we have ever seen; so porting an altered form of The Internet's protocol TCP/IP to chip was not a bad idea. The biggest advantage of doing so is that communication is decoupled from computation and reliable delivery of data transmission. This term is normally called Networks-on-chip(NoC); while sometimes called on-chip networks.

3.1 NoC Model

Following are the main components which belongs to a generic NoC model:

3.1.1 Topology

Networks-on-Chip have typical topologies just like Internet. These topologies include bus, star and mesh. Our focus will be on mesh topologies because it has been proved successful in case of NoC. In mesh topology every router is connected to its adjacent/neighbor router. In this topology if a router or a link fails, packets are sent on alternate paths. Every node is connected to its adjacent or nearly node using dedicated uni-directional or bi-directional bus, which prevents data collision. Routers are connected in mXn dimension which make it 2 dimensional mesh topology. At two dimensional, mesh have uniformly short wires allowing high speed operations without repeaters, hence allowing them to exploit physical locality between communication nodes. A mesh network have multiple paths to route packets, therefore in case a path is dogged alternate path may be used.

3.1.2 Router

A router is a device, which routes data/packets in a network; while in NoC router have same functionality. It depends on scheme of error control whether it is link-to-link or end-to-end. In link level, errors are checked at every router, while in system level it want be checked at every router. A typical router in an NoC have sender and receive protocol implementation at every geographical direction, CRC module, switch and this router is connected to Local Processing Element(LPE). This LPE is totally concerned with computation aspects of an NOC. Between the router and LPE is an adapter called Network Interface Adapter(NIA). So the basic purpose of router is to switch and route packets. Both transient and permanent fault tolerance protocols are implemented inside a router.

3.1.3 Network Interface Adapter

Communication is decoupled from computation in Network Interface Adapter(NIA). NIA defines boundaries between computation and interconnection. Core Interface is implemented at Core Side and Network Interface is implemented at network side. The basic function of the NIA is to provide high level communication services to the core by utilizing primitive services provided by the network hardware. One of the most commonly used socket is the Open Core Protocol(OCP). Other proposed standards include Virtual Component Interface (VCI) used in SPIN and Proteo NoC, AMBA Advanced Extensible Interface(AXI) in, and Device Transaction Level(DTL) protocol implemented in.

3.2 Quality of Service

There are two types of communications; connection less and connected oriented. In connection less communication there is no connection established between sender and receiver. While in connection oriented communication; first, a connection is established then data is exchanged between sender and receiver. Circuit switched network is an example of connection oriented communication. Packet switched can be used in both the cases. There are two terms named "guaranteed" and "bast effort". Guaranteed is the guarantee of the reliable end-to-end transfer of data while best effort is connectionless communications which try to provide best effort services. SoC needs both types of communication which is very costly. Instead a combined best effort and guaranteed services architecture is more feasible for NOC, considering the efficient utilization of resources on a chip.

3.2.1 Routing

Routing is the transferring of data in a network, typically following the shortest possible path. Routing is an essential component of interconnections networks. These may be an arbitrary number of paths between source and destination. Routing algorithm is designed to choose the most profitable path in terms of distance between the two nodes. In mesh based NOCS, the distance between each node is equal and fixed. Hence routing in NOC corresponds to the minimum number of hops between sender and receiver.

3.2.2 Flow Control

Flow control is also called switching policy. This switching policy performs the forwarding of packets from one input to the output port of a switch. There are two types of switches used, buffer less switch and buffered switches. In buffer less, packets are not stored in switch as switches don't have any kind have storage. These packets are forwarded immediately, but if two packets are going for a single/same port then one packet is simply misrouted, which is called deflection routing. In buffered switches/flow control, each on chip switch maintains buffer to temporarily store packets in case of contentions. Three types of buffered flow control are widely used, Store and Forwarded Switching(SAF), Virtual Cut Through(VCT) switching and wormhole switching.

3.2.3 Reliability

As today chip integrates millions of transistor, which surely notifies of reliability issue, as number of components on chip is increasing, the complexity of the components interconnection is also increasing. In this case two types of errors/faults are raised; Transient Faults and Permanent Faults. Transient Faults are fixed by governing protocols by implementing retransmission policy of loss/corrupt packets. Permanent Faults are corrected by routing schemes/protocols. Error control or reliability is implemented at either link level or at the end-to-end level. Packets are checked for errors at every node in link level, while errors are checked at cores or at system in end-to-end level. chapterResearch Plan

3.3 Our NoC Model

The model we are proposing is a typical NoC model which is widely used by researchers. The topology of the model will be based on 2D mesh, routers will be connected to its adjacent routers and LPE will be connected to each router. The router will have four sender and receiver modules connected to it in geographical directions. Arbiter is placed inside the router and CRC inside router will be checking for inconsistencies in packets[1].

3.4 Supporting Modules

Following modules are designed in limited functionality to obtain compulsory tasks. These are the supporting modules which helps in creating a typical NoC environment. These modules are used with each and every protocol.

3.4.1 Local Processing Element (LPE)

Simple Local Processing Element(LPE) modules are designed which acts as dummies. We are not supposed to decide its functionality because it can be anything for storage to processing. IP cores can be replaced over LPE for obtaining different kinds of tasks. Here in this research the functionality of LPE will only be to send/receive message.

3.4.2 Network Interface Adapter (NIA)

The sender buffers are implemented in Network Interface Adapter(NIA). Communication is decoupled from computation in NIA. It defines boundaries between computing and interconnections. Core Interface is implemented at Core Side and Network Interface is implemented at network side. The basic function of the NIA is to provide high level communication services to the core by utilizing primitive services provided by the network hardware.

3.4.3 Arbiter

Flow control is also called switching policy. This switching policy performs the forwarding of packets from one input to the output port of a switch. There are two types of switches used, bufferless switch and buffered switches. In bufferless, packets are not stored in switch as switches don't have any kind have storage. These packets are forwarded immediately, but if two packets are going for a single/same port then one packet is simply misrouted, which is called deflection routing. In buffered switches/flow control, each on chip switch maintains buffer to temporarily store packets in case of contentions. Three types of buffered flow control are widely used, Store and Forwarded Switching(SAF), Virtual Cut Through(VCT) switching and wormhole switching.

