2013

Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's

Krunal Jetly
University of Windsor

Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's

By

Krunal Jetly

A Thesis
Submitted to the Faculty of Graduate Studies
Through Electrical and Computer Engineering
In Partial Fulfillment of the Requirements for the
Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2013

Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's

By

Krunal Jetly

APPROVED BY:

G. Zhang
Mechanical, Automotive, and Materials Engineering

R. Rashid
Electrical and Computer Engineering

M. A. S Khalid, Advisor
Electrical and Computer Engineering

April, 2013

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include material from two different theses as follows:

(i) From Chapter 3 section 3.1.1.3 (Routing Algorithm), Figure 3.1, Figure 3.2, Figure 3.4, Figure 3.7, Figure 3.8 from Thesis by Mike Brugge "**Design and Evaluation of a Parameterizable NoC Router for FPGAs**". Sep 21 2009

(ii) From Chapter 2 Section 2.3 (Interface and Signals), Chapter 3 section 3.3.1(Adapter Overview), Chapter 4 Section 4.3 (NIOS II Programing) and section 4.4 (Modelsim Simulation), Figure 3.5, Figure 3.6, figure 4.6 and figure 4.9 From Thesis by Matt Murawski "**NoC Prototyping on FPGAs: Component Design, Architecture Implementation and Comparison**". May 18 2012

I have included copies of such copyright clearance permission forms to my Appendix A and the permissions were received through copyright holder's email.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Network on Chip (NoC) is an interconnection paradigm which is scalable and efficient for connecting increasing number of components on Field Programmable Systems on Chip (FPSOC). The router is a key component in NoC that impacts area performance, power consumption, etc. In this thesis we evaluate and compare two different router designs using real world benchmark. The first router uses Store-And-Forward strategy (SAF) and XY routing algorithm and the second router uses Wormhole (WH) as forwarding strategy and source routing algorithm. These routers were used to implement 4x4 mesh NoCs. A multi processor system benchmark obtained from Altera was implemented in each NoC. This enabled us to evaluate and compare the routers using the real world benchmark design. The evaluation metrics used were area, throughput, power consumption and maximum clock frequency. Experiment results show that the SAF router is superior to the WH Router.

Acknowledgements

It is an honor for me to have worked with Dr. Mohammed A. S. Khalid throughout my Meng degree and Master's in applied science degree here at the University of Windsor. His guidance, encouragement, wisdom and support carried me through the course of this thesis. My deepest gratitude goes out to him. My appreciation also goes out to my thesis committee members, G.Zhang and R.Rashid, for their time to sit on my committee and reviewing my thesis

I want to thanks my family for all their constant support and encouragement. Thanks to my parents for their understanding me. Thanks for helping to keep me focused day after day.

Finally, I need to acknowledge my friends and fellow graduate students at the University of Windsor. Matt Murawski and Mike Brugge thank you for your friendship and guidance. Thanks to Matt for your company and taking those long calls at your office and helping me out in my Research. You could always provide me with the help I needed. Lastly, thanks to the rest of my colleagues; amanjot, manveen who made this great milestone in my life so enjoyable.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
ALUT	Adaptive Look Up Table
ASIC	Application Specific Integrated Circuit
AVM	Avalon Master
AVS	Avalon Slave
AWB	Avalon-Wishbone
BE	Best Effort
BTE	Burst Type Extension
CAD	Computer Aided Design
CLK	Clock
CPU	Central Processing Unit
CS	Chip Select
CTI	Cycle Tag Identifier
DDR	Double Data Rate
EDS	Embedded Design Suite
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GS	Guaranteed Service
HDL	Hardware Description Language
I/O	Input / Output
IC	Integrated Circuit
IDE	Integrated Development Environment
IP	Intellectual Property

IRQ	Interrupt Request
JTAG	Joint Test Action Group
QoS	Quality of Service
KB	Kilo Byte
LAN	Local Area Network
LE	Logic Element
LED	Light Emitting Diode
MB	Mega Byte
MPSoC	Multi-Processor System on Chip
NA	Network Adapter
NoC	Network on Chip
OCP	Open Core Protocol
OE	Output Enable
OSI	Open Systems Interconnection
PIO	Peripheral Input Output
PWR	Parameterizable Wormhole Router
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SAF	Store and Forward
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on Chip
SOPC	System on Programmable Chip
TDM	Time Division Multiplexing
UART	Universal Asynchronous Receiver/Transmitter
VC	Virtual Channel
VCI	Virtual Component Interface

VCT	Virtual Cut Through
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
WB	Wishbone
WH	Wormhole

Chapter 1: Introduction

INTRODUCTION

The world of silicon integrated circuits has changed a lot in recent years. We have seen the feature size of IC decreased from 90nm to 40nm as predicted by moore’s law. This has enabled systems on chip (SoC) which can implement complex systems on a single chip which earlier needed the Printed Circuit Boards. The electronic industry has continuously changed and evolved by packing more functionality in smaller area of silicon, which has resulted in Increasing transistor density, higher operating frequencies, shorter time-to-market and reduced product life cycle [1]. As the number of computational modules in single IC continuous to increase the new interconnection paradigm is needed that is efficient and scalable.

The electronic industry started from IC which used to perform basic logic functions and then moved to PLD, CPLD, micro controllers and microprocessors along with these digital blocks complex modules that could implement computationally intensive task were implemented on single IC’s. Further, with increase in transistor density it was feasible to add more of these blocks in smaller area which lead us to the embedded systems. An embedded system is a combination of processor and supporting digital blocks which is designed to perform a specific task. In today’s technologically advanced world embedded system plays an important role as we can find embedded systems almost everywhere for example cameras, DVD players, washing machines and also in large stationary units like cellular base stations and factory controllers. At the present time SoC enabled us to implement a complete embedded system on a single chip.

As the transistor density is still increasing it was possible to incorporate more digital blocks and so SoCs started to contain many hardware and/or software blocks, such as processors, DSPs, memories, peripheral controllers, gateways, and other custom logic blocks. SoCs have just started to get intricately complex and in future there will be more issues coming up with more shrinkage in size of transistors. Interconnection Techniques and routing methodologies has become one of the most research intensive areas in SoC designs [2]. The issues concerned with interconnection methodologies and routing methodologies has been one of the most research intensive areas as it affects each facet of SoCs design. Interconnection methods greatly impact cost and performance of a SoC.

In earlier days the number of blocks in single IC were limited hence a point to point or simple Bus interconnection was used. But, as the designs started to get complex Bus based systems could not handle the increase in number of blocks. The routing methods used after SoCs introduction was point to point connection between IP's. Dedicated wires were effective until the system were not much complex and less number of the IP's were there on the SoC's, but as the complexity increased the routing resources were consumed too quickly. Also dedicated wiring leads to decrease in the resource reusability and flexibility. So to overcome this limitation designer incorporated usage of shared bus where a set of wire is common between different multiple cores of the system this increased the reusability and scalability of the resources. For achieving this Master and slave scheme has been carried out which uses control signals and slave waits for the data to be received or requested from the master. However, with the systems started having more masters and slave the contention increased and results to bottleneck which gets worse with increase in complexity. Along with this there is concerns regarding complication of the protocols while trying to eliminate the scalability problems. Design and verification times also grow with SoC complexity [3].

ASIC designs are application specific ICs they are used to replace the time consuming part of the software. ASIC's help in increasing the speed of the system by great extent. The ASIC design not just concentrates on increasing speed it also decreases area and power consumption of the system. ASICs have been a good implementation for large amount of production because ASIC is not easy to implement as the systems are very complex and even though automation has helped the industry ASIC designing takes away a lot of engineering resources. So to overcome this difficulty the ASICs have started to be replaced by FPGA’s. The competitive world of technology has made a new requirement of bringing the new technology out as soon as possible, which is stated as “time to market”. This has made the designers to shift to other resources which are much faster to design and validate. FPGA has been an answer to that and this is the reason why we see usage of FPGA when there is a small scale requirement FPGA is the viable choice but for large scale production still ASIC is preferred.

FPGA has captured significant part of the IC market. Before FPGA were replacing small units as it was not containing much of the logic elements and because of that we did not care of the routing carried out in FPGAs .With the increase in transistor density more number of logic elements were packed into FPGAs and current FPGAs can be considered to be FPSoCs (Field Programmable Systems on Chips). With the current trend of integration of more complex FPSoC’s we need a better communication infrastructure and protocol which will alleviate the problem of scalability by supporting multiple concurrent connections between IP cores and along with that it should allow the reuse of area specific pre-designed and pre-tested IP cores.

NoC is a promising interconnection paradigm that can be used in SoC’s and FPSoC’s. The Basic concept behind NoC is similar to computer networks, multiple computers are connected to different routers and these routers are connected to each other using different topologies. The same concept is used in NoC’s with IP cores replacing computers. Much research needs to be done in Exploring the design space of NoC implementation in FPGA’s

This Thesis is intended to shed light on some of the tradeoffs involved in NoC implementation on FPGA’s. We implemented two different Router designs and compared them using a real world benchmark application. Previous research has evaluated compared router designs using data from traffic generators which is not as good as the traffic generated by a real world application. Our research results will be useful to the future designers of NoC based systems on FPGAs.

1.1 Thesis Objectives

The main goal of this research is to evaluate and compare two NoC routers developed in previous research work. The Evaluation metrics used to compare the two routers are area, flit – size, Ports, power, throughput and clock frequency. Our research results will help designers working on implementation of large complex NoC based systems on FPGAs by allowing them to make informed choices on design tradeoffs. This research has the following major objectives:

1. Investigate SAF router implementation on FPGAs.
2. Developed network adapter based on wishbone protocol that could be interfaced to both SAF and WH router.
3. Implement two mesh NoCs using SAF and WH Routers respectively, to run a real world bench mark design.
4. The two Mesh NoCs were synthesized and results were evaluated and compared using metrics area, power, clock and throughput.

For the first objective, we have investigated and understood the functionality and design of SAF Router, WH router and NoC adapters. This led us to make changes to SAF Router and NoC adapters. Testing was done with the help of a realistic benchmark. The experimental framework was developed in VHDL, allowing NoC synthesis in Altera Quartus II design environment (Simulation was done using Modelsim). SOPC Builder CADtool was used to implement the mesh to NoCs. NIOSII IDE was used to run the Benchmark design on the NoCs implemented in FPGA.

1.2 Thesis Organization

The remainder of this thesis is organized as follows. In chapter 2, we present the background and related work for FPGAs and NoCs. Chapter 3 covers detailed description of the routers used and the network adapters. Chapter 4 describes NoC implementation and Evaluation framework. Chapter 5 presents experimental comparison and analysis. Chapter 6 concludes the thesis and discusses possible future work.

Chapter 2: Background and Previous Work

In this chapter, we will go through previous work done and also the background that is relevant to this research is presented. This chapter begins with a brief overview of FPGA technology, detailing the benefits. Then describe different aspects of Network-on-Chip (NoC). This is followed by description of the standard sockets used in NoC's. We cover the Avalon [4] and Wishbone interfaces [5]. Finally the CAD tools used in our research are briefly described and the chapter concludes with the description of related research.

2.1 FPGA Technology

Field Programmable Gate Arrays (FPGA) is special integrated circuit, in which the logic elements, routing resources are pre-fabricated. Different digital circuits can be specified using hardware description languages such as VHDL and Verilog and then synthesized and implemented on FPGAs. In many real world applications which requires low to medium volumes FPGAs have replaced ASICs due to their many advantages such as low non recurring expenses (NRE), fast time to market and flexibility. FPGAs are made up of Logic Elements (LE), in the form of look-up tables, which are used to implement custom logic. FPGAs have also evolved and with advances in technology and contain advanced components, such as DSP blocks, memory blocks, serial interfaces and hard or soft CPU cores. In this thesis, FPGAs are used as implementation fabric to evaluate and compare NoC routers.

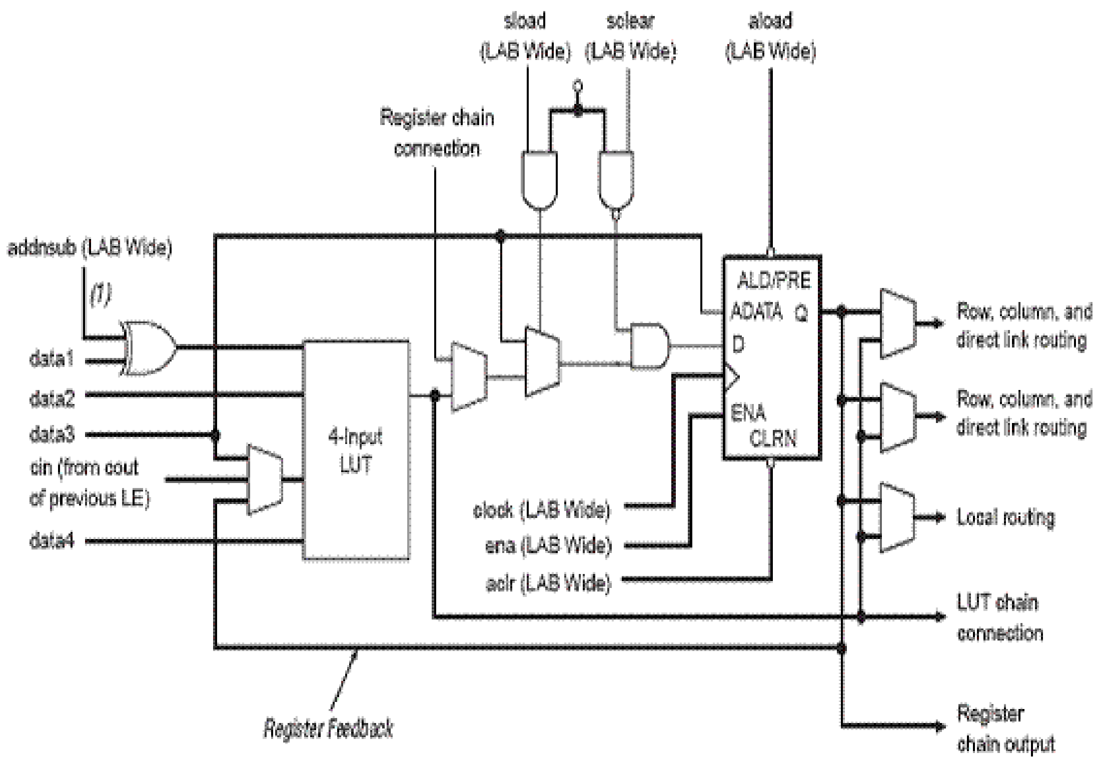


Figure 2.1: Altera Logic Element Architecture [5]

2.2 Overview of NoC

This section describes the basic building blocks used in NoCs and their operation.

2.2.1 NoC Building Blocks

NoC aims to provide a network on the chip which allows effective communication between computational components (IP cores). Figure 2.2 depicts a sample NoC which consists of 4 IP cores which are connected using a single router. An adapter is used to connect an IP Core to the router and a link is used to connect a router to adapter or to another router.

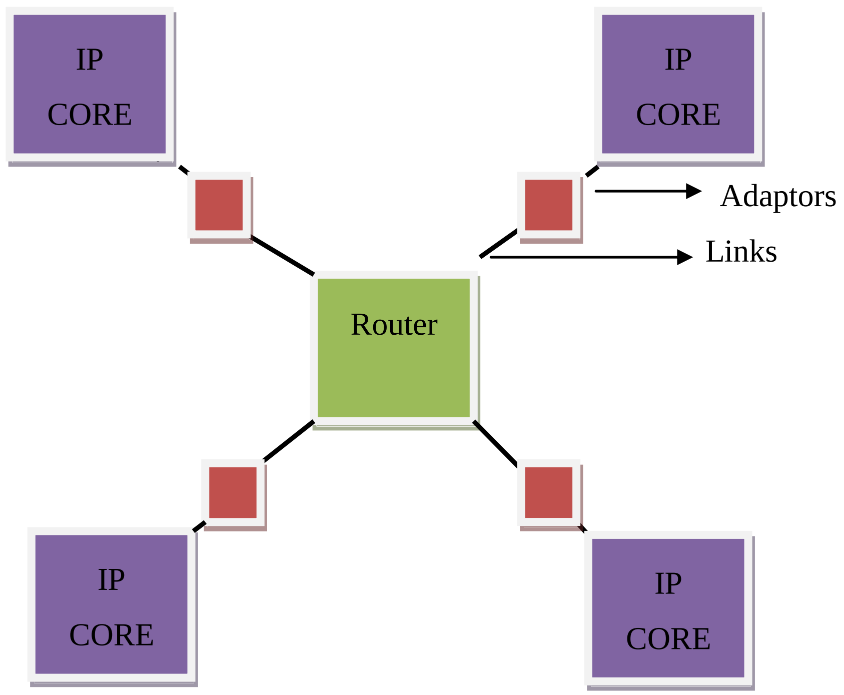


Figure 2.2: A NoC Interconnection Networks 4 Basic Functional Blocks

2.2.1.1 Links

This component provides one to one connections for a routing node with a network adapter or another routing node. It also provides separate control lines for connection establishment and teardown.

2.2.1.2 Network Adapters

Network Adapters convert the high level protocols (HLP) that IP uses into the packet-based communication protocols of the NoC and vice versa. They are responsible for storing IP core addresses, creating and disassembling messages, forming packets and breaking them into flits, implementing end-to-end flow control, crossing clock domains, and other higher level network issues.

2.2.1.3 Routing Node

The Routing node handles the flow of packets (Traffic) in the network based in NoC. It basically runs the routing algorithm which determines the method of flow of packets and it is also a central component of the NoC. The parameters of the routers are thus important as they can affect the performance of the network to a large extent. The parameters of routers are routing algorithm, forwarding strategies, flit size, and number of ports.

2.2.2 Parameters of NoC

Every NoC has its own unique parameters. The three main parameters are: Mapping, Communication Mechanism and Infrastructure. These parameters play an important role in the overall performance of the NoC and so it has been the most important research topic for the NoC researchers. It is preferred that these parameters should be chosen according to the application requirement in order to enhance the NoC performance for that particular application. This thesis attempts to provide an insight into how the choice of parameter selection affects tradeoffs in cost, performance and power consumption of NoCs in FPGAs.

2.3 Interfaces and Signals [22]

This Section describes in detail the interfaces used in our thesis along with their protocols and signals used for communication. The standard sockets used in this thesis include Silicore/Opencore.org‘s Wishbone and Altera‘s Avalon interfaces.

2.3.1 Wishbone

Wishbone is one of the handshaking protocol, we have used it as a general purpose interface between IP core modules because it’s an open source synchronous SoC interconnection architecture. It also has a great range of bandwidth in terms of data transfer speed.

2.3.1.1 Signals

Wishbone has a variety of signals, used to provide flexibility and compatibility for attached IP cores. The signals common to both master and slave devices are:

CLK_I – Clock input. All Wishbone output signals are registered on the rising clock edge.

DAT_I – Input data array, with a maximum size of 64 bits.

DAT_O – Output data array, with a maximum size of 64 bits.

RST_I – Synchronous reset signal

TGD_I – Input data tag array, containing information regarding the DAT_I signal. The data tag contains user defined information.

TGD_O – Output data tag array, associated with the DAT_O signal.

Master signals include:

ACK_I – Acknowledge signal used for the handshaking protocol, which indicates the termination of a bus cycle.

ADR_O – Address output array

CYC_O – Cycle output signal, indicating a valid bus cycle when asserted. For burst and block cycles, the CYC_O signal is held high for multiple transfers until the final cycle.

ERR_I – Error input signal, used as an alternative to ACK_I to indicate a failed transfer. The exact functionality of this signal depends on the IP core.

LOCK_O – Lock output signal, used to ensure a transfer is uninterruptable. The exact functionality of this signal depends on the IP core.

RTY_I – Retry input signal, used as an alternative to ACK_I. The exact functionality of RTY_I depends on the IP core.

SEL_O – Select output array, used for fine control over data granularity. The size of SEL_O depends on the data width and granularity. For example, 8 bits are used for a 64 bit data bus with byte granularity.

STB_O – Strobe output signal, used to indicate valid data transfer cycles. Unlike CYC_O, STB_O is deasserted after a transfer.

TGA_O – Address tag output signal, used to contain tag information associated with the ADR_O signal. For burst transfers, the TGA_O tag contains Cycle Tag Identifier (CTI), and Burst Type Extension (BTE) tags regarding burst specifics.

TGC_O – Cycle tag output signal, used to contain tag information regarding a bus cycle. It can be used to distinguish between a single, block or RMW cycle. WE_O – Write enable output signal, used to indicate a write transfer.

Slave signals receive the exact same master signals, but in an opposite direction. For example, CYC_I receives the cycle output signal, whereas ACK_I sends an acknowledge response from the slave to the master‘s ACK_O signal. The types of Wishbone bus cycles are divided into three sections – Single, block and burst.

Single transfers use a handshaking protocol shown in Figure 2.3. The master core initiates a transfer with the strobe signal, where the slave responds with ACK, ERR or RTY. Strobe is held high until a response is received, where the stobe signal is then de-asserted. A cycle termination signal (ACK, RTY or ERR) must be asserted according to the logical AND of STB and CYC.

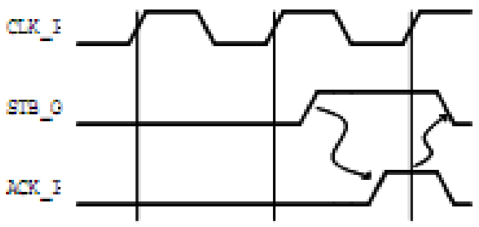


Figure 2.3: Single Transfer Hand Shake

A more detailed waveform is shown in Figure 2.4, where a sample single read transfer is shown. CYC and STB are asserted to indicate a read request, where the address, selection and associated tags are also applied. The slave responds with an acknowledge signal at (1), as well as the data and associated tags.

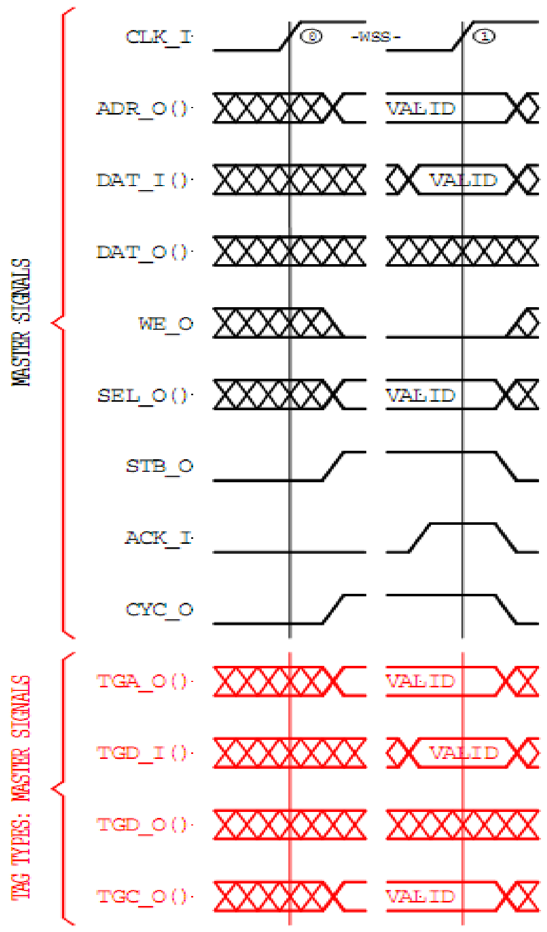


Figure 2.4: Single Read Transfer

A single write request is very similar, shown in Figure 2.5, where WE_O is asserted, data is provided by the master on DAT_O and the slave terminates the transfer with an acknowledge at edge (1).

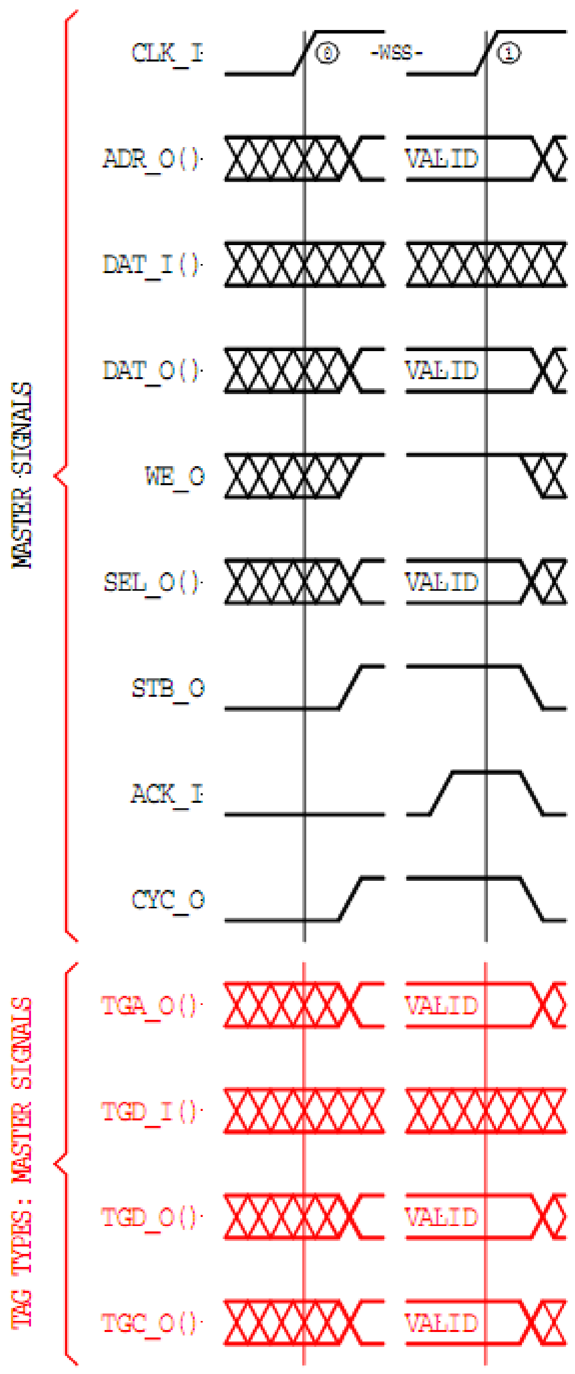


Figure 2.5: Single Write Request Signals

These two requests can be performed in a Read-Modify-Write (RMW) request, shown in Figure 2.6. The CYC signal is held high for the duration of the transfer, while the separate strobe signals perform the actual individual transfers.

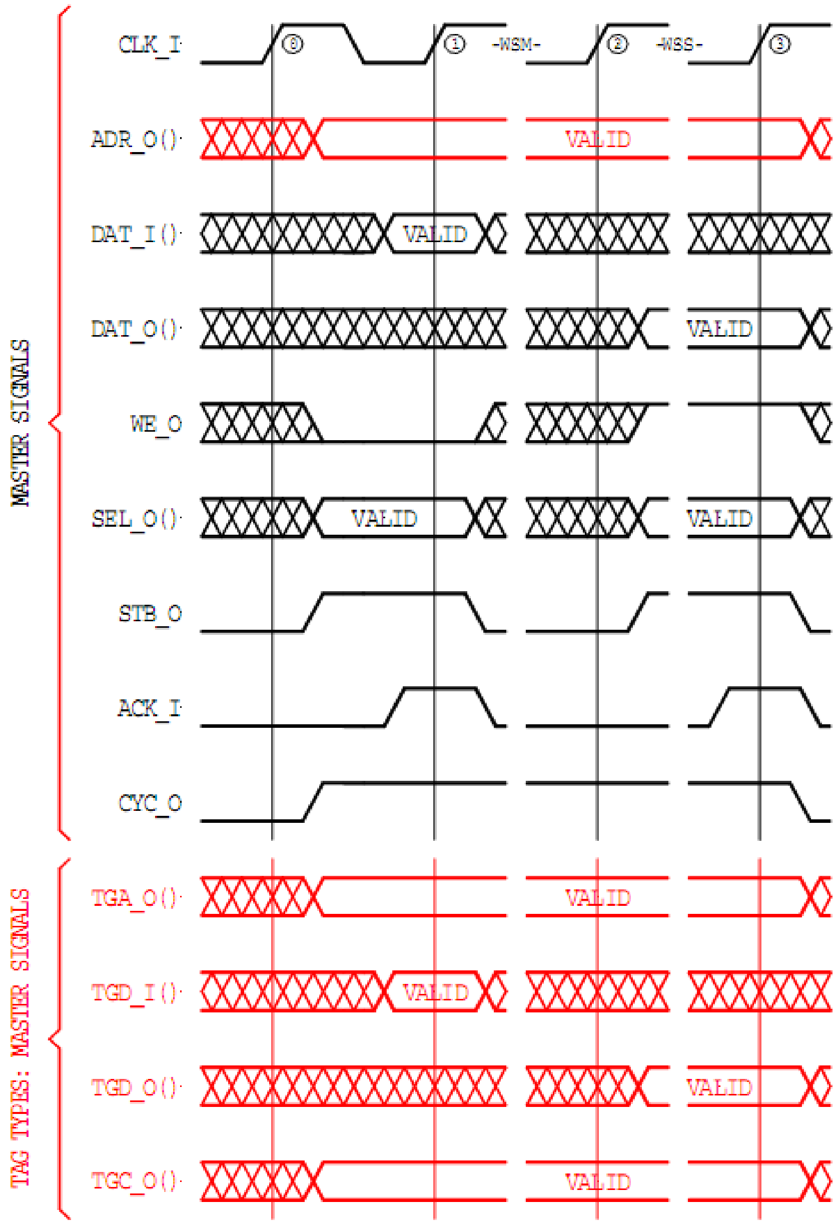


Figure 2.6: Wishbone RMW

The block transfers operate slightly differently, where the acknowledge signal may be held high for a number of cycles for multiple transfers for increased bandwidth and reduced delay. A block read request is shown in Figure 2.7. Note that CYC is asserted for the entire duration of the transfer.

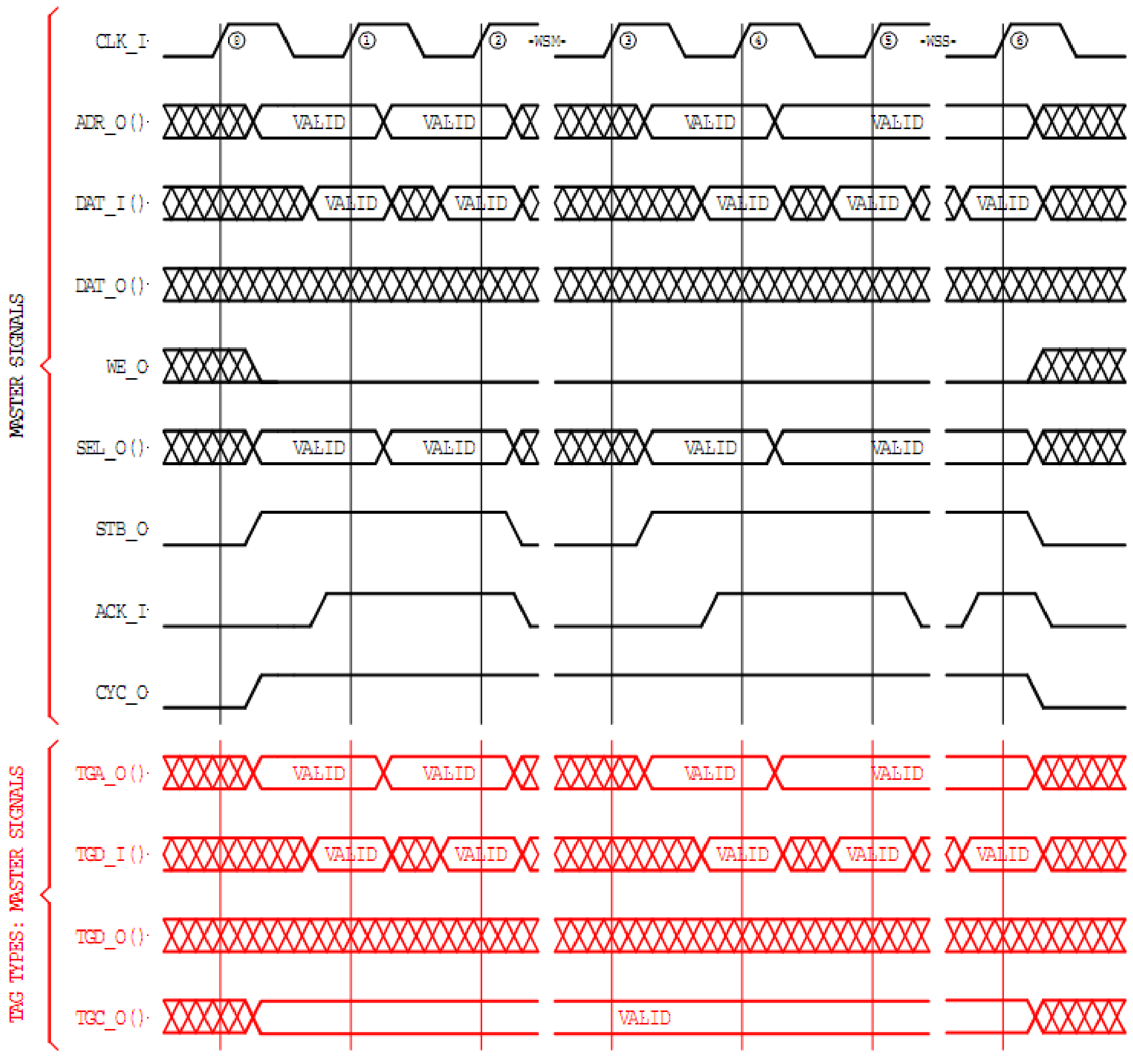


Figure 2.7: Block Read Request for Wishbone

Burst transfers address the issue of the additional delays involved when cycle termination signals, in order to reduce wire routing delay, become synchronous. Additional tag signals are used in order to let the slave know of predictable transfers in advance. The Address Tag contains two additional identifiers, used to specify burst characteristics: Cycle Tag Identifier (CTI) and Burst Type Extension (BTE). CTI is 3 bits, and BTE is 2 bits. They are shown in Table 2.1 and Table 2.2.

CTI (2:0)	Description
000	Classic Cycle
001	Constant address burst cycle
010	Incrementing address burst cycle
011-110	Unused
111	End-of-Burst

Table 2.1: Cycle Type Identifier

BTE (1:0)	Description
00	Linear burst
01	4-beat wrap burst
10	8-beat wrap burst
11	16-beat wrap burst

Table 2.2: Burst Type Extension

The Classic Cycle is not a burst transfer, where no information about future master cycles is given. End-of-Burst is used to indicate that the current cycle is the last cycle in the burst. Constant address cycle causes a continual access to the same address, until End-of-Burst is given. Lastly, incrementing address burst uses the Burst Type Extension tag to further define the address behavior. Consecutive addresses, based on BTE are applied. Linear burst simply adds one to the address per cycle, while the beat wrap bursts are modulo the wrap size. Figure 2.8 is an example of an incrementing address burst transfer.

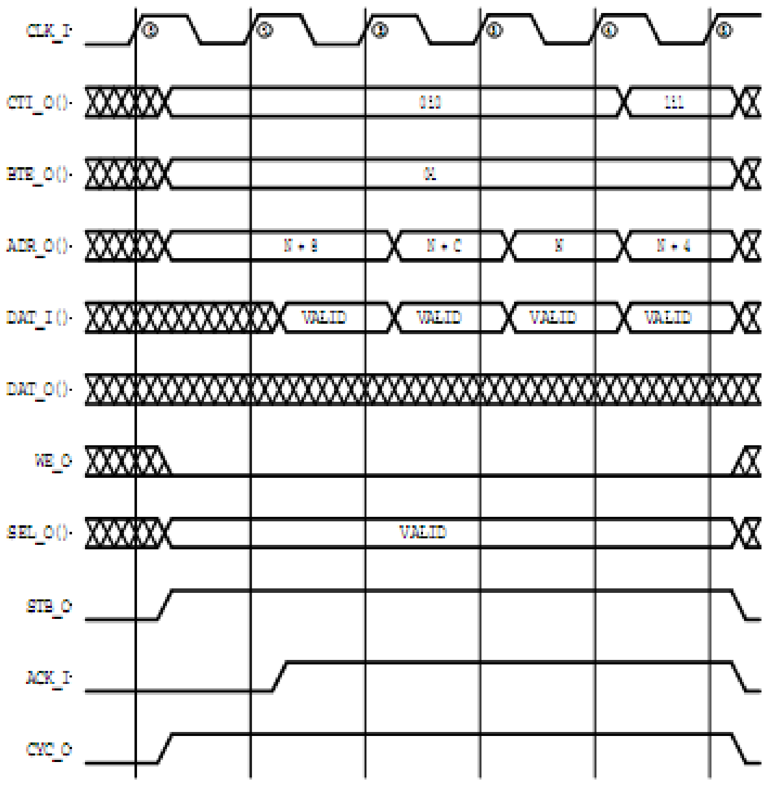


Figure 2.8: Incrementing Bursts for Wishbone

2.3.2 Avalon Interface

Altera's Avalon interface is a flexible interconnection architecture aimed at SoCs on FPGAs. While Avalon has six different types of interface – Memory Mapped, Streaming, Tristate, Clock, Interrupt and Conduit – the Memory Mapped interface will be the main focus due to the nature of the research. The other types will be briefly explained.

2.3.2.1 Avalon-MM

The slave interface uses the following signals. Note that not all of them are required.

Read – Read is asserted to indicate a read transfer, where readdata is required.

Write – Write is asserted to indicate a write transfer, where writedata is required.

Address – Contains the address used for read and write requests, and can be up to 32 bits. Readdata – Contains the data for a read response.

Writedata – Contains the data for a write request.

Byteenable – Used for fine control over data granularity. Selects a specific byte lane for transfer, and has the available bit widths of 1, 2, 4, 8, 16, 32, 64 and 128.

Begintransfer – Asserted for the first cycle of each transfer, regardless of waitrequest. Waitrequest – Asserted by the slave to indicate that it is unable to respond to a request. Readdatavalid – Asserted when data is supplied in response to a read request.

Burstcount – Indicates the number of transfers that a burst contains, with a maximum size of 32 bits.

Beginbursttransfer – Asserted on the first burst cycle to indicate the start of a burst transfer. Figure 2.9 demonstrates examples of slave read and write transfers using Avalon-MM.

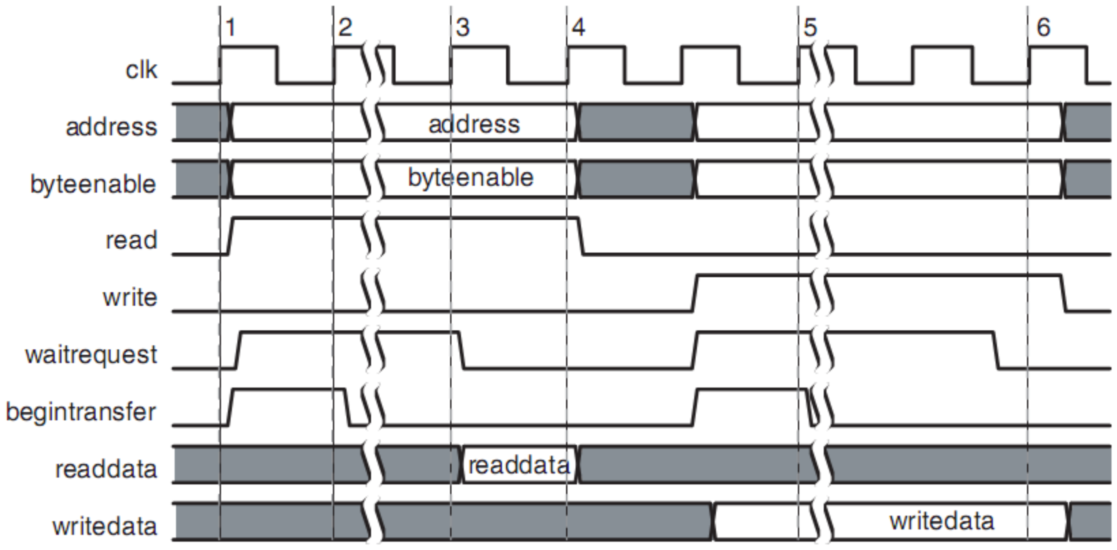


Figure 2.9: Avalon MM Transfer

2.3.2.2 Avalon-ST

Avalon Streaming (Avalon-ST) interfaces are used for driving unidirectional and high bandwidth data, where applications include DSP, packets and multiplexed streams. Connected components act as either a source or a sink, with data flowing from the source into the sink.

2.3.2.3 Avalon-MM Tristate

Avalon Memory-Mapped tri-state interfaces allow off-chip components to be used. It is relatively similar to Avalon-MM, but with the inclusion of Chip Select (CS) and Output Enable (OE) signals, as well as a bidirectional data line. When chip select is present, all signals are ignored unless CS is asserted. When OE is de-asserted, the slave will not drive its data lines.

2.3.2.4 Clock

Clock provides synchronization for the Avalon interface and includes a synchronous reset signal. All internal logic returns to initial states when reset is asserted.

2.3.2.5 Interrupt

Each applicable slave device has an interrupt output signal (IRQ), which is asserted when service is needed. The master device receives up to 32 interrupt signals and, depending on the IRQ scheme, services each interrupt according to a priority table.

2.3.2.6 Conduit

The Conduit interface is used with Altera's SOPC Builder software and is used for exporting signals for connection with external FPGA pins.

2.4 CAD Tools for NoC Implementation on FPGAs

2.4.1 Altera Quartus II

Quartus II Software is CAD tool suite provided by Altera Corporation. It is designed to map hardware designs conveniently and efficiently to altera FPGAs. The design flow for Quartus II is shown in figure 2.10

As shown in the flow chart first step of the design flow is the design specification. The design entry is done by creating an HDL file. Synthesis is the process where the HDL code is checked for any syntax or semantic errors. The HDL is compiled to an intermediate form and then an equivalent and optimized RTL implementation is synthesized. Place and Route maps the hardware described at the RTL level to available logic and routing resources on the FPGA. Timing Analysis evaluates the performance of the design implemented on FPGA and attempts to meet timing requirements and attain timing closure. Simulation is used to verify the functionality of the HDL model and its FPGA implementation. Finally, Programming and Configuration stage generates the bit stream required to configure. Quartus II Version 9.0 running on sun solaris was used in this research.

.