3.4.4 Router

Routing is the transferring of data in a network, typically following the shortest possible path. Routing is an essential component of interconnections networks. These may be an arbitrary number of paths between source and destination. Routing algorithm is designed to choose the most profitable path in terms of distance between the two nodes. In mesh based NOCS, the distance between each node is equal and fixed. Hence routing in NOC corresponds to the minimum number of hops between sender and receiver.

3.4.5 Cyclic Redundancy Check (CRC)

CRC modules are used to check for inconsistencies in packets occurred during transmission. This module is not focused here due to some of the limitations. But we assume that packets are always transmitted correctly without any errors. In future, as CRC modules are ready to be integrated, these protocols will be filled up with CRC modules.

3.4.6 Timer

We have designed counters which also acts as timers. They are used to make the machine wait for a specific time and take action after an overflow occurs. We are using a constant time frame in these timer because we are in middle of the research and don't know how many mxn mesh we will be creating for our NoC design. As this figure is confirmed, the timers will be changed and variable timers will be integrated in order to obtain variable waiting time for different scenarios.

3.5 STD to FSM

Both of the protocols are elaborated in along with its pseudo code and State Transition Diagram(STD). In order to convert these protocols into Hardware Description Language(HDL), we will have to convert it first to Finite State

Machine(FSM). This FSM will then be easy to coded in any programming language. Finite State Machine is a model of behavior including a finite number of state transitions between those states, and processing. It is similar to a "flow graph" where we can monitor the way in which the logic runs when certain conditions are met. There are two types of FSM; moore machine and mealy machine. The Moore FSM uses only entry actions, i.e., output depends only on the state. While, Mealy FSM uses only input actions, i.e., output depends on input and state. The use of a Mealy FSM leads often to a reduction of the number of states. After studying the protocols, it is obvious that it should be designed using Mealy Machine because the outputs are dependent on inputs and states.

Chapter 4

Supporting Modules

Here the designing phase will be covered along with simulation of each module. The EDA tool used to create, maintain project files and design modules is Modelsim SE 5.7f. Modelsim is a popular EDA tool used to code HDL languages and provide logic simulation or waveform facility. It can be used to design digital circuits for FPGA and ASIC. Verilog has been used as language of coding. Simulation is also performed inside Modelsim SE.

4.1 Design

These supporting modules are designed once and used with protocols again and again just by replicating it. These includes, Arbiter, Network Interface and Counter.

4.1.1 Arbiter

Arbiter is implemented inside router, whose task is to read destination address from each packet and forward it towards its destination by switching it to specific output port of the router (using geographical direction). Following is the code of designing and its simulation.

Design Code

```
module arbiter( clk,
rst_n,
east_in,
west_in,
north_in,
south_in,
ni_in,
send_st,
east_out,
west_out,
north_out,
south_out,
ni_out,
s_east,
s_west,
s_north,
s_south
);

//parameterization area
parameter MYXLOC = 3'b000;
parameter MYYLOC = 3'b000;
parameter DATA_WIDTH = 32;
//parameter XADDS = 11:9;
//parameter YADDS = 8:6;

//inputs and outputs
input clk, rst_n;
input [DATA_WIDTH-1:0] east_in, west_in,
north_in, south_in, ni_in;
input [5:0] send_st;

output [DATA_WIDTH-1:0] east_out, west_out,
north_out, south_out, ni_out;
```

```
output s_east, s_west, s_north, s_south;
reg s_east, s_west, s_north, s_south;
reg [DATA_WIDTH-1:0] east_out, west_out,
north_out, south_out, ni_out;

always @(negedge rst_n)
begin
east_out <= 0;
west_out <= 0;
north_out <= 0;
south_out <= 0;
ni_out <= 0;
end

//functionality of xy routing
always @(east_in)
begin
if(east_in[11:9] < MYXLOC)
west_out <= east_in;
else if(east_in[11:9] > MYXLOC)
east_out <= east_in;
else if(east_in[8:6] < MYYLOC)
north_out <= east_in;
else if(east_in[8:6] > MYYLOC)
south_out <= east_in;
end

always @(west_in)
begin
if(west_in[11:9] < MYXLOC)
west_out <= west_in;
else if(west_in[11:9] > MYXLOC)
east_out <= west_in;
else if(west_in[8:6] < MYYLOC)
north_out <= west_in;
else if(west_in[8:6] > MYYLOC)
```

```
south_out <= west_in;
end

always @(north_in)
begin
if(north_in[11:9] < MYXLOC)
west_out <= north_in;
else if(north_in[11:9] > MYXLOC)
east_out <= north_in;
else if(north_in[8:6] < MYYLOC)
north_out <= north_in;
else if(north_in[8:6] > MYYLOC)
south_out <= north_in;
end

always @(south_in)
begin
if(south_in[11:9] < MYXLOC)
west_out <= south_in;
else if(south_in[11:9] > MYXLOC)
east_out <= south_in;
else if(south_in[8:6] < MYYLOC)
north_out <= south_in;
else if(south_in[8:6] > MYYLOC)
south_out <= south_in;
end

always @(send_st)
begin
if(send_st[5:3] < MYXLOC)
begin
s_west <= 1;
s_east <= 0;
s_north <= 0;
s_south <= 0;
end
```

```
else if(send_st[5:3] > MYXLOC)
begin
s_west <= 0;
s_east <= 1;
s_north <= 0;
s_south <= 0;
end
else if(send_st[2:0] < MYYLOC)
begin
s_west <= 0;
s_east <= 0;
s_north <= 1;
s_south <= 0;
end
else if(send_st[2:0] > MYYLOC)
begin
s_west <= 0;
s_east <= 0;
s_north <= 0;
s_south <= 1;
end
end

endmodule
```

Simulation Code

```
module arbiter_tb();

reg clk, rst_n;
reg [5:0] send_st;
reg [31:0] east_in, west_in, north_in, south_in, ni_in;
wire [31:0] east_out, west_out, north_out, south_out, ni_out;
wire s_east, s_west, s_north, s_south;
```

```
always #5 clk = ~clk;

initial
begin
clk = 0;
rst_n = 1;
#2 rst_n = 0;
#2 rst_n = 1;

#2 send_st <= 6'b001000;
#2 east_in <= 32'b01_00000001_0001_000000_000001_111111;

#100 $stop;
end

arbiter arbiter_1( .clk(clk),
.rst_n(rst_n),
.east_in(east_in),
.west_in(west_in),
.north_in(north_in),
.south_in(south_in),
.ni_in(ni_in),
.send_st(send_st),
.east_out(east_out),
.west_out(west_out),
.north_out(north_out),
.south_out(south_out),
.ni_out(ni_out),
.s_east(),
.s_west(),
.s_north(),
.s_south()
);

endmodule
```