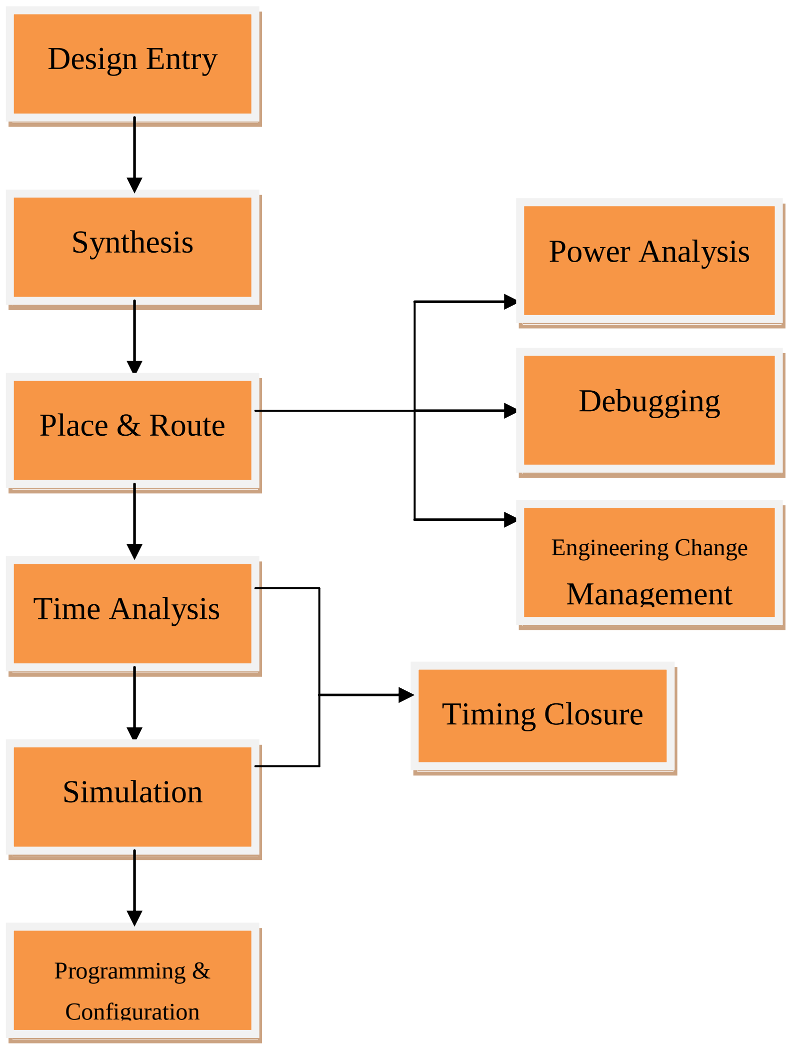


Figure 2.10: Quartus II Design Flow

2.4.2 Altera SOPC Builder

System-On-a-Programmable-Chip (SOPC) Builder is included with the Quartus II software. SOPC Builder provides an environment for design implementation using pre-designed components such as Nios II CPU for the embedded systems. In our research we have used these pre-designed components and they are interconnected with the help of default Avalon fabric and our NoC component which was imported as VHDL coded files.

2.4.3 Nios II Embedded Design Suite (EDS)

The Nios II EDS consists of Eclipse IDE and provides an environment where we can configure, program, debug and carry out simulation of Nios II CPUs. All this is carried out by using C/C++ programming language which is further compiled, linked and assembled for Nios II. It also supports in-circuit debugging and Flash Programming

2.4.4 Mentor Graphics ModelSim

Mentor Graphics ModelSim [6] is a tool which provides an environment to run the simulation tests for the VHDL or Verilog designs. It has many features to debug the problem, like assertion tests, breakpoints and in-depth signal variable simulations. It also provides us with code coverage and all these features are not well supported in Quartus II's [7] simulation engine.

2.5 Related Work

Our Routers has been designed and synthesized for an Altera Cyclone IV FPGA. This section provides a brief review of the state of the art for NoC routers. After that, we describe the Avalon-Wishbone glue logic and discuss related work in that area. Next, we look at related work in the area of NoC adapters, followed by related work that builds a NoC with the Nios II CPU and supporting software. Other areas of related work include similar routers synthesized for FPGAs and their evaluation methodologies.

Vestias et al. propose GNoC in [8], a generic router which supports a range of routing, switching and arbitration protocols. They create a tool for exploring the sharing of some decentralized components to reduce area that is based on the injection rate of ports. Unfortunately, they lock all protocols to certain values and do not explore them further. Their tool shows how they can save area when injection rates are low but does not test to see if performance is degraded.

MoCres, designed by Janarthanan et al. in [9], uses complex VCT flow control and attempts to reduce area by combining multiple components into a single component. They create multi-clock domain to enable high clock frequencies during transfers. Optimizations from XY routing in the crossbar matrix have been extended to the routing algorithm, and gave us the idea for a further arbitration unit extension. We have also used their idea of creating VHDL wrappers to simulate the stand-alone router or routing configurations to compare parameters.

Porting from Wishbone Bus to Avalon Bus was the concept given by **Xing, Xu, et al**. They have discussed regarding the glue logic between the Wishbone and Avalon interface sockets, a Wishbone compatible I2C controller was ported to the Avalon bus [10]. The glue logic was verified with simulation results. While the logic is correct for single transfers, there is much missing in the way of variable latency support and high speed Avalon block transfers. The *readdatavalid* signal is not supported in this paper and block transfers will not be queued and hence, forced into a wait state. Lastly, there is no mention of burst transfer glue logic.

A packet-switched wormhole router was implemented [11], utilizing Virtex-4 SRL16 components for FIFO implementation, which increases efficiency but decreases portability and design reuse. A Wishbone adapter was included, which supports burst transfers. Since the routers are input queued, deadlock becomes an issue and was solved by adding a separate read request buffer into the Wishbone adapters, which halts any incoming request when the buffer fills. They tested the design with a 16 switches, memories and transaction generators. The individual router was synthesized for Xilinx FPGAs with four and five ports and compared to related work.

Design and implementation of a Plesiochronous multicore system [12], a 4x4 packet-switched mesh NoC was implemented with SOPC Builder using Nios II CPUs. Multiple Stratix II FPGA boards running at 50MHz were used in order to fit the entire design, which results in an on-board throughput of 650Mbps. Inter- board communication operates at 50MBps. A software driver is used to access NoC functions within the Nios II CPUs. The system was verified by probing certain NoC components as a message traverses the network and returns to the sender, and it was found that the maximum communication rate was 434,000 Packets/second. This large difference between the theoretical bandwidth of 640Mbps is due to the large amount of time required for the packet to traverse the software routines.

In HERMES [13], a packet-switched wormhole router with input queuing was designed and analyzed. The router has four regional ports and one local port, and uses X-Y routing. 3x3 mesh NoC architecture was implemented with traffic generators attached. The buffer size and traffic patterns were analyzed and explored, resulting in overall increased performance as buffer size increased. A 2x2 NoC was synthesized targeting a Xilinx XC2V1000 FPGA.

2.6 Summary

In this chapter, the relevant background material and related previous work was presented. First, the basic concepts of FPGA technology were discussed then a short collection of concise definitions of NoC building blocks was presented. We then listed relevant concepts and theories about interfaces and signals. Finally, the Chapter concluded with a discussion of some of the previous work that is closely related to this research, and how it was used to motivate our research. In Chapter 3, a detailed description of architecture and functionality of SAF and WH routers is presented.

Chapter 3: Description of SAF and WH Router Architecture

This chapter discusses the design and implementation of Routers and Adapters explored and evaluated in this thesis. It begins with a discussion of basic functionality of the NoC router. That is followed by a discussion of the NoC router architectures, describing their components, and data flow.

3.1 Functionality

In the following sections, we discuss the functionality of the routers which includes protocols and algorithms. This will give us a basic idea of how routers differ from each other.

3.1.1 Protocols and Algorithms

NoC router protocols and algorithms govern the flow of data through the NoC network. They make decisions on where data flows, at what speed, in what order, how it is stored, etc. Therefore they directly affect performance and area consumption. Careful selection is crucial and there is much work to be done in testing existing protocols and algorithms and proposing and evaluating new ones. The following section describes protocol and algorithm choices used in SAP and WH router.

3.1.1.1 Flow Control

Both the routers used packet switched flow control (PS). In PS networks, data is separated into small blocks called packets at the core. This packet includes a header which has information about its destination. Upon creation of the packet, IP cores simply release the packet into the network where a series of interconnected routers forward the packet to its destination. PS is referred to as connectionless as there is no direct connection between communicating cores. This is an attractive choice as it allows multiple IP cores to communicate concurrently without contention.

3.1.1.2 Switching Mode

Switching mode can often be confused with flow control as it plays a large part on the flow of the packet. Switching mode is only a parameter of PS networks. This parameter determines how a packet is allocated with buffers and channels and when it will receive service. A packet is broken down into flow control units (flits). Each flit is the size of the channel. The two routers have two different switching modes (i) store-and-forward scheme and (ii) wormhole scheme. In SAF scheme, packets are buffered at each router, and the router waits for the full packet to arrive before forwarding. This prevents a single packet from blocking more than one channel at a time. The disadvantage is that it increases the buffering requirements of each router. While in WH router instead of full packet only the flit is stored and then the path is blocked for the rest of the flits. This scheme mitigates the disadvantage of SAF Router as there is no buffering required. However, after evaluating these routers we realized that each switching mode has its advantages under different applications.

3.1.1.3 Routing Algorithm [21]

The routing algorithm is used in the router and determines how the path is chosen to the packets destination. SAF Router uses XY routing algorithm known for its simplicity and low area overhead. WH Router uses source routing which is a deterministic algorithm that gives the designer a chance to determine the routing path and optimize placement with the help of floor planning. We will now briefly discuss how the XY routing algorithm and how the WH routing works.

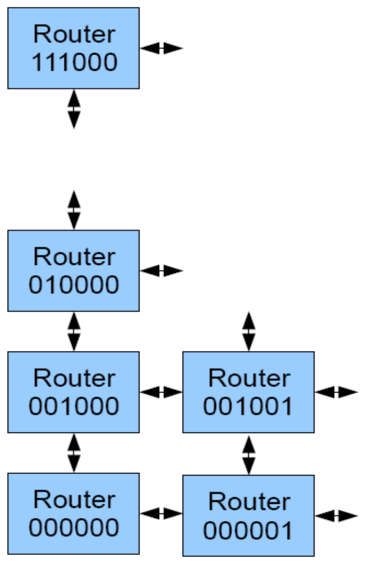


Figure 3.1: Coordinate Configuration for XY Routing [21]

In XY routing, each router is given a coordinate based on its position in the network. We restrict our mesh size to 4X4 and therefore our coordinate is 4 bits. The most significant 2 bits portrays the routers vertical displacement with 00 being the lowest (southern) router and 11 being the highest (northern) router. The least significant 2 bits portrays the routers horizontal displacement with 00 being the left most (western) router and 11 being the right most (eastern) router. Figure 3.1 shows router coordinate configuration within a mesh. A packet arrives at the router with a 16 bit header. This header contains the destination of the packet along with the type of packet. The vertical displacement is checked first. If the destination is greater than the coordinate, the packet is forward north. If the destination is lesser then the coordinate, the packet is forward south. If the destination is equal to the coordinate, then its vertical displacement is ok. The same process then occurs for the horizontal displacement. Eventually, the packet arrives at the router with the proper coordinate. At this point the packet is at the proper port and must now be forwarded to the correct destination port. Since routers in our mesh can have up to 4 ports, the least significant 2 bits of the header are used to distinguish among local ports. Figure 3.2 shows the configuration of local ports within the router.

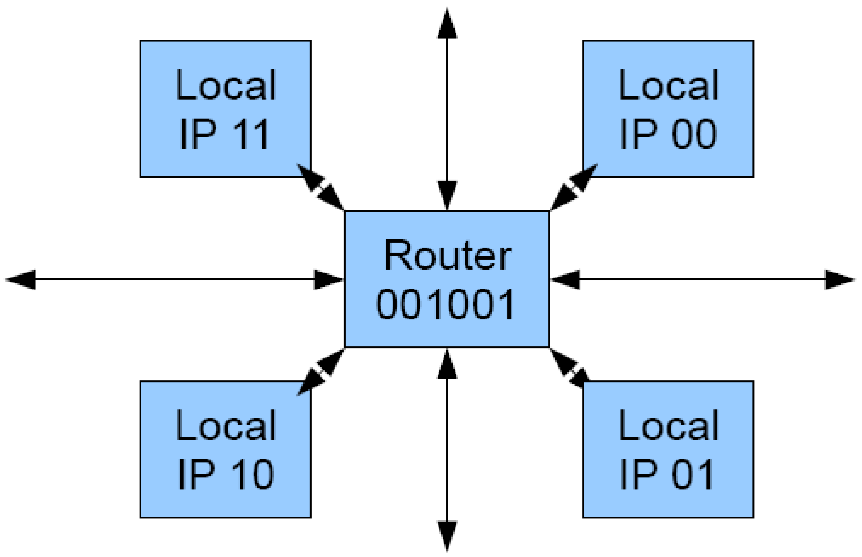


Figure 11: Configuration of Local Ports for XY Routing [21]

An important note can be made about this algorithm. Since the vertical displacement is always found first, a packet coming in from the east or west ports must already be in its proper vertical position. Therefore, a packet coming in from the east or west ports cannot be forward north or south. This observation is exploited later to optimize the area selected components.

XY routing prevents livelock from occurring. Since all packets leaving the same source and headed for the same destination will travel the same path, it also prevents having to deal with complex scenarios like packet reordering. Unfortunately, using the same logic, XY routing cannot provide any type of congestion control.

WH router uses source routing as an algorithm. In source routing the routers do not need any router coordinates to be given. Source routing completely depends on routing table which helps determining the port that packet has to use in order to progress towards destination. In source routing the number of IP cores is an important factor in determining the resource usage because more number of IP cores leads to a larger routing table. The routing table is a simple lookup table where the information of the next hop is provided to the packet after reading the source and destination address from the header flit.

In source routing the header flit consists of the source and destination address. Depending on the flit size, the size of router input buffer is determined, as it is used to store the header flit which can be consisting of two or more flits. First, the header flit is read and from there the source and destination address is extracted, the source and destination widths can be determined by designer as it is a generic parameter and can be given as input. There is a specific port assigned to the packet with respect to the source and destination addresses of the packet. The port determination is shown in Figure 3.3.

	Destination 1	Dest 2	Dest 3	Dest 4	Dest 5…..
Source 1	Port 1	Port 4	Port 1	Port 4	Port 2
Source 2	Port 2	Port 1	Port 2	Port 1	Port 1
Source 3	Port 4	Port 1	Port 1	Port 2	Port 1
Source 4	Port 3	Port 2	Port 3	Port 3	Port 3
Source 5…	Port 1	Port 3	Port 1	Port 1	Port 4

Figure 12: Port correlation in routing table

Figure 3.3 gives a clear idea of how a designer can determine the path of the packet. This is the reason why source routing is said to be a deterministic routing algorithm. Building the routing table is considered to be the most important task of the designer, as avoiding livelock, deadlock and congestion is possible by proper port allocation. Along with this the designer can do floor planning and improve the overall performance by placing densely connect IPs closer to each other.

3.1.1.4 Scheduling

Scheduling determines the order in which the data is sent and can be done by both the IP cores and routers. If 2 or more packets request the same port at the same time or while it is busy, the requested (output) port will have to make a decision on which to grant access first. This is called arbitration. Both the routers allows for some flexibility in choosing arbiter schemes. Arbitration schemes consider priority of packets in routers among the network and include static and dynamic.

In static arbitration schemes, the priority of each port is chosen during design. One of the examples is generic fixed scheme where priority is given to the north first, and degrades clockwise. While, dynamic arbitration makes a decision at run-time and is more flexible, also requires a larger area. However, dynamic schemes can avoid deadlock. One of the schemes gives priority to the port that has been busiest (sending the most requests).

When comparing the two routers we used a first come first serve arbitration scheme more details will be provided in the following chapters. This was done to make a fair comparison between the routers.

3.2 Router Implementation

We now discuss the general structure of both the routers. The SAF Router was designed with 4 ports for communication with neighboring routers, North, East, South, and West and anywhere from 0 to 4 local ports for communication to IP cores. A router with no local ports would be used just to complete a mesh or act as a congestion control unit within the network. Generic port and component design was used. Therefore, input port has the ability to forward to its own output port, although this situation could never occur. The router is decentralized meaning each port runs its own control logic and hence can request and set up concurrent connections. . The block diagram of SAF Router is shown in Figure 3.4.

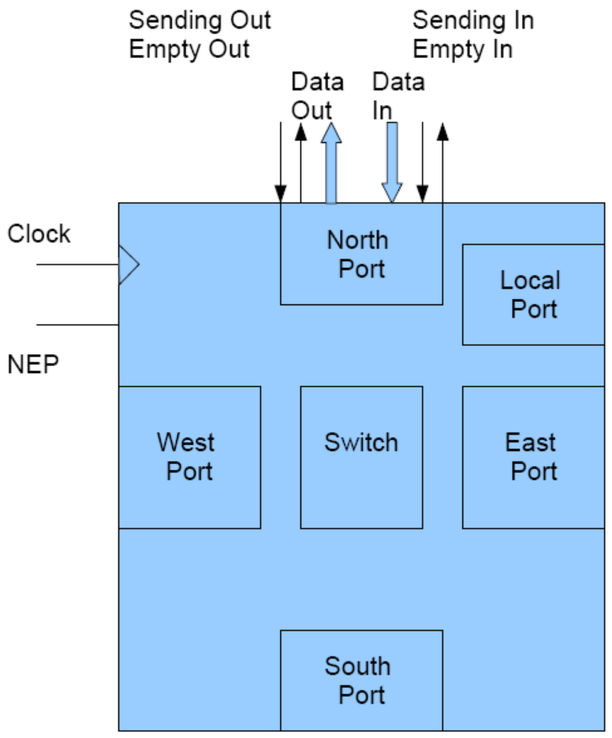


Figure 13: SAF Routers Exterior Structure [21]

The WH router was designed with the flexibility of choosing the number of ports. Generic ports give us opportunity to save area, for example the boundary routers will not need all the ports for connection. It also gives us the design flexibility to implement many topologies without any changes to VHDL code. The Block diagram of WH router is shown in Figure 3.5.

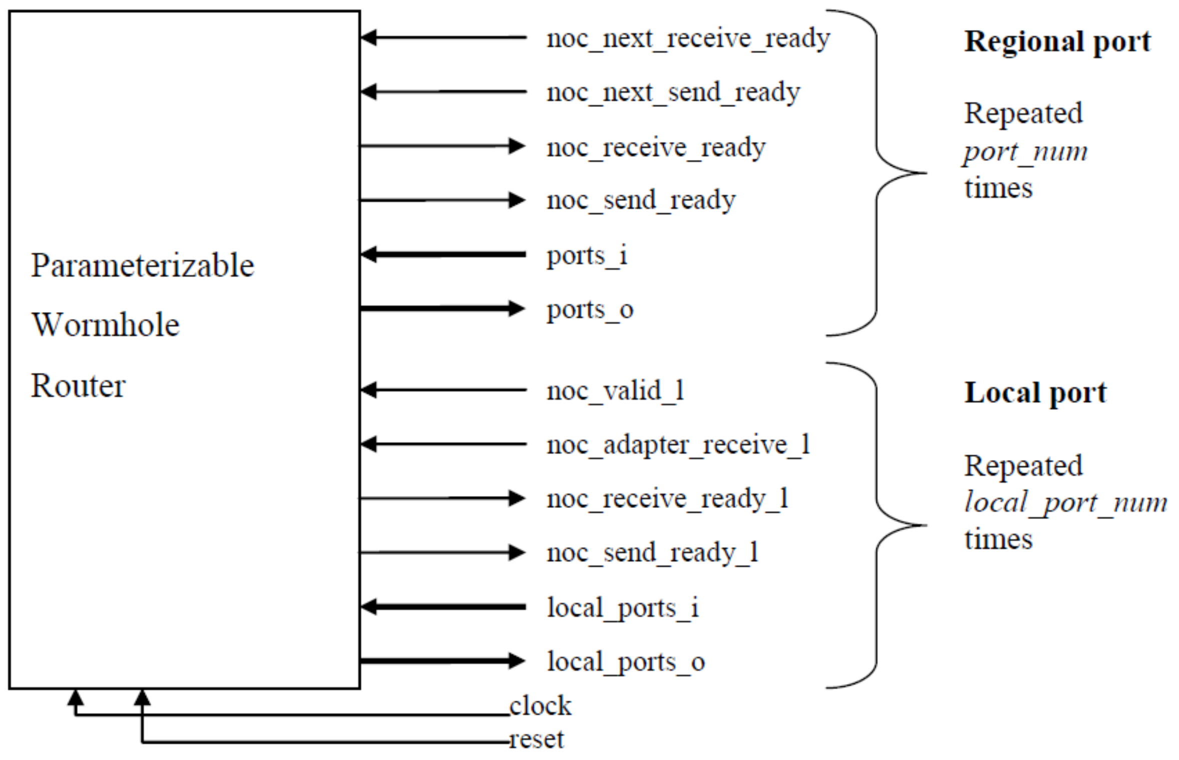


Figure 14: WH routers Exterior Structure [22]

3.2.1 Internal Structure and Data Transfer

The WH Router's internal structure consists of functional blocks such as input buffer, output buffer, arbiter, crossbar, routing table, switch, priority table and a counter as shown in figure 16. We now briefly explain the functionality of aforementioned blocks. The input buffer register is used to store the incoming flits. Once it is stored, the input and output ports are locked and a counter is started. As the counter reaches zero, the worm is completed and the entire packet is sent through the node. Now, in parallel we have to decide which node to go through for that we have a routing table which contains the path for the worm to follow and on that basis the input and output ports are locked as mentioned before. The worm travels through the router as shown by thick lines in Figure 3.6.

The most resource consuming blocks in this design are the routing table, crossbar and the input and output buffers. This has implication on how the flit size affects the area of the router.