Simulation Results

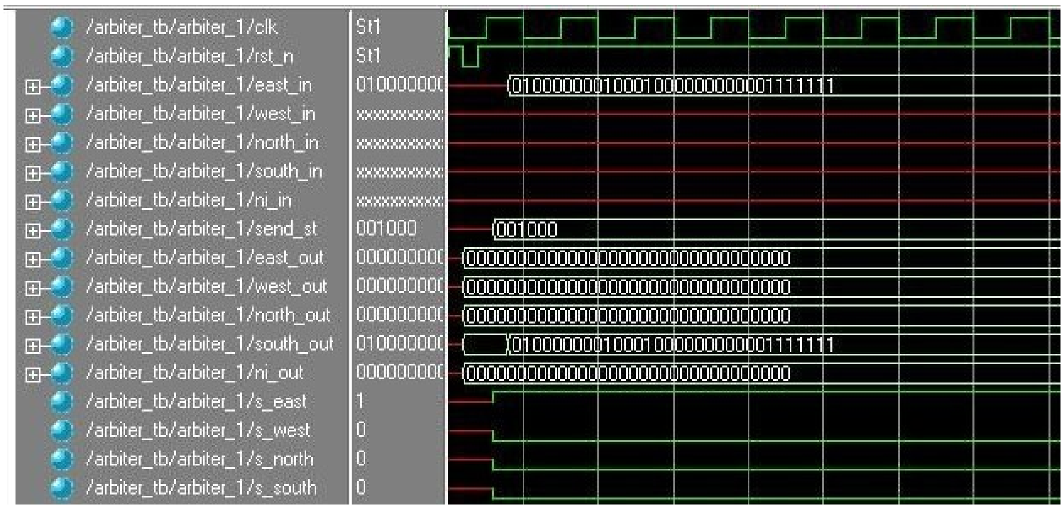

/arbiter_tb/arbiter_1/clk St1
/arbiter_tb/arbiter_1/rst_n St1
/arbiter_tb/arbiter_1/east_in 01000000C 01000000010001000000000001111111
/arbiter_tb/arbiter_1/west_in xxxxxxxxxx
/arbiter_tb/arbiter_1/north_in xxxxxxxxxx
/arbiter_tb/arbiter_1/south_in xxxxxxxxxx
/arbiter_tb/arbiter_1/ni_in xxxxxxxxxx
/arbiter_tb/arbiter_1/send_st 001000 001000
/arbiter_tb/arbiter_1/east_out 000000000 00000000000000000000000000000000
/arbiter_tb/arbiter_1/west_out 000000000 00000000000000000000000000000000
/arbiter_tb/arbiter_1/north_out 000000000 00000000000000000000000000000000
/arbiter_tb/arbiter_1/south_out 01000000C 01000000010001000000000001111111
/arbiter_tb/arbiter_1/ni_out 000000000 00000000000000000000000000000000
/arbiter_tb/arbiter_1/s_east 1
/arbiter_tb/arbiter_1/s_west 0
/arbiter_tb/arbiter_1/s_north 0
/arbiter_tb/arbiter_1/s_south 0

Figure 4.1: Arbiter Simulation

In the design code of arbiter, the arbiter is monitoring every input of its module. If any port receives any kind of packet, its destination address is extracted and stored in a temporary register. This address have two parts, x address and y address; both of the addresses are stored in separate registers.The address is compared with the address of the arbiter; the address of arbiter is stored in MYXLOC and MYYLOC. If the x address is lesser than the x address of the arbiter it is switched to the west (left) output port; but if the x address is greater than the x address of the arbiter it is switched to the east (right) output port. If the y address is lesser than the y address of the arbiter it is switched to the north (top) output port; but if the y address is greater than the y address of the arbiter it is switched to the south (down) output port. The always block of sendst allows the packets to move from LPE to required output port to reach its destination. The simulation code of the arbiter simulates the functionality of the arbiter and creates a waveform or logic simulation of the design behavior. It checks two scenarios; first, arbiter has to act as an intermediate node (just switch packet from one input port to another output port) and second, it acts as source to send packets. Here is the simulation of arbiter module.

4.1.2 Network Interface

Network Interface acts as middle-man between computation and interconnection in an NoC. It takes message from LPE, convert it into equal size of packets and deliver it to router and vice versa.

Design Code

```
module ni(clk,
rst_n,
pno_in,
w_buff_no,
w_buff_pack,
pack_out,
data_avail_out,
send_st
);

input clk, rst_n;
input [3:0] pno_in, w_buff_no;
input [31:0] w_buff_pack;
output data_avail_out;
output [5:0] send_st;
output [31:0] pack_out;
reg data_avail_out;
reg [5:0] send_st;
reg [31:0] pack_out;
reg [31:0] temp_pack;

reg [31:0] buff [0:9];

always @(negedge rst_n)
begin
send_st <= 6'bxxx_xxx;
data_avail_out <= 0;
send_st <= 0;
```

```
end

always @(pno_in) pack_out = buff[pno_in];

always @(w_buff_no)
begin
buff[w_buff_no] = w_buff_pack;
temp_pack <= buff[0];
send_st <= temp_pack[11:6];
end

endmodule
```