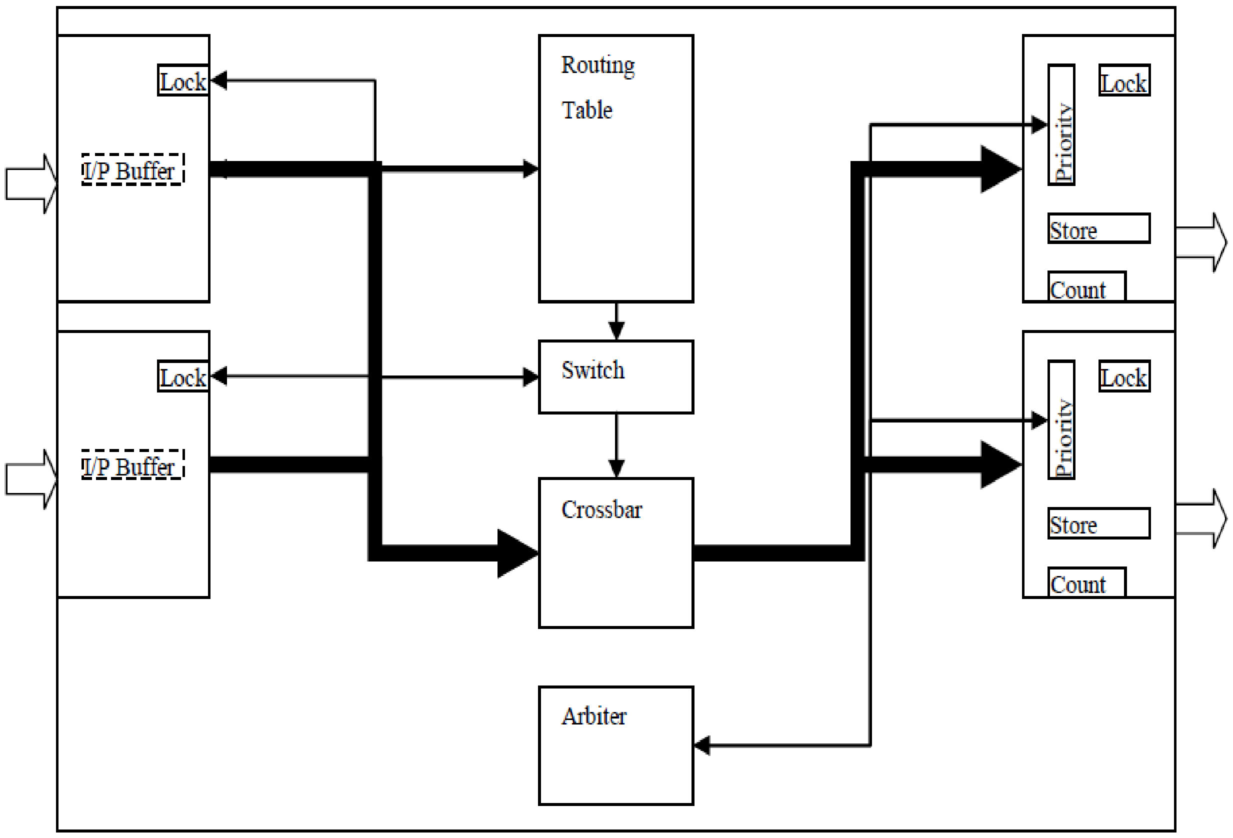


Figure 15: WH router's Internal Structure [22]

Figure 3.7 shows all the functional blocks of the SAF router. It consists of input buffer, output buffer, Partial crossbar switch, input control and output control. The input buffer stores all the flits of the incoming packet. Once the whole packet is stored header is read by the input controller. It determines the next hop and notifies the output controller of the respective output buffer. For this we have two separate sets of control signals full, empty, take in, spit out and another set is request and grant. The first set of signals get activated during communication between input buffer, input controller and also between output buffer and output controller while the other set of signals gets activated during communication between the input and output controller. Now these signals are used to build up a sequence for the data to flow inside the router.

So as soon as the input buffer is full the full signal is sent to the input controller of the respective input buffer. Along with full signal we sent the first two flits of the packet. They contain the destination address and indirectly the information of next hop. Input controller reads the header and determines which port has to be blocked and then communicate with the output controller of that respective port. Output controller will grant permission to the request. The request is only granted after the output controller confirms that the output buffer of the requested port is empty and ready to receive data through the crossbar switch this confirmation is done by the checking the empty signal coming from the output port. Now the path is blocked during this time and it works same as in worm switching technique but only during this period of the data flow. The output controller de-asserts the grant once it receives a full signal from the respective output buffer. The data flow is shown in figure 3.7 which is in form of thick blue line and control signals are denoted by thin black lines.

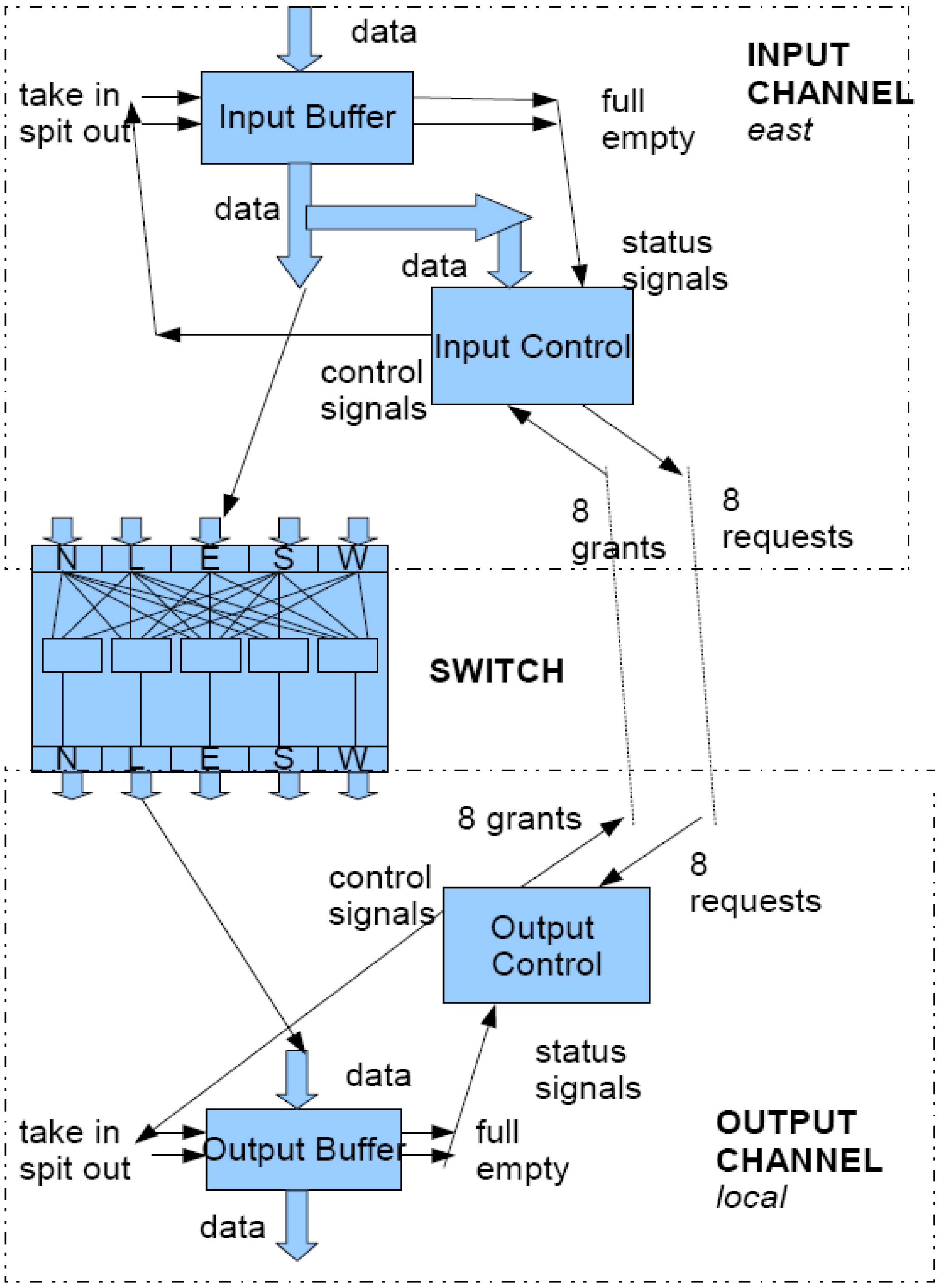


Figure 16: SAF Router's Internal Structure [21]

3.2.2 Switching Mechanism

The crossbar switch is shown in figure 3.8. It is a set of demultiplexers having an interconnection allowing all possible connections between input and output channels. The crossbar switch is used in both the routers but there were some optimizations done in SAF Router's crossbar switch as it reduced the area consumption without affecting the functionality of the system. First, it uses a partial scheme, which includes one 5 by 1 unit for each output rather than one 5 by 5 unit for all outputs, for a 5 port router. Initial design included 2 switching options, full and partial switch. Early synthesis results eliminated the full switch design because it was larger in terms of area and delay. Each output is connected to a different port. Next, there are no multiplexers in the design. The input data is connected to all partial crossbar units which will choose the appropriate data for the output. The fact that at a time, the output channel can only serve one input request is exploited here. The final optimizations are made in the partial units of the north and south. Though analysis of the XY routing algorithm, we can conclude that these units will never receive data from the east or west. This reduces the inputs of all of these units by two.

WH router on the other hand has fully functional crossbar switch as the above optimization were not appropriate for the routing scheme decided for this router. This lead to more area consumption by the cross bar switch.

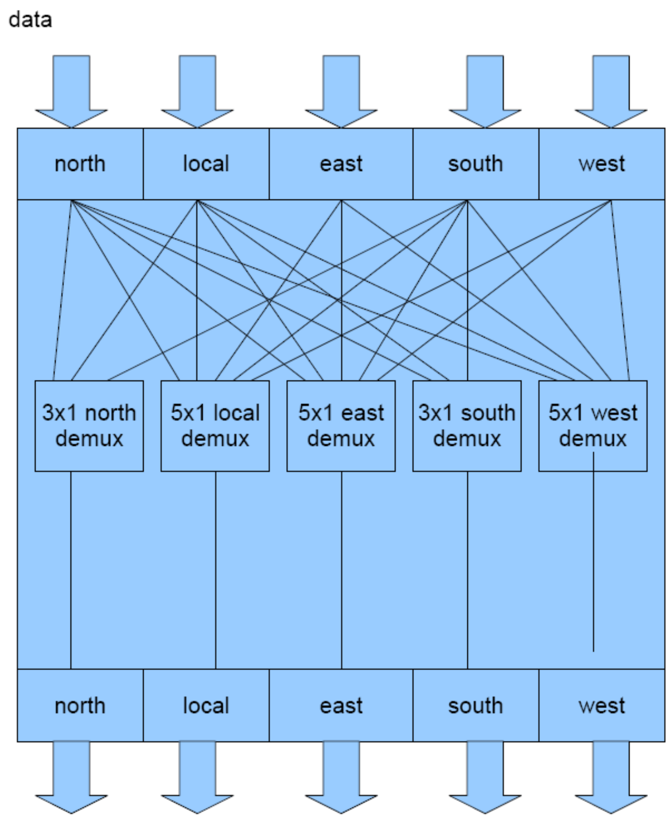


Figure 17: Architecture of Switching Fabric

3.3 Standard Sockets

This section describes the standard sockets used in our research. Standard sockets are predesigned IP cores that allow easy interface between computational blocks and routers in an NoC. For example in our research the computational blocks are designed to work on Avalon protocols while the router works on wishbone protocols so our socket converts signal between Avalon and wishbone protocols and can be used as network adapters in NoCs. This kind of design practice reduces the design cycle time and also allows designer to concentrate on the core functionality of the system. In our case the core functionality is that of the routers while the network adapters are used as standard sockets.

3.3.1 Adapter Overview [22]

The adapter acts as an interface between IP cores and the routers. Its main function is packetizing and de-packetizing. The adapter is designed for wishbone protocol [14] which is open source and supported by many IP Core vendors

The adapter works with routers designed for wishbone protocols and IP cores Designed for Avalon protocols. This adapter facilitates NoC implementation on Altera FPGAs with router running on wishbone protocols.

There are two types of adapters – Master and Slave. As illustrated in Figure 3.9, the Master adapter is responsible for receiving requests from a master component (such as a CPU) and providing the response signals. The Slave adapter is responsible for receiving the master requests and providing the slave responses.

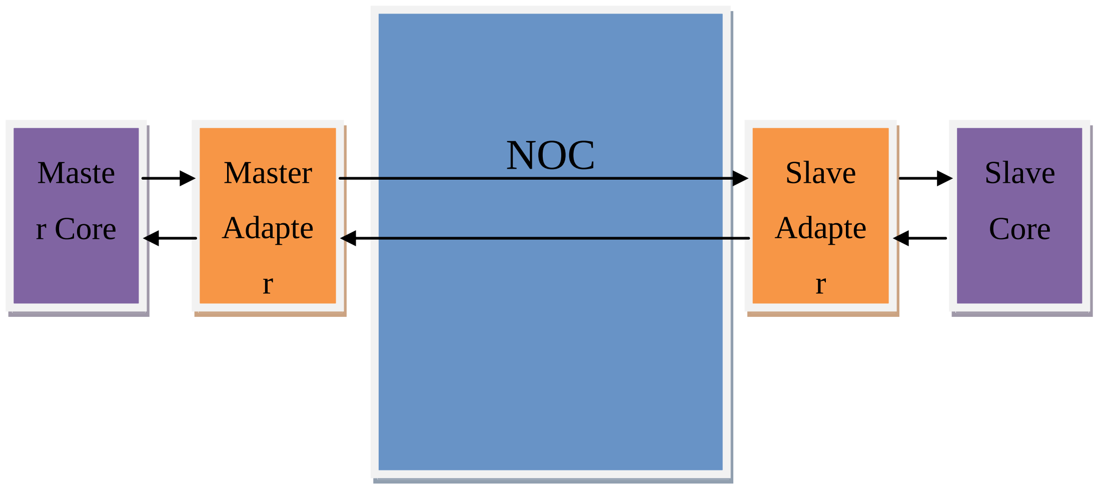


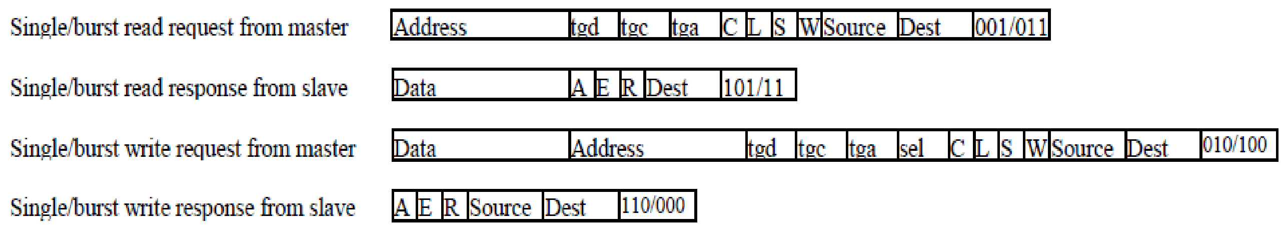
Figure 18: Adapters Overview

The adapter contains a variety of VHDL generics, offering a degree of design flexibility. The adapter is designed to be compatible with a wide range of signal widths and to conform to Avalon and Wishbone standards. Avalon interface compatibility is obtained through the use of a glue logic module. The logic utilization of the glue logic is very small, and hence negligible. These parameters are divided up into three sections: Interface, NoC and internal. Interface parameters provide flexibility with the Wishbone/Avalon interfacing. NoC parameters allow the adapter to operate in a variety of different NoC architectures. Internal parameters concern the internal operation of the adapter. Common for both adapters, data width (*WB_width*), address width (*adr_width*), address tag width (*tga_width*), cycle tag width (*tgc_width*), data tag width (*tgd_width*) and selection width (*sel_width*) are VHDL generics used to specify Wishbone interface parameters. Specific to the slave adapter, *cti_lsb* and *bte_lsb* both indicate cycle type identifier and burst type extension least significant bit locations, respectively.

NoC parameters are *flit_size, fifo_depth, src_width and dest_width. Flit_size* is the size of a flit, in bits. *Fifo depth* is the number of registers in the adapter‘s FIFOs, which allows the adapter to queue up flits if the NoC is congested. *Src* and *dest width* are the bit widths of the source and destination NoC addresses, respectively. They should both be equal, where the separate parameters are present for future optimization allowing lower bits for source addresses.

The internal parameters are *fast_burst, burst_depth, burst_tag_en, no_ack, sdram_delay* and *Avalon_bursts. Fast_burst* indicates that burst and block transfers are to be queued up using a burst buffer, thus opening request types 4 and 5. *Burst_depth* is the size of the burst buffer – this parameter is useful if there is a small flit size but large packet size (due to large data width, for example) since more requests can be queued and hence the CPU does not get stalled. *Burst_tag_en* is used to enable burst tags for Wishbone transfers – 1 to enable, 0 to disable. *No_ack* is used when there is no acknowledge signal for reads and writes – For this thesis, it is set to 1. *Sdram_delay* is used with single transfers and delays forming a packet by one cycle – this was needed for interfacing with SDRAM in single transfer mode. *Avalon_bursts* is used if Avalon block transfers are used – this distinction is required since Wishbone block transfers are different from Avalon‘s as explained in chapter 2.

Packets are made up of flits and the minimum packet size is three bits. The first three bits in a packet is always the request type, while the rest of the packet depends on the request type. The adapter analyses the request (or response) of the IP core and chooses the appropriate request type. Table 3.1 indicates all the request types and their size. In the case of this research, only request types 3, 4 and 7 are used due to the exclusive use of Avalon block transfers.



Single/burst read request from master	Address	tgd	tgc	tga	C	L	S	W	Source	Dest	001/011	
Single/burst read response from slave	Data	A	E	R	Dest	101/11						
Single/burst write request from master	Data	Address	tgd	tgc	tga	sel	C	L	S	W	Source	Dest	010/100
Single/burst write response from slave	A	E	R	Source	Dest	110/000						

Table 3.1: Request Type Design

The complete adapter is formed of five modules – *adr2dest*, *awb*, *fifo*, *master/slave sampler* and *master/slave top*. *Adr2dest* is responsible for converting the address signals into NoC destinations. *Awb* is the Avalon-Wishbone glue logic. *FIFO* is the first-in, first-out register bank used to queue incoming and outgoing flits for the adapter. The sampler is the main logic of the adapter, responsible for packetizing and de-packetizing the interface requests and responses. Finally, the top module is responsible for the hand-shaking protocol between the sampler and FIFOs, and the sampler and NoC. Each component is described below.

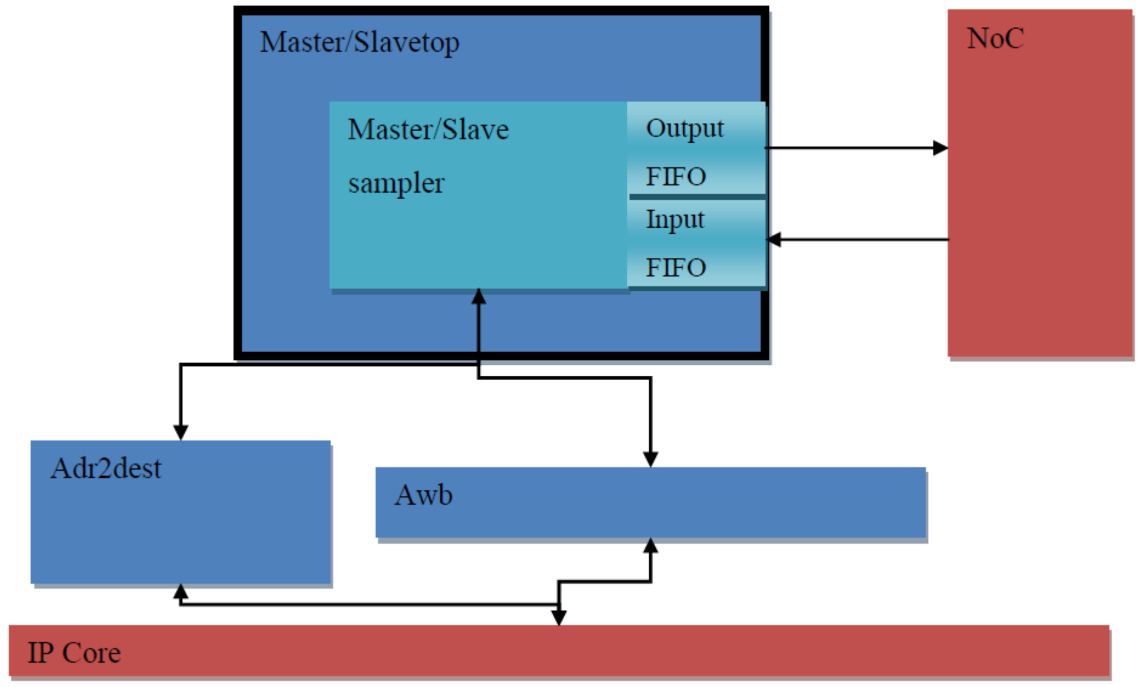


Figure 19: Adapter Internal Design Overview

Awb is the Avalon-Wishbone glue logic. It contains both the slave and master adapter interfaces, indicated with an initial *wbs/wbm* or *avs/avm* for Wishbone and Avalon, respectively. Most of the logic is simple name changes for the signals to make building in SOPC Builder [15] easier and the component interface conversion is bidirectional. There is additional clocked logic used to delay the de-assertion of Avalon read/write signals by one cycle since de-asserting these signals is not allowed immediately when the *wait_request* signal is de-asserted as well. The connections are illustrated in Figure 3.11.

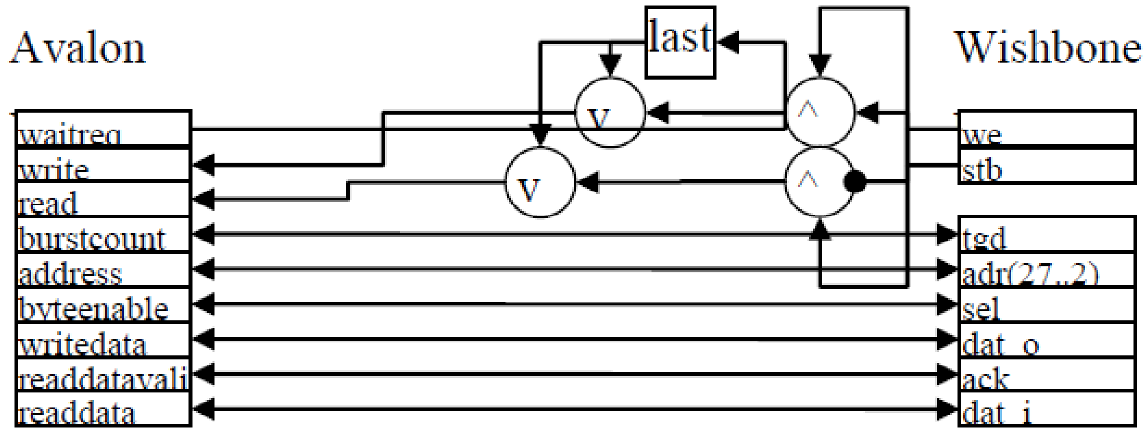


Figure 3.11: Avalon Wish bone Glue Logic

FIFO is an array of registers, responsible for queuing flits into and out of the adapter. An extra overflow register is provided to help stop issues with control signal latency. The FIFO‘s depth is specified with VHDL generics. Empty and full are used to indicate when the FIFO can be read or written to.

The samplers have two unique versions – master and slave. The operation of the samplers is based around the idea of sampling and saving bus signals, yet the operation of the adapters is more complicated than this. Simply sampling the bus at specific intervals, placing in a packet and sending over the network would cause a lot of wasted packets being sent since the transactions are predictable. The Wishbone operation handles three types of transactions: Single, block and burst. Single transactions in the adapter perform one complete transaction at a time. Block transactions is essentially the same as single transactions for Wishbone, but with a key difference in that the acknowledge signal is predicted to be asserted for write requests and is done so artificially, thus increasing the speed of the adapter. Read transactions for Wishbone block transfers operate the same as single transfers, since a response is required and cannot be predicted. Burst transfers include the cycle and address tags (CTI and BTE, respectively) so read requests can be sped up, similar to how block writes work. The Avalon block transfers operate differently in that requests can be queued without an acknowledgement for the previous request. Thus, a specific VHDL generic (*Avalon_bursts*) is used to switch the adapter into Avalon‘s block request queuing mode. The operation of the slave adapter in this mode is illustrated in Figure 3.12, and the master adapter is shown in Figure 3.13.

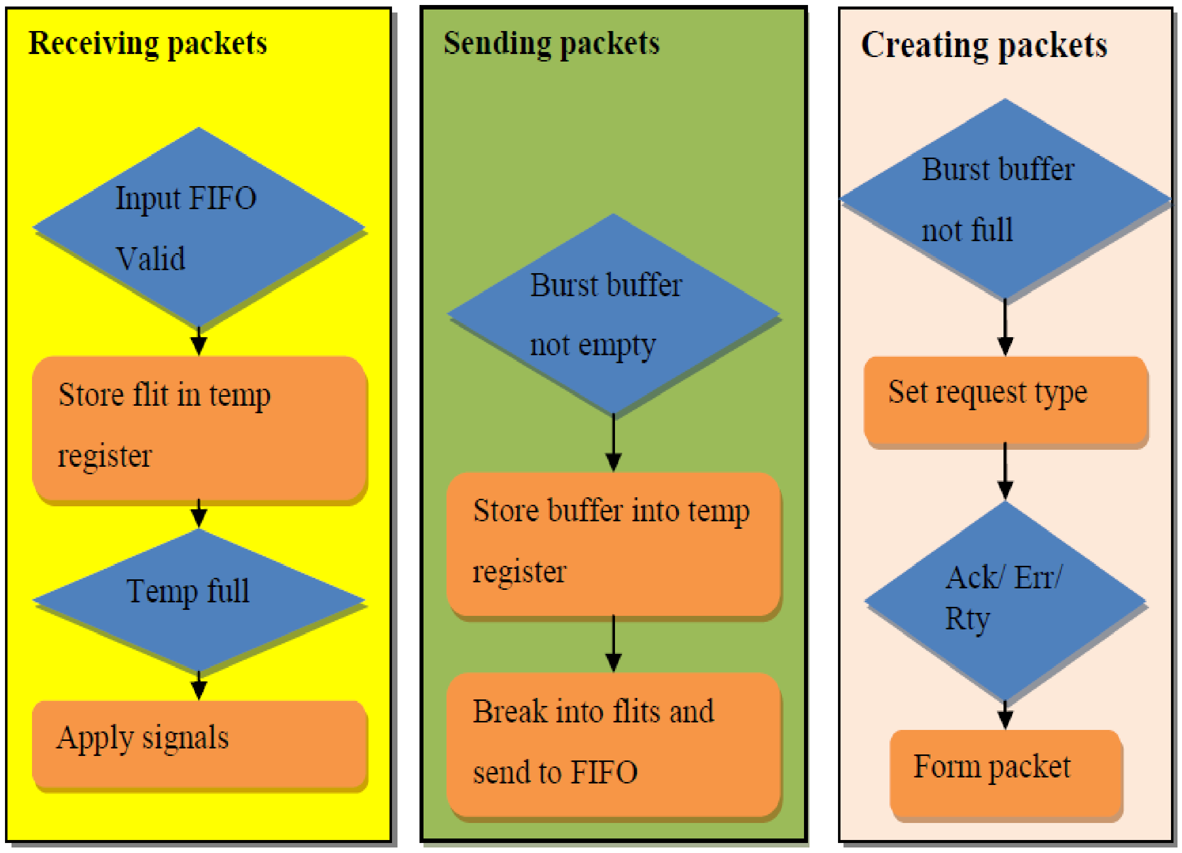


Figure 20: Slave Sampler Process Flow Chart

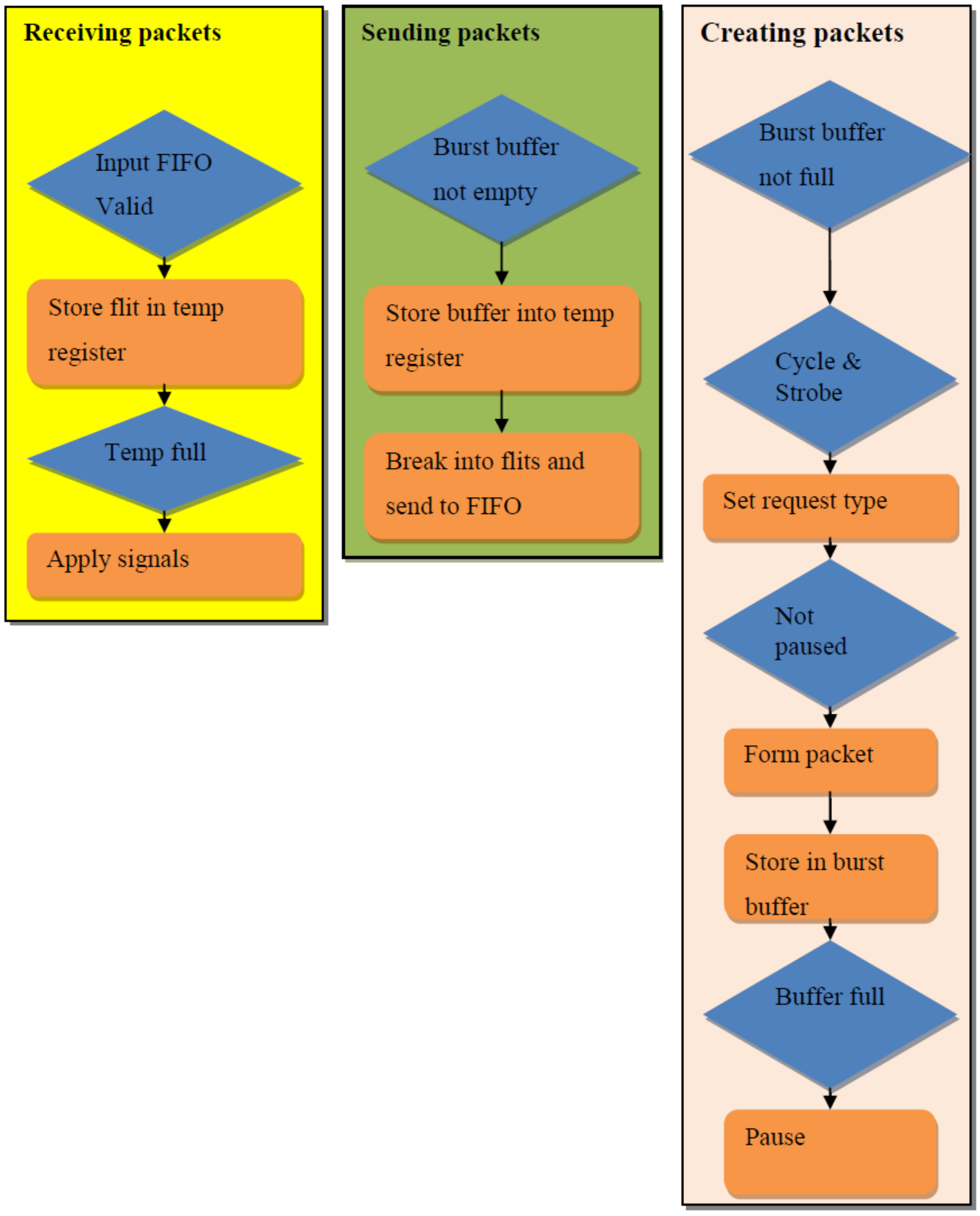


Figure 21: Master Sampler Process Flow Chart

The Top modules (*mastertop* and *slavetop*) connect the samplers with input and output FIFOs. The top modules are also responsible for providing the handshaking protocol between the FIFOs and the NoC. This is done with two flip-flops –*wait_for_NoC_ack* and *NoC_sent*. *Wait_for_NoC_ack* is set when an output FIFO sends a flit and is cleared when the NoC acknowledges (via. Deasserting the *receive_ready_*signal). *NoC_sent* is similar where it only writes the first flit to the input FIFO until *NoC_send* is deasserted.

3.4 Summary

In this chapter, we discussed architecture and functionality of SAF and WH routers. The chapter concluded with a detailed description of the NoC adapter. In Chapter 4, experimental framework will be discussed.

Chapter 4: NoC Implementation and Evaluation Framework

This chapter begins with a discussion of the real world benchmark design used to test our NoC system. This design was used to evaluate two mesh NoCs implemented using SAF and WH routers. Chapter 4 concludes with discussion of how test revealed some flaws in SAF Router.

4.1 Real World Benchmark Design

This thesis aims to compare two different router designs SAF and WH router with the help of real world traffic from a practical system. NoCs have been previously tested by using real benchmark design [16], however they were not evaluated and compared using different routers. The benchmark design used was a multi-processor design example chosen from Altera‘s website. We did some modifications to make the system suitable for simulation and also decrease the overall system size. The modified multiprocessor example, shown in Figure 4.2, contains three Nios [15] II/f soft core CPUs, three 1ms timers, 16MB of flash memory (AMD29LV128M123R_BYTE), a mutex, 64KB of on-chip RAM, 1KB of message buffer RAM (on-chip), 256 Mbit (16 bit) SDRAM (Nios Development Board, Cyclone II), a JTAG UART interface module, a sysid module and an LED PIO.

This benchmark is implemented on 4x4 mesh NoC as shown in figure 4.1. Each block in the figure represents either a standalone router or a router connected to an IP core through an adapter. The system starts by booting from a flash memory which is 16MB in our design. Each CPU starts reading data and instruction code from the DDR-SDRAM. Mutex lock is used to provide exclusive access to the message buffer by a single CPU at a time. CPUs try to acquire the mutex lock simultaneously. The Acquisition depends upon timers; each timer is responsible for sending interrupt requests to the CPUs. CPU 1 is responsible for reading the message buffer, clearing it and sending the message to the JTAG UART interface module. CPU 2 and 3 sends signals to LED and PIO along with sending the message to the message buffer. The program code is contained within the SDRAM for all three CPUs, in separate locations. All the CPUs have their reset vector in FLASH memory. CPU 1‘s interrupt vector is contained within the on-chip memory module, while CPU 2 and 3‘s interrupt vectors are in the SDRAM. The functionality of this benchmark design generates enough traffic that can be used to evaluate the routers in the NoC. This is more useful compared to using synthetic traffic generators

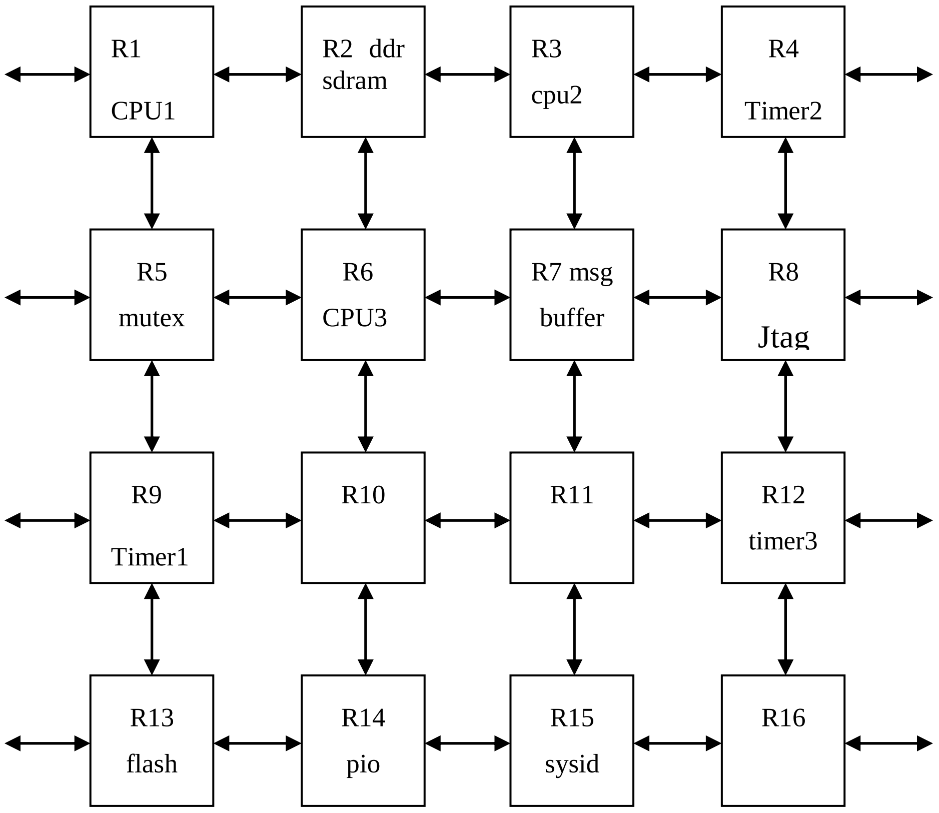


Figure 4.1: Mapping of Benchmark Design Components on 4x4 NoC

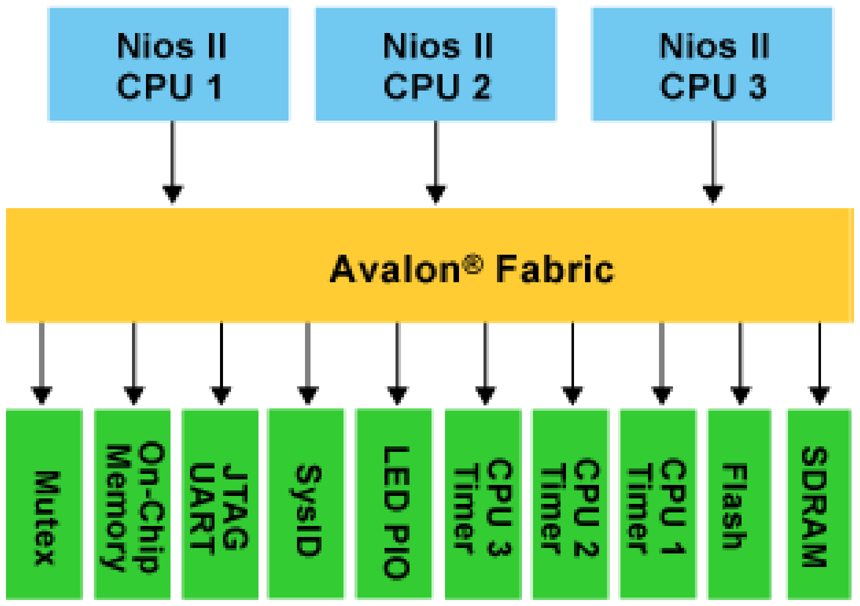
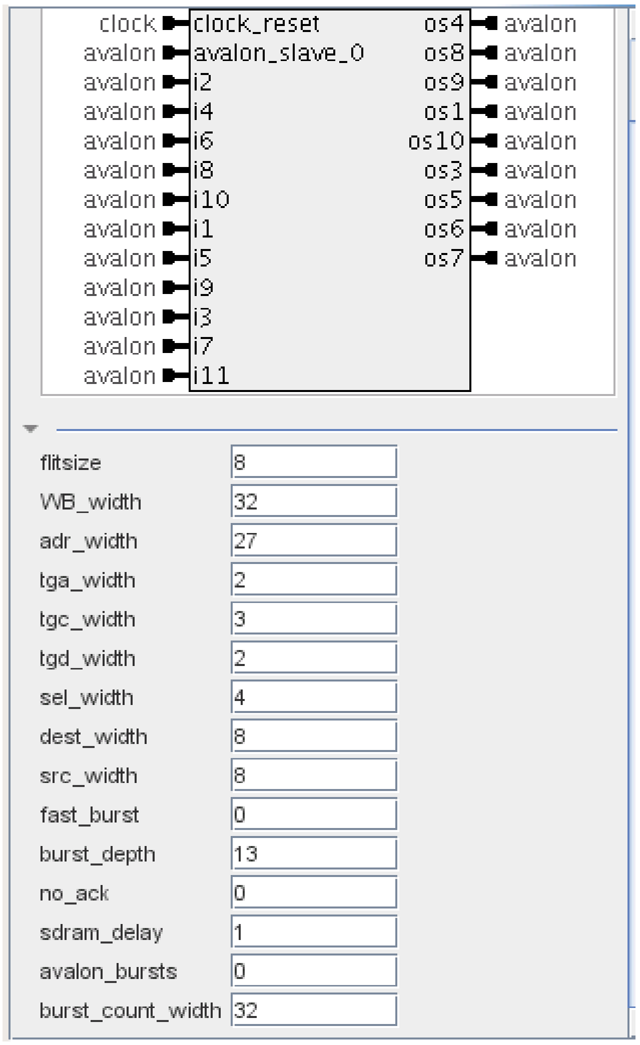


Figure 4.2: Altera Multiprocessor Design Example

4.2 Implementation of NoC in SOPC Builder---- Add description of SOPC

SOPC Builder is a CAD tool included in Quartus II. It facilitates interconnection of pre-designed components such as computational blocks, memories, bridges, processors etc .Thus designers can use SOPC builder to develop a system without worrying about generating the VHDL files for interfacing the components. In order to include the whole of our 4x4 mesh NoC we generated a single component named ***Samnrouter*** and its parameters are shown in figure 4.3.