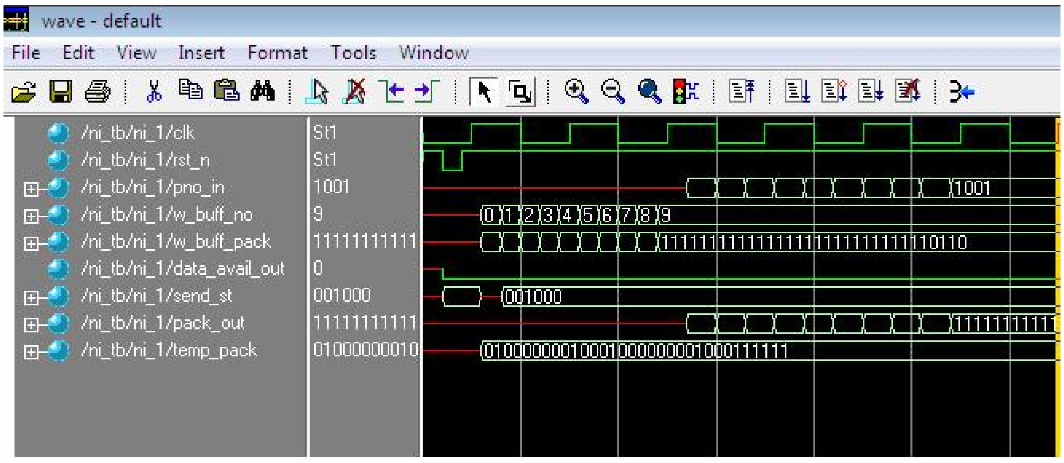
Simulation Code

```
module ni_tb();

reg clk, rst_n;
reg [3:0] pno_in, w_buff_no;
reg [31:0] w_buff_pack;
wire data_avail_out;
wire [5:0] send_st;
wire [31:0] pack_out;

always #5 clk = ~clk;

initial
begin
clk = 0;
rst_n = 1;
#2 rst_n = 0;
#2 rst_n = 1;

#2 w_buff_no = 4'b0000;
w_buff_pack = 32'b01_00000001_0001_000000_001000_111111;
```

```
#2 w_buff_no = 4'b0001;
w_buff_pack = 32'b1111111111111111_1111111111111110;

#2 w_buff_no = 4'b0010;
w_buff_pack = 32'b1111111111111111_1111111111111101;

#2 w_buff_no = 4'b0011;
w_buff_pack = 32'b1111111111111111_1111111111111100;

#2 w_buff_no = 4'b0100;
w_buff_pack = 32'b1111111111111111_1111111111111011;

#2 w_buff_no = 4'b0101;
w_buff_pack = 32'b1111111111111111_1111111111111010;

#2 w_buff_no = 4'b0110;
w_buff_pack = 32'b1111111111111111_1111111111111001;

#2 w_buff_no = 4'b0111;
w_buff_pack = 32'b1111111111111111_1111111111111000;

#2 w_buff_no = 4'b1000;
w_buff_pack = 32'b1111111111111111_1111111111110111;

#2 w_buff_no = 4'b1001;
w_buff_pack = 32'b1111111111111111_1111111111110110;

#3 pno_in = 4'b0000;
#3 pno_in = 4'b0001;
#3 pno_in = 4'b0010;
#3 pno_in = 4'b0011;
#3 pno_in = 4'b0100;
#3 pno_in = 4'b0101;
#3 pno_in = 4'b0110;
#3 pno_in = 4'b0111;
#3 pno_in = 4'b1000;
```

```
#3 pno_in = 4'b1001;



#1000 $stop;
end

ni ni_1(
.clk(clk),
.rst_n(rst_n),
.pno_in(pno_in),
.w_buff_no(w_buff_no),
.w_buff_pack(w_buff_pack),
.pack_out(pack_out),
.data_avail_out(data_avail_out),
.send_st(send_st)
);

endmodule
```

Simulation Results



```
/ni_tb/ni_1/clk              St1
/ni_tb/ni_1/rst_n            St1
/ni_tb/ni_1/pno_in           1001
/ni_tb/ni_1/w_buff_no        9
/ni_tb/ni_1/w_buff_pack      11111111111
/ni_tb/ni_1/data_avail_out   0
/ni_tb/ni_1/send_st          001000
/ni_tb/ni_1/pack_out         11111111111
/ni_tb/ni_1/temp_pack        01000000010
```

Figure 4.2: Network Interface Simulation

In this module of NI, a buffer of 10 packets is maintained which is used to store packets. NI communicates with the sender protocols of every geographical direction of the router. NI forward these packets towards the router and router decides where will these packets go. After deciding the arbiter decides which sender will have these packets to send it and meanwhile inform that sender to pick these packets and send them. The simulation maintain the buffer of packets and fill it with packets. Whenever the packets are to be sent, sender always requests for desired packet and NI transfers the required packet to sender module(as sender don't maintain buffer). Following is the simulation of NI.

4.1.3 Counter

The counters are used as timer with a constant value to count, which requires fixed time to complete the process. These counters are used as timers.

Design Code

```
module counter( clk,
rst_n,
en,
cnt_val,
overf
);

input clk, rst_n, en;
output [7:0] cnt_val;
output overf;
reg [7:0] cnt_val;
reg overf;

always @(posedge clk or negedge rst_n)
begin
if(!rst_n)
begin
cnt_val <= 0;
overf <= 0;
```

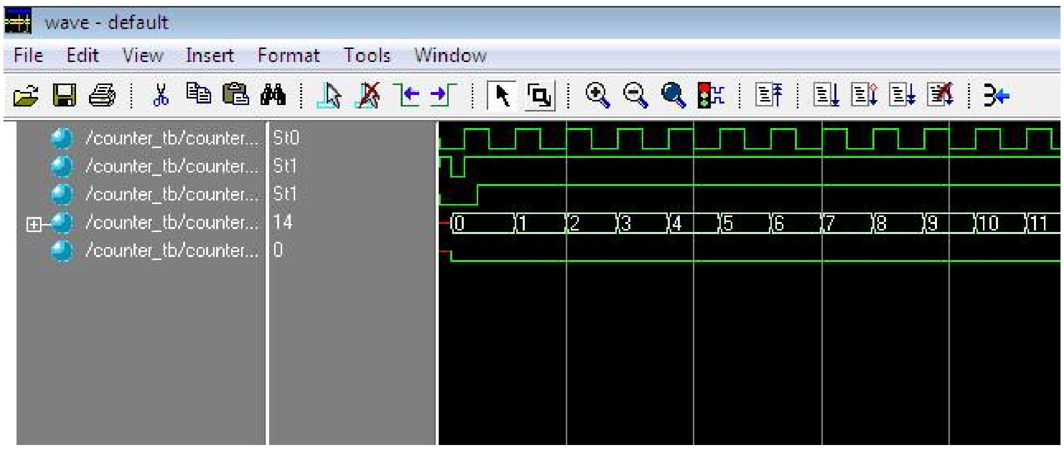
```
end
else
begin if(en)
begin
if(cnt_val == 50)
begin
cnt_val <= 0;
overf <= 1;
end
else
begin
cnt_val <= cnt_val + 1;
overf <= 0;
end
end
else
begin
cnt_val <= cnt_val;
overf <= 0;
end
end
end

endmodule
```

Simulation Code

```
module counter_tb();

reg clk, rst_n, en;
wire [7:0] cnt_val;
wire overf;

always #10 clk = ~clk;
```

```
initial
begin
clk = 0;
en = 0;
rst_n = 1;
#5 rst_n = 0;
#5 rst_n = 1;
#5 en = 1;

#500 $stop;
end

counter counter_mod( .clk(clk),
.rst_n(rst_n),
.en(en),
.cnt_val(cnt_val),
.overf(overf)
);

endmodule
```

Simulation Results



```
/counter_tb/counter...  St0
/counter_tb/counter...  St1
/counter_tb/counter...  St1
/counter_tb/counter...  14    0 1 2 3 4 5 6 7 8 9 10 11
/counter_tb/counter...  0
```

Figure 4.3: Counter Simulation

Here in the design and simulation code of counter/timer we kept fixed value to count which take approximately 1 microsecond to complete its process of counting. It can be made variable easily by providing parameters during runtime. Here is the simulation of counter module.