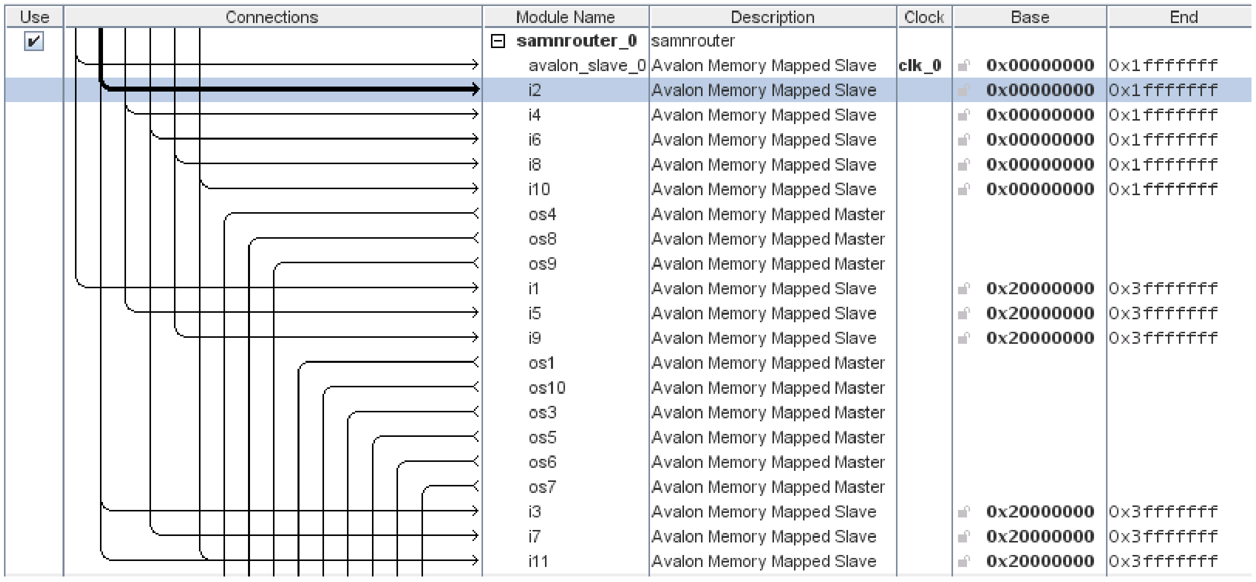
The components in SOPC builder are connected using Altera's Avalon interface which has separate interface signals for the master and slave components. While the NoC runs on wishbone protocol which does not have different interface signals for master and slave. Hence, we need to differentiate master and slave interfaces in ***Samnrouter*** as shown in figure. This will avoid any confusion while debugging during simulation as it is difficult to figure out the difference between master and slave interface signals in the vector forms.



flitsize	8
WB_width	32
adr_width	27
tga_width	2
tgc_width	3
tgd_width	2
sel_width	4
dest_width	8
src_width	8
fast_burst	0
burst_depth	13
no_ack	0
sdram_delay	1
avalon_bursts	0
burst_count_width	32

Figure 4.3: NoC Parameters in SOPC Builder

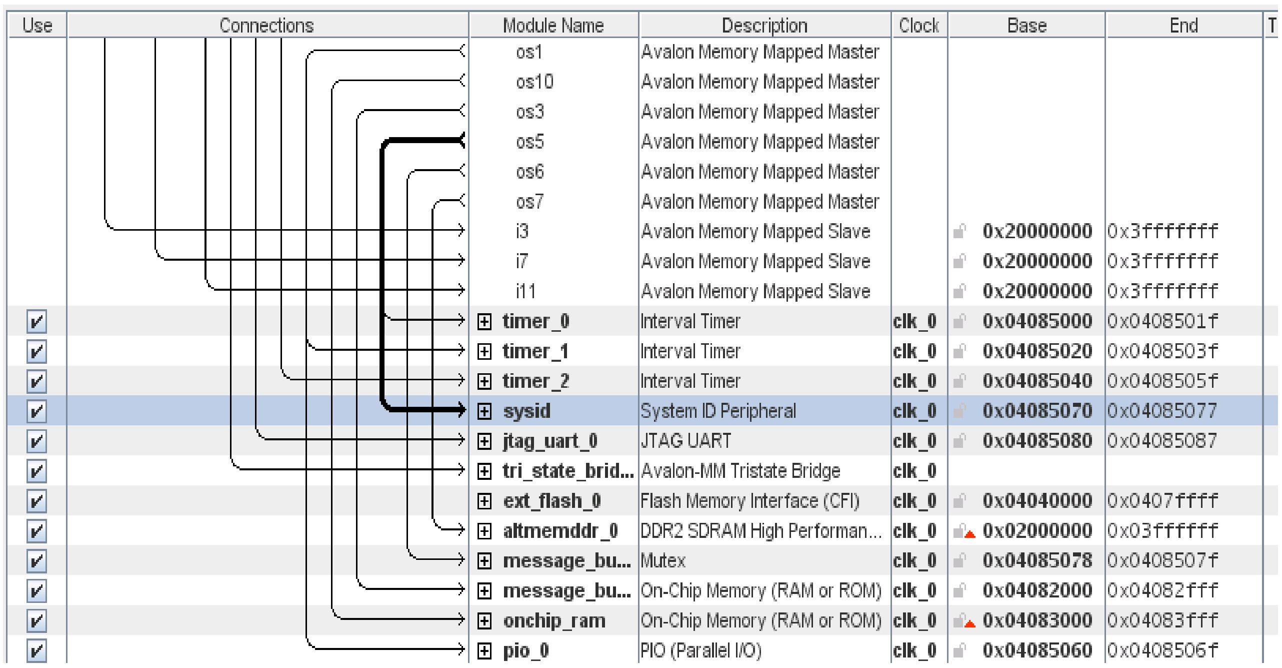
The component is then added to the system and the interfaces are connected. Figure 4.4 shows the torus NoC implemented in SOPC Builder. Each address is assigned manually, where the address bus from the Nios II is 32 bits and the address bus from the NoC is 27 bits. This means the upper 5 bits are ignored by the NoC but are used by the Avalon fabric‘s arbitration. Since masters are connected to their own buses, then master adapters can have the same addresses.



Use	Connections	Module Name	Description	Clock	Base	End
✓		samnrouter_0	samnrouter			
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00000000	0x1fffffff
		i2	Avalon Memory Mapped Slave		0x00000000	0x1fffffff
		i4	Avalon Memory Mapped Slave		0x00000000	0x1fffffff
		i6	Avalon Memory Mapped Slave		0x00000000	0x1fffffff
		i8	Avalon Memory Mapped Slave		0x00000000	0x1fffffff
		i10	Avalon Memory Mapped Slave		0x00000000	0x1fffffff
		os4	Avalon Memory Mapped Master			
		os8	Avalon Memory Mapped Master			
		os9	Avalon Memory Mapped Master			
		i1	Avalon Memory Mapped Slave		0x20000000	0x3fffffff
		i5	Avalon Memory Mapped Slave		0x20000000	0x3fffffff
		i9	Avalon Memory Mapped Slave		0x20000000	0x3fffffff
		os1	Avalon Memory Mapped Master			
		os10	Avalon Memory Mapped Master			
		os3	Avalon Memory Mapped Master			
		os5	Avalon Memory Mapped Master			
		os6	Avalon Memory Mapped Master			
		os7	Avalon Memory Mapped Master			
		i3	Avalon Memory Mapped Slave		0x20000000	0x3fffffff
		i7	Avalon Memory Mapped Slave		0x20000000	0x3fffffff
		i11	Avalon Memory Mapped Slave		0x20000000	0x3fffffff

Figure 4.4: SAF Router based NoC Implemented in SOPC Builder

The slave components are assigned addresses manually, but each component must have a unique address range in the range of 27 bits. Figure 4.5 illustrates the NoC connected to the slave components and their respective assigned addresses.



Use	Connections	Module Name	Description	Clock	Base	End	T
		os1	Avalon Memory Mapped Master				
		os10	Avalon Memory Mapped Master				
		os3	Avalon Memory Mapped Master				
		os5	Avalon Memory Mapped Master				
		os6	Avalon Memory Mapped Master				
		os7	Avalon Memory Mapped Master				
		i3	Avalon Memory Mapped Slave		0x20000000	0x3fffffff	
		i7	Avalon Memory Mapped Slave		0x20000000	0x3fffffff	
		i11	Avalon Memory Mapped Slave		0x20000000	0x3fffffff	
✓		⊞ timer_0	Interval Timer	clk_0	0x04085000	0x0408501f	
✓		⊞ timer_1	Interval Timer	clk_0	0x04085020	0x0408503f	
✓		⊞ timer_2	Interval Timer	clk_0	0x04085040	0x0408505f	
✓		⊞ sysid	System ID Peripheral	clk_0	0x04085070	0x04085077	
✓		⊞ jtag_uart_0	JTAG UART	clk_0	0x04085080	0x04085087	
✓		⊞ tri_state_brid...	Avalon-MM Tristate Bridge	clk_0			
✓		⊞ ext_flash_0	Flash Memory Interface (CFI)	clk_0	0x04040000	0x0407ffff	
✓		⊞ altmemddr_0	DDR2 SDRAM High Performan...	clk_0	0x02000000	0x03ffffff	
✓		⊞ message_bu...	Mutex	clk_0	0x04085078	0x0408507f	
✓		⊞ message_bu...	On-Chip Memory (RAM or ROM)	clk_0	0x04082000	0x04082fff	
✓		⊞ onchip_ram	On-Chip Memory (RAM or ROM)	clk_0	0x04083000	0x04083fff	
✓		⊞ pio_0	PIO (Parallel I/O)	clk_0	0x04085060	0x0408506f	

Figure 4.5: SAF Routers NoC Connected to Slave Components

Once the component is created, a Tcl script is automatically created by SOPC Builder. This script must be manually modified in order to make the master adapters act as bridges so the reset and exception vectors can be set in the Nios II CPUs. Figure 4.6 illustrates the concept of bridges, where the Nios II see's a master adapter as being directly connected to a memory module. Since each Nios II has their reset and exception vectors pointing to different memory components, and that an interface can only bridge to one other component, it follows that there must be two adapters – one for each vector. If the reset and exception vectors pointed to the same memory module, then only one adapter would be needed. For each master adapter, the *set_interface_property bridgesToMaster* parameters must be modified so they contain the slave adapter's name.

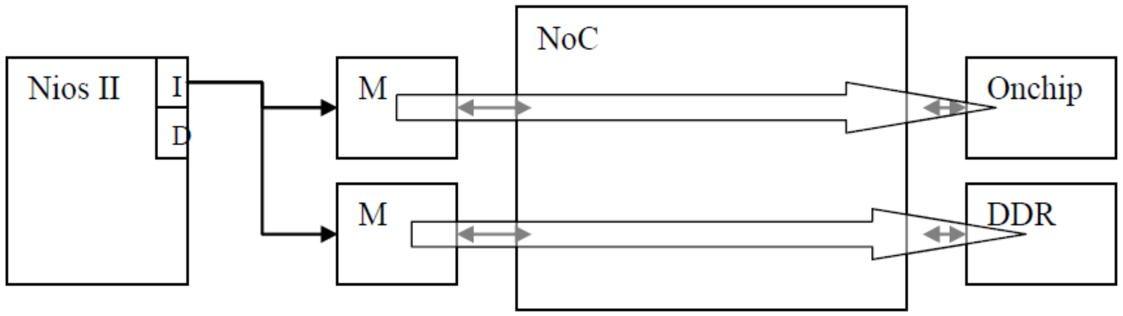


Figure 4.6: Bridging Example

The last issue involves the number of adapters for each Nios II CPU. It was set to 4 for each Nios II adapter so that each Nios II bus (data and instruction) gets its own adapter.

4.3 Nios II Programming

Before getting into details about the benchmark program, an issue with Nios II EDS [17] must be addressed. Since each program resides in different portions of the same memory block and that Nios II EDS overwrites the data block when compiling the code for each CPU, a script was set up to copy and concatenate the program files after each compile. Each CPU attempts to acquire a mutex lock, which results in them writing an incrementing counter to the message buffer. The counters stop at 5, after which no more messages are sent from that CPU. The three CPUs are numbered one to three, where CPU1 is responsible for clearing the message buffer and writing to the message to UART. CPUs 2 and 3 do not clear the message buffer or write to UART, but they write to the PIO. They have the exact same code, but different program locations in the SDRAM. The timers interrupt their respective CPUs, which cause them to attempt to acquire a mutex lock. Figure 4.7 illustrates the flowchart of the programs.

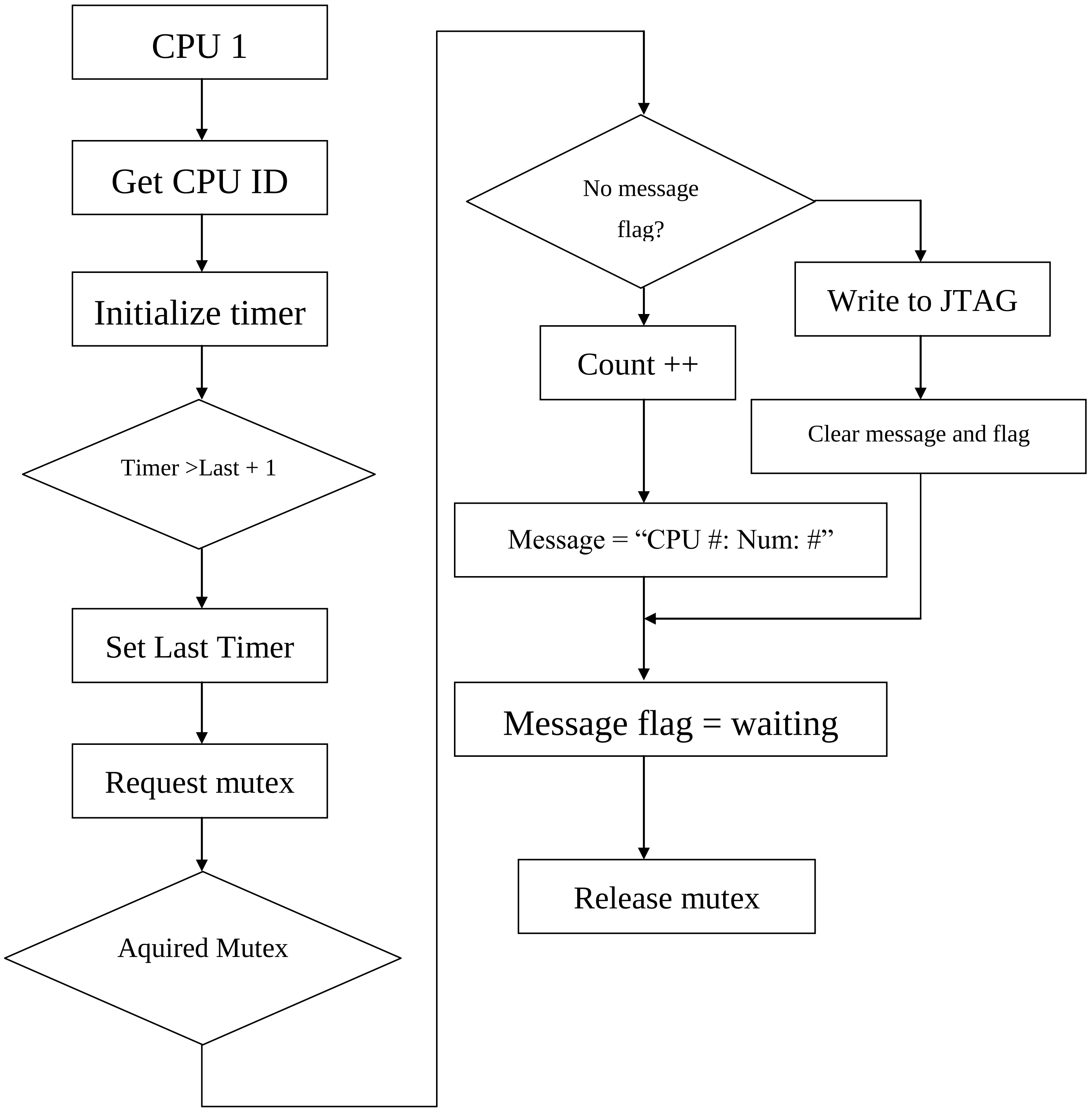


Figure 4.7: CPU 1 Benchmark Flowchart

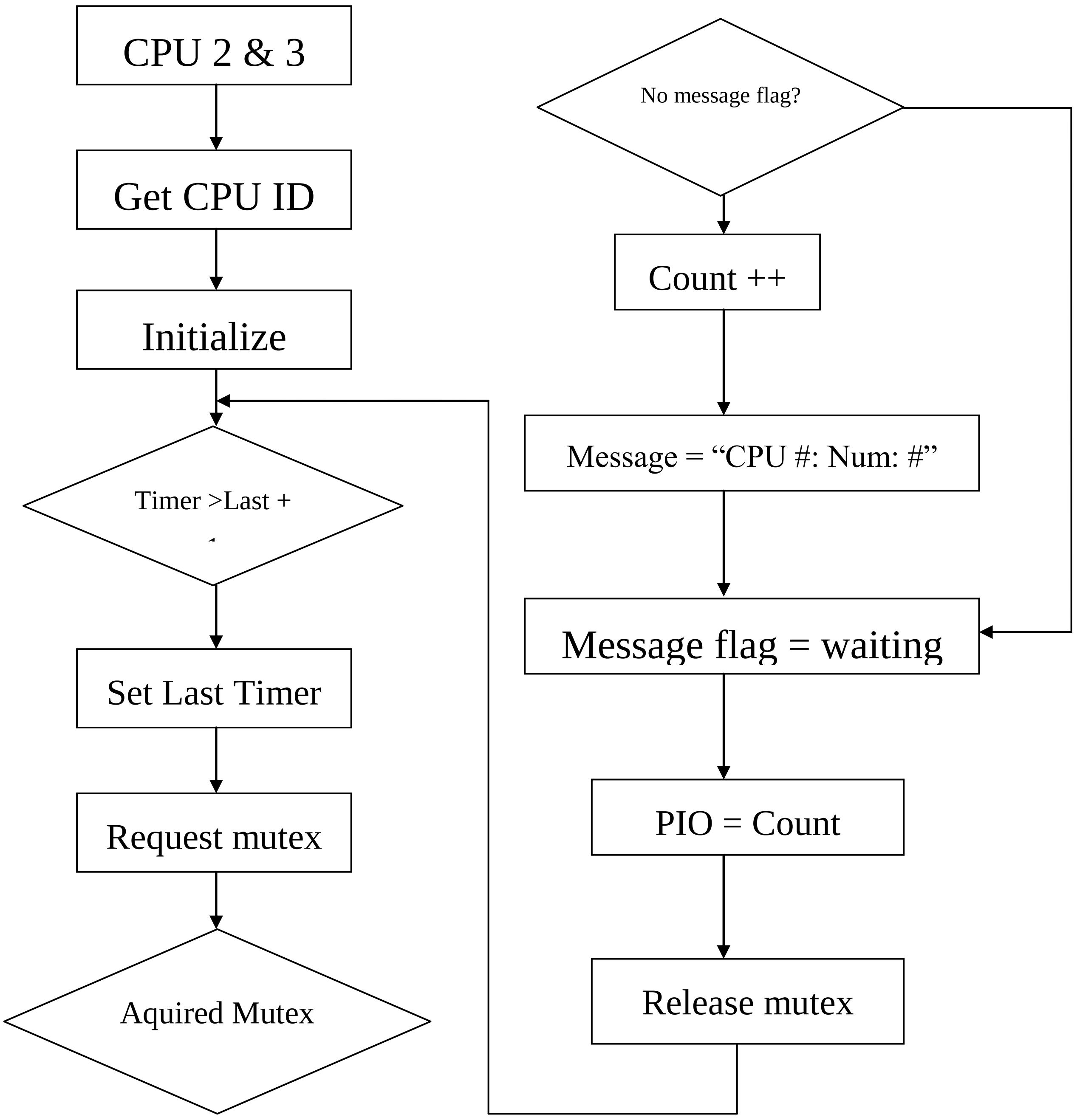


Figure 4.8: CPU2 and CPU 3 Benchmark Flowchart

4.4 Modelsim Simulation

Modelsim [18] provides good simulation and debugging capabilitites, and hence is perfect for the purposes of this research. Using Altera‘s built-in scripts, the program code is loaded into Modelsim and the automatically generated project files are used. The JTAG UART module outputs its messages to Modelsim‘s console, which is then used as a basis for simulation end-time. Once each CPU outputs its 5 messages (CPU #: Num: #), the runtime is recorded at the final write operation to SDRAM. Figure 4.9 shows a sample of the WH Router regional handshaking protocol and Figure 4.10 shows a sample of the SAF Router as seen in Modelsim. At simulation time indicated by label 1 figure 4.9, the router sets the send bit to high on port number two. Four cycles later, at simulation time indicated by label 2 figure 4.9, there is a response on the receive signal from port number two, indicating that it has received the flit. The router de-asserts the send signal on port two and, one cycle later, reasserts it to send another flit. This is just an example of what how Modelsim is useful in verifying the operation of the system.

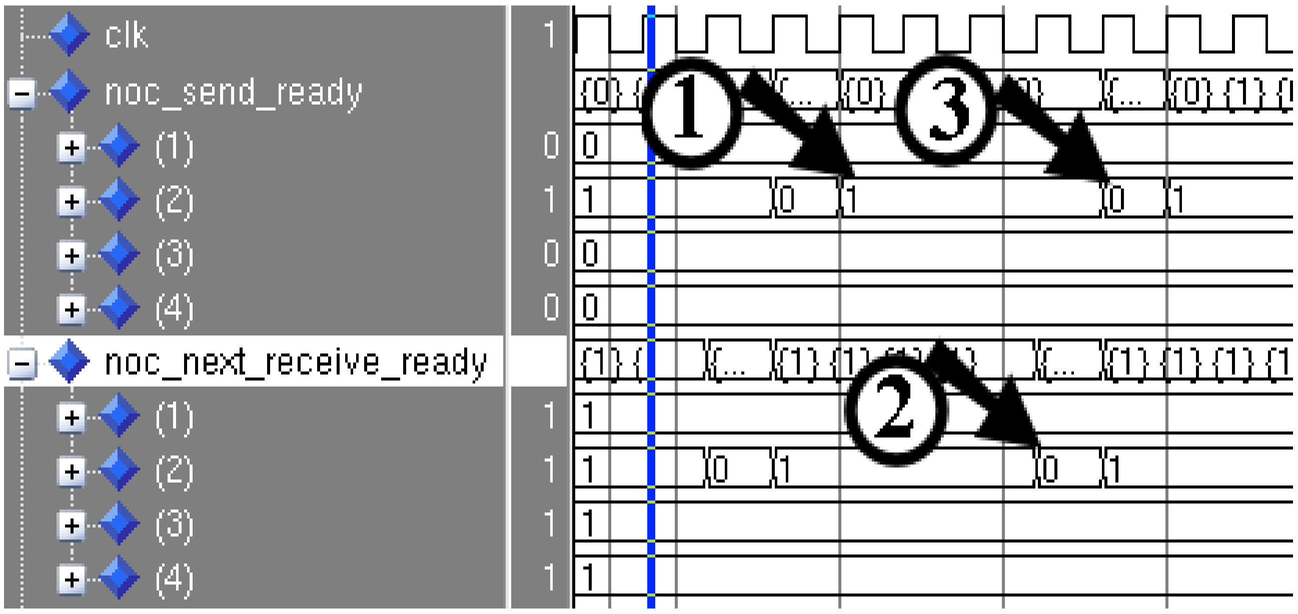


Figure 4.9: WH Routers Regional Handshaking

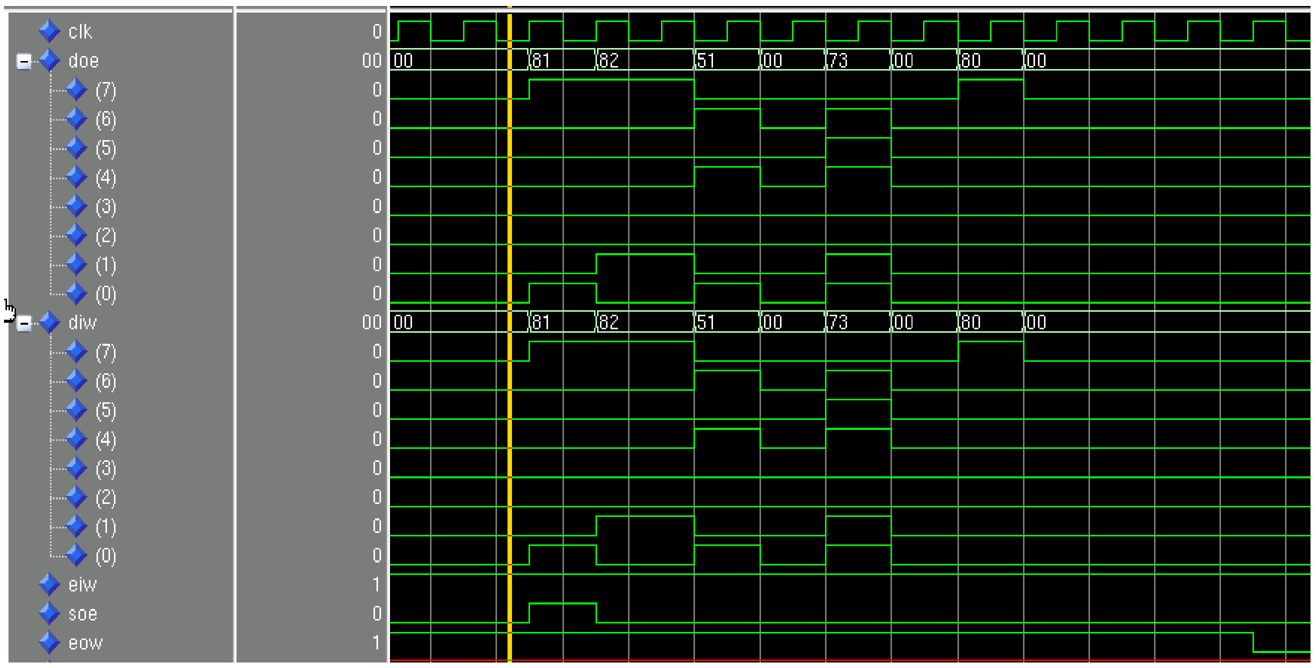


Figure 4.10: RF Routers Regional Handshaking

4.5 Re-design of SAF Router

After testing SAF Router with our benchmark design we discovered some flaws in the design of the router. These flaws were detected when we used traffic generated by a real world design. SAF router was designed with an assumption that the number of flits in the packets will be constant. That is not the case for the real benchmark design. So we made changes in SAF router by adding a counter which keeps track of packet size throughout the network and accordingly we made changes to the components. The design of input and output buffer changed significantly which increased the logic consumption. Since buffers are used in every ports this lead to significant increase in area of the router. This shows the importance of testing NoCs using real world design rather than synthetic traffic generators.