Chapter 5

Protocol Design and Simulation

This chapter covers the design and simulation of sender and receiver protocol of End-to-End Error Control Protocol using go-back-n scheme. Below each and every line is explained in detail and simulation is explained in the end.

5.1 Sender

Following is the complete design of the sender protocol.

5.1.1 Design Code

```
module sender(//inputs
clk,
rst_n,
buff_flag,
data_avail,
pack_in,
control_in,
//outputs
pno,
control_out,
pack_out
);

parameter SID = 6'b000000;

parameter DWIDTH = 32;
parameter CWIDTH = 16;

parameter [2:0] IDLE = 000,
S1 = 001,
S2 = 010,
S3 = 011,
S4 = 100,
S5 = 101;

input clk, rst_n, buff_flag, data_avail;
input [DWIDTH-1:0] pack_in;
input [CWIDTH-1:0] control_in;
output [DWIDTH-1:0] pack_out;
output [CWIDTH-1:0] control_out;
output [3:0] pno;
reg [3:0] pno;
reg [DWIDTH-1:0] pack_out;
reg [CWIDTH-1:0] control_out;
reg cont_in = 0, conn_up = 0, en_timer,
en_timer_init, next_buff = 0;
reg [2:0] current_state, next_state;
reg [7:0] X = 8'b00000000, P, S = 8'b00000000;
wire ovf_timer, ovf_timer_init;

//E2EEC-G SENDER IMPLEMENTATION
always @(posedge clk or control_in or current_state)
begin
case(current_state)
IDLE: if(buff_flag == 1)
if(conn_up == 1)
next_state <= S3;
```

```
else
next_state <= S1;
else
next_state <= IDLE;

S1:
begin
X <= 8'b00000000;
P <= 8'b00000000;
control_out <= 16'b10_00000000_000000;
en_timer_init <= 1;
next_state <= S2;
end

S2:
if(cont_in)
begin
if(control_in == 16'b00_00000001_000000)
begin
conn_up <= 1;
en_timer_init <= 0;
P <= control_in[13:6];
cont_in <= 0;
next_state <= S3;
end
else if(ovf_timer_init == 1)
begin
en_timer_init <= 0;
next_state <= S1;
end

end
else
next_state <= S2;

S3: if(P < 11)
```

```
begin
pno <= P;
pack_out <= pack_in;
P <= P + 1;
next_state <= S4;
end
else
begin
next_state <= S5;
end

S4: if(cont_in)
begin
//X <= control_in[13:6];
//P <= X;
P <= control_in[13:6];
pno <= P;
cont_in <= 0;
next_state <= S3;
end
else
begin
next_state <= S3;
end

S5: begin
if(P == 11)
begin
S <= S + (P - 1);
P <= 4'bxxxx;
en_timer <= 1;
end
if(control_in[13:6] == S)
begin
if(data_avail)
next_state <= IDLE;
```

```
else
begin
control_out <= 16'b11_00000000_000000;
current_state <= 3'bxxx;
P <= 8'bxxxxxxxx;
pno <= 8'bxxxxxxxx;
conn_up <= 0;
en_timer <= 0;
end

end
else
begin
if(ovf_timer == 1)
begin
en_timer <= 0;
S <= S - 10;
P <= 0;
next_state <= S3;
end
end
end

default: next_state <= 3'bxxx;
endcase
end

always @(posedge clk or negedge rst_n)
begin
if(rst_n == 0)
current_state <= IDLE;
else
current_state <= next_state;
end

//status bits
```

```
always @(control_in)
begin
cont_in = 1;
next_state <= S4;
end

//IP Cores
counter timer01( .clk(clk),
.rst_n(rst_n),
.en(en_timer),
.cnt_val(),
.overf(ovf_timer)
);

counter timer02( .clk(clk),
.rst_n(rst_n),
.en(en_timer_init),
.cnt_val(),
.overf(ovf_timer_init)
);

endmodule
```

5.1.2 Simulation Code

```
module sender_tb();

reg clk, rst_n, buff_flag, data_avail;
reg [31:0] pack_in;
reg [15:0] control_in;
wire [31:0] pack_out;
wire [15:0] control_out;
wire [3:0] pno;

always #5 clk = ~clk;
```

```
initial
begin
clk = 0;
rst_n = 1;
#2 rst_n = 0;
#2 rst_n = 1;

#3 buff_flag = 1;
#3 data_avail = 0;

#30 control_in = 16'b00_00000001_000000;

//#90 control_in = 16'b00_00000101_000000;

#180 control_in = 16'b00_00001010_000000;


#1000 $stop;
end

always @(pno)
begin
case(pno)
4'b0000: pack_in = 32'b00000000000000000000000000000000;
4'b0001: pack_in = 32'b00000000000000000000000000000001;
4'b0010: pack_in = 32'b00000000000000000000000000000010;
4'b0011: pack_in = 32'b00000000000000000000000000000011;
4'b0100: pack_in = 32'b00000000000000000000000000000100;
4'b0101: pack_in = 32'b00000000000000000000000000000101;
4'b0110: pack_in = 32'b00000000000000000000000000000110;
4'b0111: pack_in = 32'b00000000000000000000000000000111;
4'b1000: pack_in = 32'b00000000000000000000000000001000;
4'b1001: pack_in = 32'b00000000000000000000000000001001;
default: pack_in = 32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
endcase
end
```

```
sender sender_1(//inputs
.clk(clk),
.rst_n(rst_n),
.buff_flag(buff_flag),
.data_avail(data_avail),
.pack_in(pack_in),
.control_in(control_in),
//outputs
.pno(pno),
.control_out(control_out),
.pack_out(pack_out)
);

endmodule
```

The sender protocol coding starts with the declaration of the port list specifying input and output ports. This is followed by some parameterization lines. Parameterization is performed in order to make the coding reusable for future use. In these parameters source address variable, control bus width, data bus width and state machines are defined. Later on inputs and output ports are defined along with their register size with some other registers and wires.

Mealy machine is implemented using ALWAYS block. There are two blocks of ALWAYS for the implementation of mealy machine. First block defines the operations of each state; second block assigns states to the current state variable.

Each state in the code performs different kinds of tasks derived from the protocol. Inside the first ALWAYS block a CASE is operating which checks the value of current state and jumps to the matching state. The CASE block have IDLE, S1, S2, S3, S4 and S5 state.

IDLE state checks if that buffers at Network Interface have packets to send or not. In case of no packets in buffers of NI, the next state will be IDLE. If it have packets to be sent, the state checks for the connection status with the receiver; If connection is already established then S3 is the next state but if connection is not established then S1 will be the next state.

S1 state comes only if there is no connection between sender and receiver; this state is used to initiate handshaking communication between sender and receiver. To do so, X and P are initialized to zero, a REQUEST packet is sent the receiver, timer is enabled (to monitor the ACK from receiver) and in the end S2 will be next state.

S2 will always check that if any control packet has been received or not. If a control packet is received, contin flag is set HIGH. if a control packet is received and its an ACK packet then following operations are performed; connup flag is set to HIGH (to show that connection has been established), timer set to receive ACK previously is disabled, P is assigned the sequence number, contin is set to LOW (for future alerts of control packets) and finally S3 is assigned the next state. But if the timer set previously overflows, timer is disabled and S1 will be the next state (to send REQUEST packet again). In the last if no control packet is received the machine remains at the same state.

S3 is responsible for sending all the packets in sequence. First, it checks if the current packet sequence is not exceeding the maximum number of packets in buffer. If it is exceeds, S5 is assigned as next state; but if it does not exceeds then pno is assigned the current packet sequence number (pno: ask NI for desired packet), the requested packet from NI is forwarded to the output port of the sender, P is incremented and finally S4 is set as next state.

S4 checks for any NACK received. If any NACK is received, its sequence number is assigned to P, control is again set to LOW and next state will be S3. This result of this state will stop the normal sending procedure of packet in sequence and will start sending from packet number P. If there is no NACK at control port, packets will be sent its normal sequence.

This state comes when all the packets are sent from buffer. In S5 state, if packet sequence number exceeds buffer size, current packet sequence number is stored in register S (to continue further sequence), initialize P as undefined and enable timer. If further packets are available at NI, IDLE is assigned the next state and the process will start from beginning; but if no data is available at NI, TERMINATE packet is sent, timers are disables and connection is broken. Further if no ACK is received (after sending all the packets from buffer) and timeout occurs, all the 10 packets are sent again.

On every positive clock cycle, next state is assigned to current state and whenever control packet is received cont in flag is set HIGH and state is explicitly changed to S4.

The simulation of the sender simply acts as if it is receiver and respond to sender module in different scenarios. It send ACK when sender sends the REQUEST signal, it send NACK for the fifth packet and also sends an ACK when all the packets are received. This simulation of sender is basically the supposition of the availability of receiver[5].

5.1.3 Simulation Results

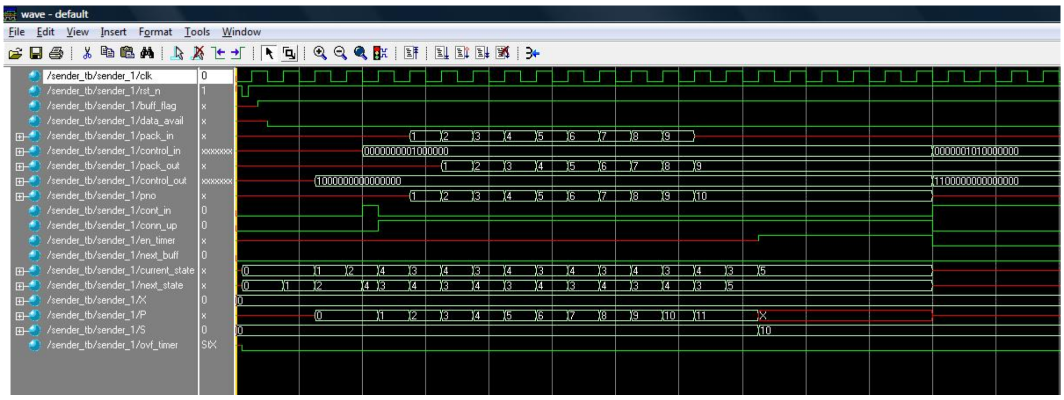

wave - default
File Edit View Insert Format Tools Window
/sender_tb/sender_1/clk 0
/sender_tb/sender_1/rst_n 1
/sender_tb/sender_1/buff_flag x
/sender_tb/sender_1/data_avail x
/sender_tb/sender_1/pack_in x
/sender_tb/sender_1/control_in xxxxxxx
/sender_tb/sender_1/pack_out x
/sender_tb/sender_1/control_out xxxxxxx
/sender_tb/sender_1/pno x
/sender_tb/sender_1/cont_in 0
/sender_tb/sender_1/conn_up 0
/sender_tb/sender_1/en_timer x
/sender_tb/sender_1/next_buff 0
/sender_tb/sender_1/current_state x
/sender_tb/sender_1/next_state x
/sender_tb/sender_1/X 0
/sender_tb/sender_1/P x
/sender_tb/sender_1/S 0
/sender_tb/sender_1/ovf_timer StX
0000000001000000
0000001010000000
1000000000000000
1100000000000000

Figure 5.1: Sender Simulation

5.2 Receiver

Here is the design and simulation code of receiver for End-to-End Error Control protocol.