4.6 Summary

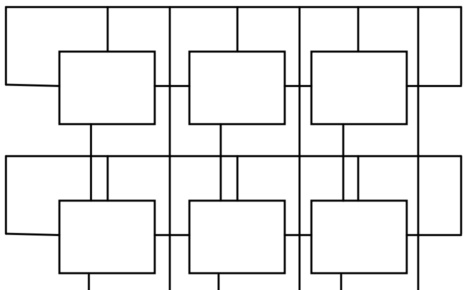
This chapter discusses the NoC implementation and evaluation using the altera SOPC Builder tool. Issues involved in implementing the NoC within this system as well as the details of the system operation were discussed. It concluded with a discussion of the ModelSim evaluation environment used in this thesis. Chapter 5 will discuss comparison of two Mesh NoC’s using SAF and WH routers.

Chapter 5: Experimental Comparison Results

This chapter presents a quantitative comparison of SAF and WH routers using a real world benchmark design. These routers were implemented on a 4X4 mesh NoCs and they are compared using the following evaluation metrics: area, power consumption, maximum clock frequency and throughput.

5.1 Mesh Topology

There are many regular topologies that can be used to connect the routers in a NoC. For example, a 3X3 mesh, 3X3 tarus and 10 node ring topologies are shown in figure 5.1 and 5.2 respectively. The choice of a topology depends on number of factors including the application for which NoC based system is implemented. Since, our objective is to compare two routers we have chosen a simple 4X4 mesh NoC topology to implement our benchmark design. Implementation of mesh topology and mesh routing algorithm for both the routers is relatively easy compared to other topologies.



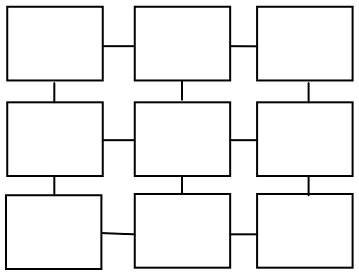


Figure 5.1: Mesh and Torus Topology

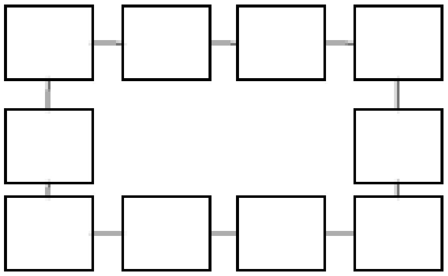


Figure 5.2: 10-node ring topology

5.1.1 Placement of the IP Cores

When mapping the components of the benchmark design on mesh NoC a placement is needed which assigns closely connected components together in order to minimize congestion. After analyzing the connection requirements the placement selected for mapping benchmark design components to mesh NoC is shown in Figure 5.3. Note that routers 10, 11 and 16 are pure routing nodes with no components attached.

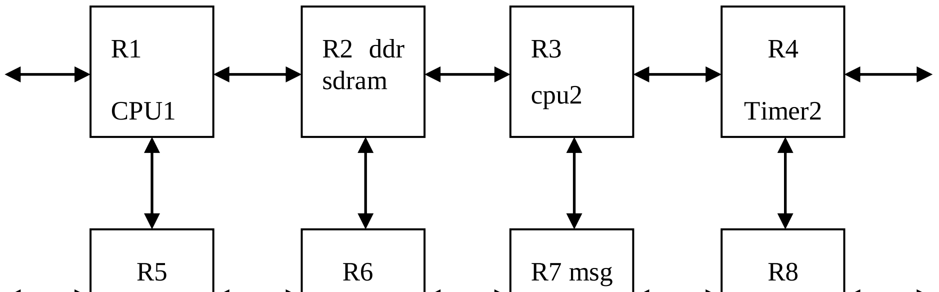


Figure 22.3: Placement and Routing

5.2 Design Space Exploration and Comparison of SAF and WH Routers

The two routers were compared by synthesizing the NoC for different values of parameters which are flit size and number of ports. The synthesis and comparison results are discussed below.

5.2.1 Redesign the SAF router

As discussed in section 4.5 SAF router had to be redesigned to handle real world traffic. The increase in area of SAF router after this design change is shown in figure 5.4

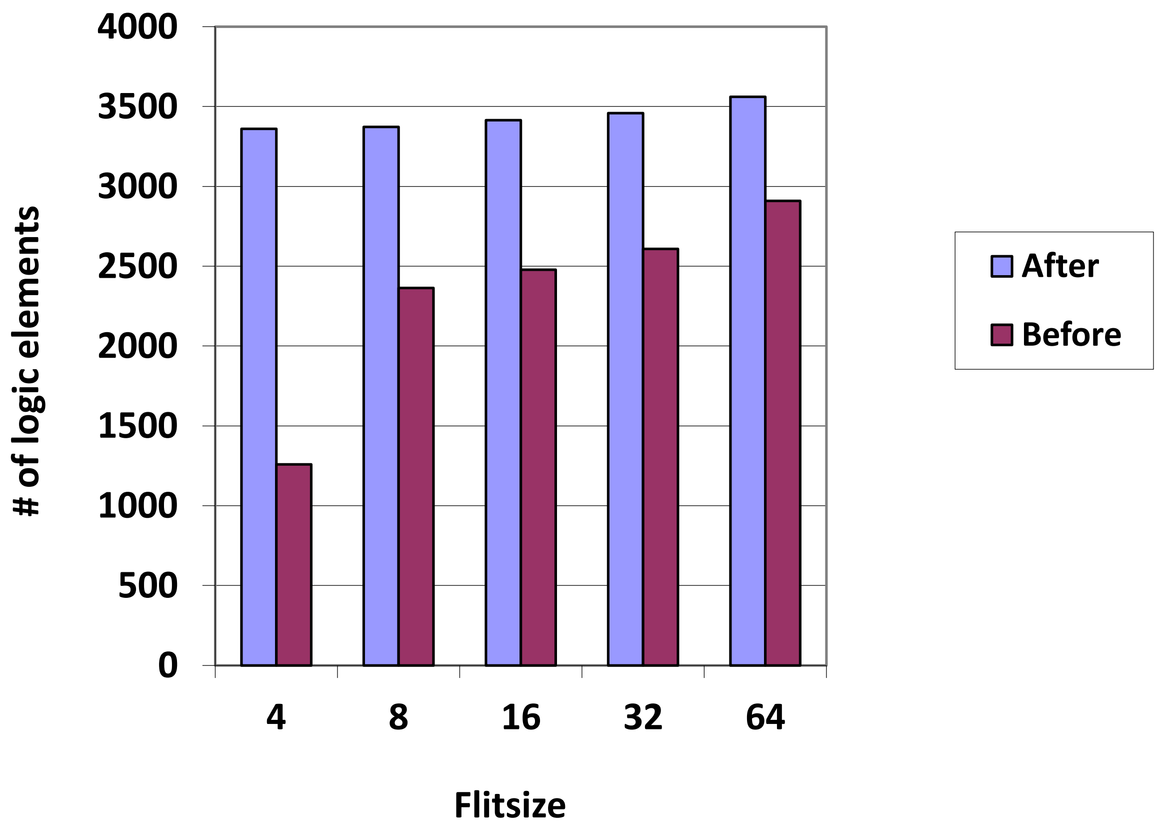


Figure 5.4: Comparison of SAF router area consumption before and after redesign

5.2.2 Area Results

We have compared the area for WH Router and SAF Router by varying their flit size and number of local ports. Flit size is the size of smallest unit of the packet that traverses through network. The number of local Ports is used to connect IP’s to the routers.

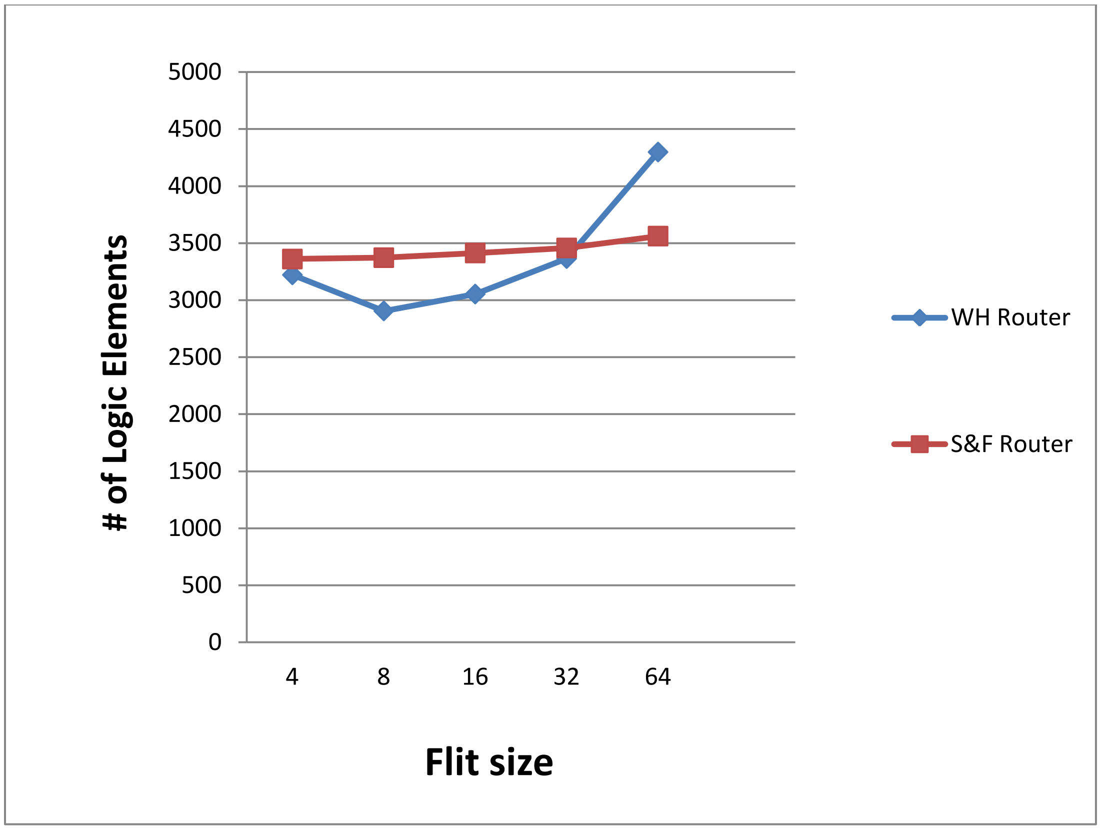


Figure 5.5: Area consumption of standalone routers with respect to flit size

Figure 5.5 shows a plot of area consumption verses flit size for SAF router the increase in area is very small as the flit size increases. The reason for small increase in area of SAF router is because the buffer size remains the same and there is only a small increase in logic usage. For WH router the area decreases slightly as the flit size goes from 4 to 8 and then increases significantly for flit size greater than 32 .There is a dip in area consumption of WH Router when flit size is 8, this may be possible because Quartus II CAD software is able to find better placement and routing in case of flit size 8.

Figure 5.6 show that area consumption increases for both routers as the number of local ports increases. The increases in area of WH router is steeper because the complexity of the full cross bar switch increases significantly as more ports are added. In case of SAF routers only the number of buffers for input and output port increase and the partial cross bar switch does not require much increase in area.

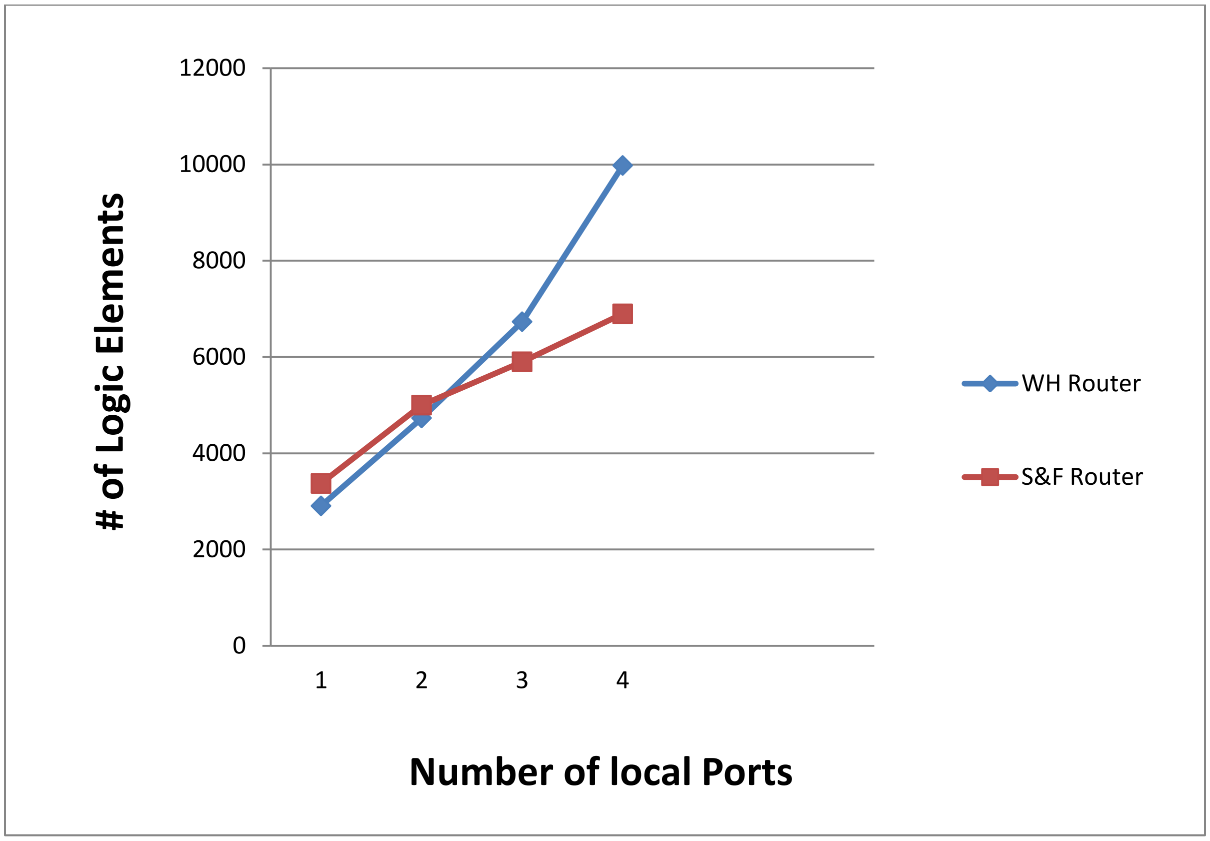


Figure 5.6: Area consumption of standalone routers with respect to number of local ports

Figure 5.7 shows the area consumption of 4X4 mesh NoC versus flit size, as can be seen the NoC based on WH router consistently requires more than 40% increase in area compared to SAF router. When we are implementing the whole NoC, area consumption of WH router increases significantly due to the number of FIFO's used in the adapter (As shown in Figure 3.10).

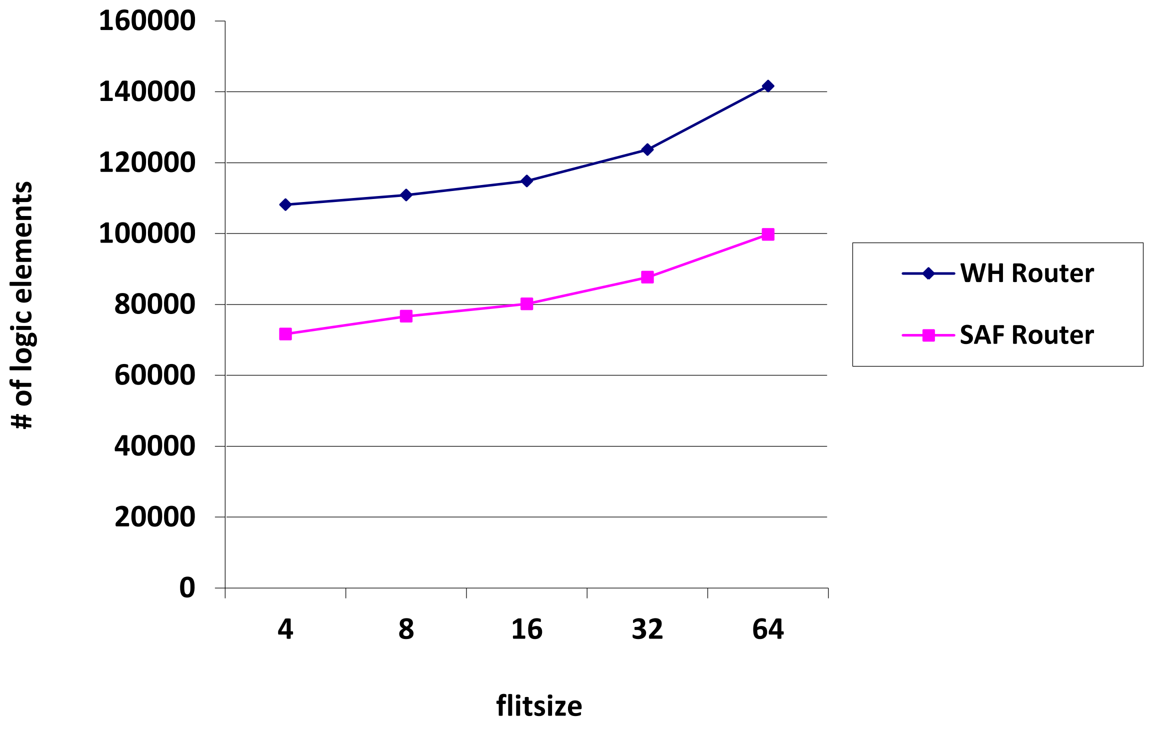


Figure 5.7: Area consumption of the whole NoC with respect to flit size (# of local ports is 1)

As can be seen in Figure 5.8 the area consumption of WH router is consistently higher as the number of local ports increases from 1 to 4.

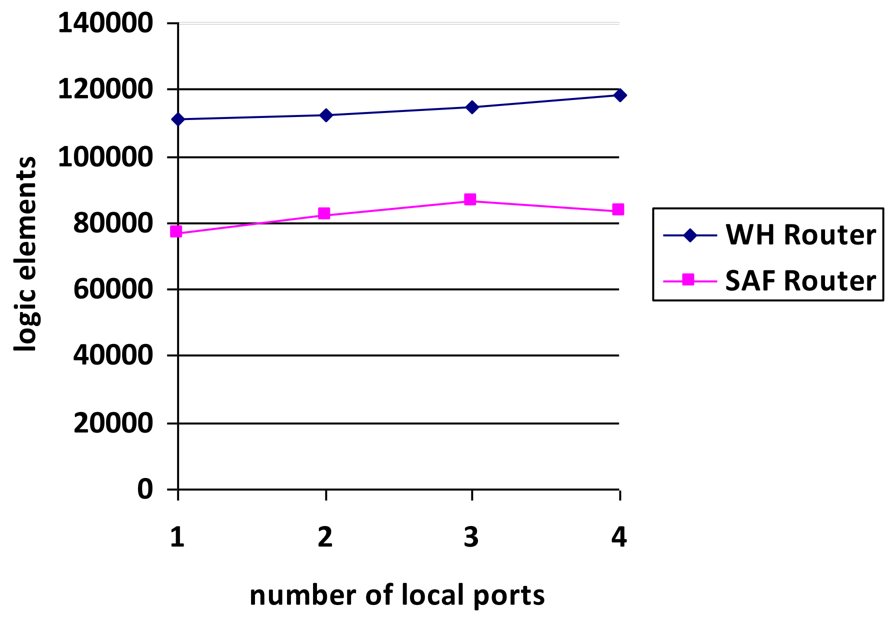


Figure 5.8: Area of consumption for whole NoC with respect to number of local ports (flit size=8)

So from the above discussion we can conclude that if area consumption is a critical factor then SAF router is a better choice compared to WH router.