5.2.1 Design Code

```
module receiver( //inputs
clk,
rst_n,
pack_in,
control_in,
//outputs
pack_out,
control_out
);

parameter DWIDTH = 32;
parameter CWIDTH = 16;

parameter [2:0] LISTEN = 000,
WAIT = 001,
S1 = 010,
S2 = 011;

parameter [1:0] ACK = 2'b00;

input clk, rst_n;
input [DWIDTH-1:0] pack_in;
input [CWIDTH-1:0] control_in;
output [DWIDTH-1:0] pack_out;
output [CWIDTH-1:0] control_out;
reg [DWIDTH-1:0] pack_out, pack_temp;
reg [CWIDTH-1:0] control_out;
reg [2:0] current_state, next_state;
reg [5:0] SID;
reg [7:0] Y = 8'b00000000, buffer_counter_val;
integer buffer_max = 9;
reg control_flag, conn_up = 0, buffer_counter_en = 0,
timer_listen_en = 0, pack_flag = 0, nack_flag = 0;
wire timer_listen_timeout;
```

```
//E2EEC-G RECEIVER IMPLEMENTATION
always @(posedge clk or current_state)
begin
case(current_state)
LISTEN:
begin
if(control_flag == 1)
begin
if(control_in[15:14] == 2'b10)
begin
SID <= control_in[5:0];
buffer_counter_val <= control_in[13:6];
conn_up <= 1;
timer_listen_en <= 1;
buffer_counter_en <= 1;
next_state <= WAIT;
control_out <= {ACK, buffer_counter_val, SID};
//control_flag <= 0;
end
else if(control_in[15:14] == 2'b11)
begin
SID <= 6'bxxxxxx;
conn_up <= 0;
buffer_counter_en <= 1'bx;
timer_listen_en <= 1'bx;
current_state <= 3'bxxx;
next_state <= 3'bxxx;
control_flag <= 0;
end
end
else
next_state <= LISTEN;
end

WAIT:
begin
```

```
if(pack_flag == 1)
begin
pack_temp <= pack_in;
next_state <= S1;
end
else if(timer_listen_timeout == 1)
begin
timer_listen_en = 0;
next_state <= S2;
end
else if(control_in[15:14] == 2'b11)
begin
SID <= 6'bxxxxx;
conn_up <= 0;
buffer_counter_en <= 1'bx;
timer_listen_en <= 1'bx;
current_state <= 3'bxxx;
next_state <= 3'bxxx;
control_flag <= 0;
end
else
next_state <= WAIT;
end

S1:
begin
if(pack_temp[29:22] == buffer_counter_val)
begin
if(buffer_counter_en == 1)
buffer_counter_val = buffer_counter_val + 1;
nack_flag = 0;
if(pack_temp[29:22] == buffer_max)
begin
control_out <= {ACK,buffer_counter_val+1,SID};
if(control_in[15:14] == 2'b11)
begin
```

```
buffer_counter_en <= 1'bx;
timer_listen_en <= 1'bx;
current_state <= 3'bxxx;
next_state <= LISTEN;
control_flag <= 0;
end
else
begin
timer_listen_en <= 1'b1;
next_state <= WAIT;
end
end
else
begin
timer_listen_en <= 1'b1;
next_state <= WAIT;
end
end
else
begin
//Drop Packet
if(nack_flag == 1)
begin
control_out <= {ACK,Y,SID};
next_state <= S2;
end
else
begin
nack_flag <= 1;
next_state <= S2;
end
end
end

S2:
begin
```

```
//Send NACK
control_out <= {ACK,buffer_counter_val,SID};
Y <= buffer_counter_val;
timer_listen_en <= 1;
next_state <= WAIT;
end

default: next_state <= 3'bxxx;

endcase
end

always @(posedge clk or negedge rst_n)
begin
if(rst_n == 0)
current_state <= LISTEN;
else
current_state <= next_state;
end

always @(control_in)
begin
control_flag <= 1;
end

always @(pack_in)
begin
timer_listen_en <= 1'b0;
pack_flag <= 1;
end

counter counter_rec( .clk(clk),
.rst_n(rst_n),
.en(timer_listen_en),
.cnt_val(),
.overf(timer_listen_timeout)
);

endmodule
```

5.2.2 Simulation Code

```
module receiver_tb();

reg clk, rst_n;
reg [31:0] pack_in;
reg [15:0] control_in;
wire [31:0] pack_out;
wire [15:0] control_out;

always #5 clk = ~clk;

initial
begin
clk = 0;
rst_n = 1;
#5 rst_n = 0;
#5 rst_n = 1;

#5 control_in <= 16'b10_00000000_000000;

#25 pack_in = 32'b00000000000000000000000000000000;
#20 pack_in = 32'b00000000000000000000000000000001;
#20 pack_in = 32'b00000000000000000000000000000010;
#20 pack_in = 32'b00000000000000000000000000000011;
#20 pack_in = 32'b00000000000000000000000000000100;
#20 pack_in = 32'b00000000000000000000000000000101;
#20 pack_in = 32'b00000000000000000000000000000110;
#20 pack_in = 32'b00000000000000000000000000000111;
#20 pack_in = 32'b00000000000000000000000000001000;
#20 pack_in = 32'b00000000000000000000000000001001;
```

```
#80 control_in <= 16'b11_00000000_000000;

#500 $stop;
end

receiver receiver_1( //inputs
.clk(clk),
.rst_n(rst_n),
.pack_in(pack_in),
.control_in(control_in),
//outputs
.pack_out(pack_out),
.control_out(control_out)
);

endmodule
```

In receiver protocol design, port list is declared followed by parameterization of control bus width and data bus width, which makes it reusable in case of changing the packet's size. States and ACK is defined using parameters. Input and output ports are defined with bus widths, followed by some internal registers and wires. Just like sender protocol, receiver is also using the mealy machine to design the functionality of the protocol.

The first ALWAYS block contains the state machines and its functionalities in detail. These states are inside CASE block; there are four total states, LISTEN, WAIT, S1 and S2.

LISTEN state always listen for requests from sender module. It monitors the control port, if any control packet is received. If its a REQUEST then SID grabs the source address from the packet, buffercounterval grabs the packet sequence number from the packet, connup is set HIGH because connection is established, timer is initialized, buffercounteren is enabled(which enables the counting of number of packets received), an ACK is sent back showing that connection has been established (and receiver is ready to receive packets) and finally WAIT is set as next state. But if the control packet received is a TERMINATE packet then all the flags are disabled and states are also disabled (which stops the execution and connection is terminated). If there are no control packets received it will remain on the same state.

WAIT state monitors if any data packet is received or not. If any data packet is received the packflag will be set HIGH; if its HIGH the packet is temporarily stored in a register and S1 is assigned as the next state. But if timeout occurs and no data packet has been received then timer is disabled and S2 is assigned as next state. If at any stage TERMINATE packet is received all the flag are disabled and connection is terminated.

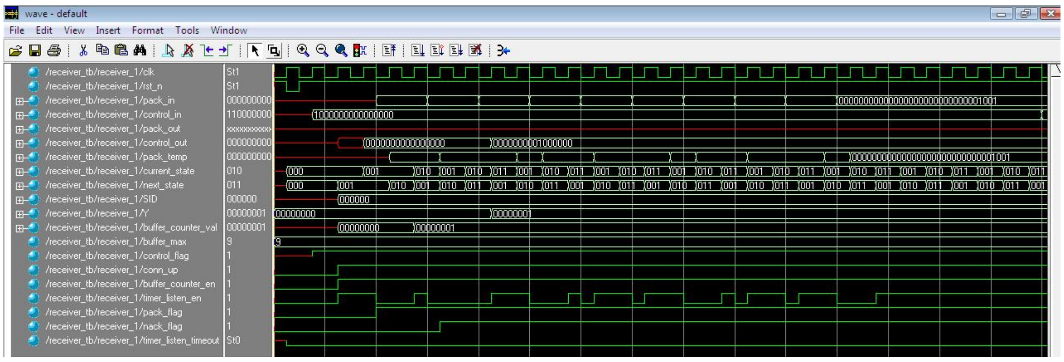
S1 checks the sequence number of received packet, if its in order it is forwarded to NI and the counter is incremented. But if the received packet is the last packet of the sequence an ACK packet is sent informing the sender that all the packets have been successfully received. After informing about the successful delivery of all the packets in the sequence, receiver initialize the timer and wait for response. If the response is TERMINATE all the flags are disabled and execution is stopped but if sender have to send further packets WAIT is assigned as the next state. In case if the received packet is out of order or corrupt or haven't received till timeout nackflag is set to HIGH (which indicates that a NACK packet has to be sent). In this way a NACK is sent and S2 is assigned as next state.

S2 comes in operation when a NACK is sent but still no response has been received, in which case timer is initialized and a NACK is again sent, WAIT is set as next state (so that expected packet could be received).

The second ALWAYS block assigns the next state variable to current state variable, which maintains the execution of the mealy machine. When ever a control packet is received, contin is set to HIGH and whenever data packet is received, timer is disabled and pack_flag is set to HIGH.

The simulation shows the supposition of sender, it behaves exactly like a sender is communicating with the receiver. Which helps us in determining the logic simulation of receiver. The simulation send a REQUEST packet to receiver and after successful establishment of communication it sends all the ten packets in sequence and in the end of successful delivery a TERMINATE packet is sent, which terminates the session[5].

5.2.3 Simulation Results



```
/receiver_tb/receiver_1/clk                     St1
/receiver_tb/receiver_1/rst_n                   St1
/receiver_tb/receiver_1/pack_in                 000000000
/receiver_tb/receiver_1/control_in              110000000
/receiver_tb/receiver_1/pack_out                xxxxxxxxxx
/receiver_tb/receiver_1/control_out             000000000
/receiver_tb/receiver_1/pack_temp               000000000
/receiver_tb/receiver_1/current_state           010
/receiver_tb/receiver_1/next_state              011
/receiver_tb/receiver_1/SID                     000000
/receiver_tb/receiver_1/Y                       00000001
/receiver_tb/receiver_1/buffer_counter_val      00000001
/receiver_tb/receiver_1/buffer_max              9
/receiver_tb/receiver_1/control_flag            1
/receiver_tb/receiver_1/conn_up                 1
/receiver_tb/receiver_1/buffer_counter_en       1
/receiver_tb/receiver_1/timer_listen_en         1
/receiver_tb/receiver_1/pack_flag               1
/receiver_tb/receiver_1/nack_flag               1
/receiver_tb/receiver_1/timer_listen_timeout    St0
```

Figure 5.2: Receiver Simulation

Chapter 6

Conclusion

The main objective of this thesis was to design and simulate the end-to-end reliable protocols of NoC along with its supporting modules. This is a very strong step towards creating a complete NoC model and implementing these protocols on real-time FPGA hardware. The design of these protocols unleashed the core problems which can be avoided in future. Efforts has been made to make it reusable which may help future researchers/designers to save time by picking ideas or code and change it accordingly. This coding resulted in the deep studies of NoC, its model and these protocols; which unveils the minute details of the designing aspect of NoC by declaring the requirements needed to fulfill in achieving a reliable NoC model.

In future, this research will be taken forward making these designs as milestones. Future work includes revising of these protocols in much detail and making it more perfect. Integrating CRC module at every router, building a 2D mesh network of about 100 LPEs (10 x 10 interconnections). As this model is completed, it will be implemented in Xilinx ISE in order to calculate its coverage over an FPGA chip, later on it will be synthesized and downloaded to FPGA chip. Real-time debugging will be performed using ChipScope Pro in order to test the performance of these protocols in real-time environment.

When all the above processes are performed, the results will be compared with the results of NS-2 Simulator. If the results proves the improved reliability of these protocols, the research will be forwarded to international conferences and will be published using research papers.

References

[1] Ali, M., Welzl, M., Zwicknagl, M., *Networks on Chips: Scalable Interconnects for Future Systems on Chips*, Proceedings of the 3rd IEEE International Conference on Circuits and Systems for Communications (ICCSC'06), 6-7 July 2006, Bucharest, Romania.

[2] Ali, M., Welzl, M., Zwicknagl, M., Hellebrand, S., *Considerations for Fault-tolerant Networks on Chips*, IEEE ICM'05, Islamabad, Pakistan, 13-15 December 2005.

[3] Ali, M., Welzl, M., Hessler, S., Hellebrand, S., *A Fault Tolerant Mechanism for handling Permanent and Transient Failures in a Network on Chip*, International Journal of High Performance Systems Architecture (IJHPSA), Vol.. 1 No. 2. 2007, pp. 113-123, Inderscience Publishers

[4] Ali, M., Welzl, M., Hessler, S., Hellebrand, S., *A Fault Tolerant Mechanism for handling Permanent and Transient Failures in a Network on Chip*, Proceedings of the 4th International Conference on Information Technology : New Generations (IEEE ITNG 2007), Las Vegas, USA, 2-4 April 2007.

[5] Ali, M., Adnan, A., Welzl, M., *Design of a fast and low-level fault-tolerant protocol for network on chips*, Proceedings of the IEEE International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (IEEE CISSE 2006), online conference, 4-14 December 2006.