5.2.3 Power Consumption Results

Figure 5.9 shows the power consumption versus flit size for the two standalone routers. The power consumption increases slowly as the flit size increases and SAF router uses consistently less power compared to WH router. The difference in power consumption between two routers is small but it adds up in large NoC's where we need many routers. Therefore, for bigger systems the difference in power consumption will be significant. If we analyze the internal structure of routers we can see that it has more components communicating frequently compared to the SAF router. This results in more active links at any given time and hence increases power consumption. From these results we conclude that SAF has lower power consumption compared to WH router.

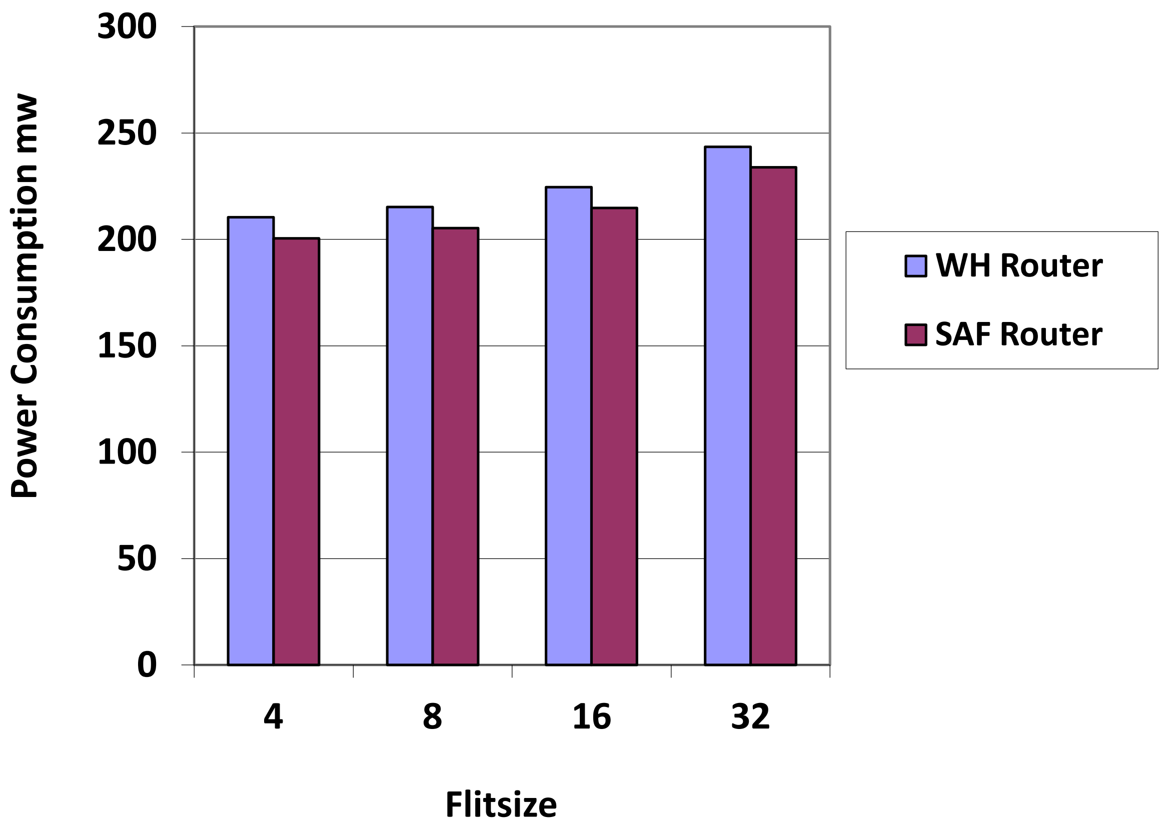


Figure 5.9: Power Consumption Versus flit size for standalone routers

5.2.4 Clock Frequency Results

Figure 5.10 shows the maximum clock frequency versus flit size of standalone routers. The maximum clock frequency remains the same as the flit size increases but SAF router is able to run at a much higher frequency (more than 3 times higher). This is because in case of WH routers a packet is routed after the path is determined so it holds the previous packet until the next path is decided. Where as in SAF router the whole packet is stored in buffer and the direction of next hope is determined quickly. Even though SAF router has higher clock frequency the speed performance of the router depends on its throughput not just the clock frequency.

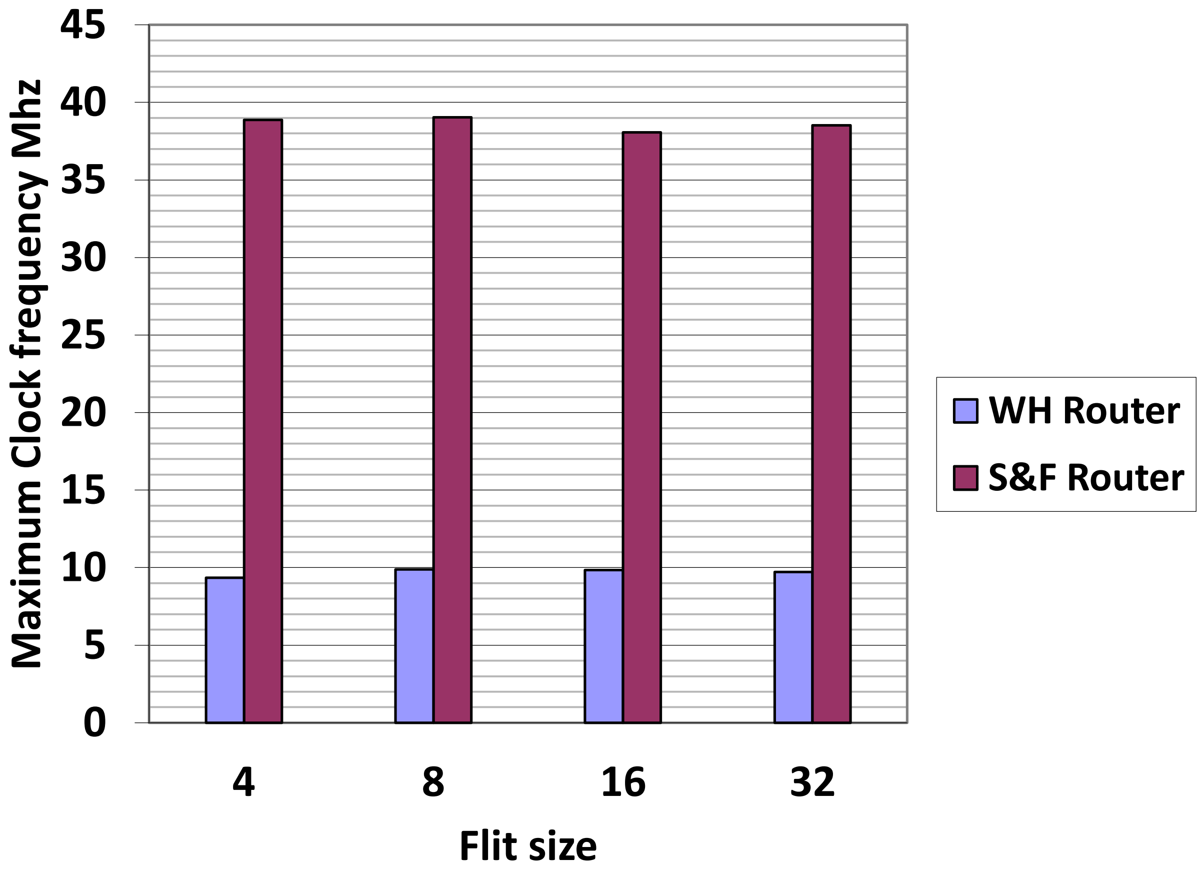


Figure 5.10: Maximum Clock frequency versus flit size for standalone routers

5.2.5 Throughput Results

Throughput is one of the most important parameters that can be used to compare two routers. Throughput measures the data handling capability of the NoC. For NoC based systems the message throughput, TP, can be defined as follows:

$$TP = \frac{(\text{Number of messages} * \text{Message Length})}{\text{Number of IPs} * \text{Total time taken}}$$

This definition of throughput is taken from [20]. Number of messages completed refers to the number of whole messages that successfully arrive at their destination IPs, Message length is measured in flits, Number of IP blocks is the number of functional IP blocks involved in the communication, and Total time is the time (in clock cycles) that elapses between the occurrence of the first message generation and the last message reception

Figure 5.11 shows a plot of throughput versus flit size for the whole of NoC. We can see that there is decrease in throughput as the flit size increases. This is because as flit size increases the message length decreases and total time increases only slightly and other factor are constant so the throughput decreases as the flit size increases. So from throughput point of view either of the routers is preferable except at flit size of 8.

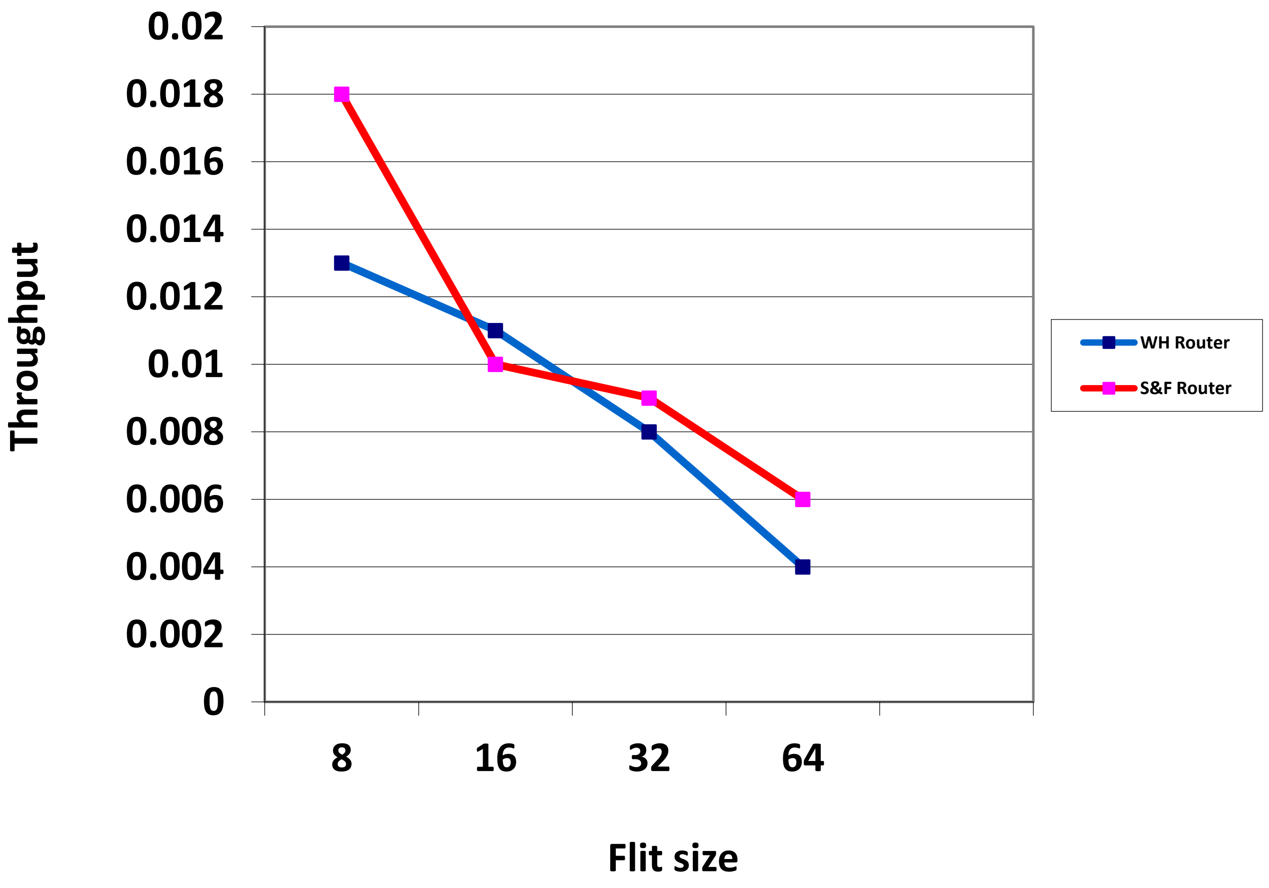


Figure 5.11: Throughput versus flit size for the whole NoC

5.3 Conclusion

This chapter presented an experimental comparison of SAF and WH router. These routers were compared by mapping a real world benchmark to 4X4 mesh NoC. Experimental results show that SAF router is superior to WH router in terms of area and power consumption.

Chapter 6: Conclusions and Future Work

As long as Moore's Law is valid the number of transistors that can be placed on a single chip will continue to increase exponentially leading to larger and increasingly complicated system-on-chip. The traditional bus-based or point to point communication infrastructure becomes resource intensive and design restrictive. NoC based systems provide an efficient and scalable communication infrastructure for future SoC and FPSoC's.

NoC designs have been evaluated using synthetic traffic generators but they do not provide us with accurate results [16]. There has not been much research done addressing real world testing and comparison of the NoC designs. Also much less work has been done to implement real benchmarks on NoCs using commercially available SoC CAD software, such as Altera's SOPC builder, Nios IDE and Modelsim simulator. Therefore, in this thesis we have compared and evaluated two different NoCs based on two different router designs. The NoCs uses common topologies (mesh) in order to compare two different router designs. The motivation of using a common topology was that we can compare the routers performance using different parameters such as area, power consumption, maximum clock frequency and Throughput. Given the time constraints we could not implement NoCs using other topologies. The mesh topology was chosen because it is easier to implement mesh based NoC [19].

Considerable work was done on creating adapters which could work with Avalon bus on the IP core side and wishbone protocol on the router side.

6.1 Research Contributions

This thesis presented experimental evaluation and comparison of two different routers WH and SAF. A real world benchmark design was used to compare two routers it was shown that SAF router gave superior area, power consumption and maximum clock frequency results.

Testing of the NoC Design with a real world benchmark exposed the flaws in SAF router. These flaws were due to lack of consideration of different packet sizes by the router. This resulted in many design changes and consequently higher area consumption as discussed in section 5.2.1. Also there were changes done to SAF Router prior to testing phase to make it work with standard socket mentioned in chapter 2. These changes made it feasible to use the router as a separate component which helped in decreasing experimental evaluation setup time and gives more flexibility to the NoC designer.

6.2 Future Work

The SAF and WH router designs used in this thesis can be utilized in future research. They can be used to implement evaluate and compare different NoC topologies using a number of benchmark designs. Work can be done on creating a library of parameterizable NoC components to automate the process of NoC implementation especially on FPSoC platforms.

Appendix A

Copyright Permissions

REQUEST FOR PERMISSION TO USE COPYRIGHTED MATERIAL

[February 22nd 2013]

[Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's]

Dear Matt Murawski,

I am completing a Master's Thesis at the University of Windsor entitled "**Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's**"

I would like your permission to include in my thesis the following material:

(i) From Chapter 2 Section 2.3 (Interface and Signals), Chapter 3 section 3.3.1(Adapter Overview), Chapter 4 Section 4.3 (NIOS II Programing) and section 4.4 (Modelsim Simulation), Figure 3.5, Figure 3.6, figure 4.6 and figure 4.9 From Thesis by Matt Murawski "**NoC Prototyping on FPGAs: Component Design, Architecture Implementation and Comparison**". May 18 2012

I have used this information in my experiments and also this information was best described in the above mentioned thesis written by you. I have also used this information to compare it with the other router developed by mike Brugge.

My thesis will be deposited to the University of Windsor's online theses and dissertations repository (http://winspace.uwindsor.ca) and will be available in full-text on the internet for reference, study and / or copy.

I will also be granting Library and Archives Canada and ProQuest/UMI a non-exclusive license to reproduce, loan, distribute, or sell single copies of my thesis by any means and in any form or format. These rights will in no way restrict republication of the material in any other form by you or by others authorized by you.

Please confirm by email that these arrangements meet with your approval.

Thank you very much for your attention to this matter.

Sincerely,

[Krunal Jetly]

REQUEST FOR PERMISSION TO USE COPYRIGHTED MATERIAL

[February 22nd 2013]

[Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's]

Dear Mike Brugge:

I am completing a Master's Thesis at the University of Windsor entitled "**Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's**"

I would like your permission to include in my thesis the following material:

(i) From Chapter 3 section 3.1.1.3 (Routing Algorithm), Figure 3.1, Figure 3.2, Figure 3.4, Figure 3.7, Figure 3.8 from Thesis by Mike Brugge "**Design and Evaluation of a Parameterizable NoC Router for FPGAs**". Sep 21 2009

I have used this information as I have been comparing the functionality of the router defined in above mentioned thesis with the other router defined in thesis by matt murawski.

My thesis will be deposited to the University of Windsor's online theses and dissertations repository (http://winspace.uwindsor.ca) and will be available in full-text on the internet for reference, study and / or copy.

I will also be granting Library and Archives Canada and ProQuest/UMI a non-exclusive license to reproduce, loan, distribute, or sell single copies of my thesis by any means and in any form or format. These rights will in no way restrict republication of the material in any other form by you or by others authorized by you.

Please confirm by email that these arrangements meet with your approval.

Thank you very much for your attention to this matter.

Sincerely,

[Krunal Jetly]

References

[1] C. Hilton and B. Nelson, "PNoC **“a flexible circuit-switched NoC for FPGA-based systems**," IEEE, Computers and Digital Techniques, May 2005, Page(s): 181-188

[2] Saleh, Resve A, Wilton Steven J E, Mirabbasi Shahriar, Hu Alan J. “**System-on-Chip: Reuse and Integration,**” IEEE, 2006 Proceeding, Page(s): 1050-1069.

[3] F. Moraes, A. Mello, L. Möller, L. Ost, N. Calazans, "**HERMES: an infrastructure for low area overhead packet-switching networks on chip**," ACM, Integration -the VLSI Journal, 2004, Page(s) 69-93.

[4] **Altera Corporation.** Altera Avalon Interface Specifications. Literature: SOPC Builder. [Online] April 2009. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf.

[5] http://www.element-14.com/community/docs/DOC-12235

[6] **Mentor Graphics Corporation.** ModelSim SE User's Manual. ModelSim SE | Verilog, VHDL, SystemVerilog Design & Simulation | ModelSim - Advanced Simulation. and Debugging:. [Online] 2010. [Cited: May 5, 2010.] http://portal.model.com/modelsim/resources/references/modelsim_se_user.pdf.

[7] **Quartus II Handbook Version 9.1** - Volume 4 SOPC Builder. Quartus II Development Software Literature. [Online] November 2009. [Cited: May 5, 2010.] http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf.

[8] M. P. Vestias, H. C. Neto, "**Area and performance optimization of a generic network-on-chip architecture**," ACM, 19th annual symposium on Integrated circuits and systems design, 2006, Page(s) 68-73.

[9] Janarthanan, V. Swaminathan, K. A. Tomko, "**MoCReS: an Area-Efficient Multi-Clock On-Chip Network for Reconfigurable Systems**," IEEE, Symposium on Computer Society, March 2007, Page(s): 455-456.

[10] **"Porting from Wishbone Bus to Avalon Bus in SoC Design"**. Xing, Xu, et al**.** Xi'an : s.n., 2007. Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference on. Vol. 1, pp. 862-865.

[11] **"An FPGA Based Open Source Network-on-Chip Architecture"**. Ehliar, A. and Liu, Dake**.** Amsterdam : s.n., 2007. International Conference on Field Programmable Logic and Applications, 2007. pp. 800-803.

[12] **"Design and Implementation of a Plesiochronous Multi-Core 4x4 Network-on-Chip FPGA Platform with MPI HAL Support"**. Minhass, Wajid Hassan, Öberg, Johnny and Sander, Ingo**.** Stockholm, Sweden : Proceedings of the 6th FPGAworld Conference, 2009. 978-1-60558-879-7.

[13] **"HERMES: An infrastructure for low area overhead packet-switching networks on chip"**. Moraes, Fernando, et al**.** 1, Amsterdam, The Netherlands : Elsevier Science Publishers B. V., 2004, Vol. 38. 0167-9260.

[14] **Silicore corporation; opencores.org.** WISHBONE, Rev.B3 Specs. *Wishbone::OpenCores.* [Online] July 9, 2002. [Cited: 05 03, 2010.] http://opencores.org/downloads/wbspec_b3.pdf.

[15] **Altera Corporation.** Nios II Processor Reference Handbook. Literature: Nios II Processor. [Online] November 2009. [Cited: April 20, 2010.] http://www.altera.com/literature/lit-nio2.jsp.

[16] R. Marculescu et al, "**Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives**, " IEEE Trans. on CAD of ICs and Systems, vol. 28, no. 1, January 2009.

[17] Nios II Software Developer's Handbook. Literature: Nios II Processor. [Online] November 2009. [Cited: May 5, 2010.] http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf.

[18] T. Le, “**Implementation and Evaluation of an NoC Architecture for FPGAs**,” M.S. Thesis, University of Windsor, 2009.

[19] Mirza-Aghatabar, M. “**An Empirical Investigation of Mesh and Torus NoC topologies Under Different Routing Algorithms and Traffic Models**” Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on.

[20] **“Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures”**. **Pande, Partha Pratim, et al.** 8, s.l. : Computers, IEEE Transactions on , 2005, Vol. 54. 0018-9340.

[21] M. Brugge, “**Design and Evaluation of a Parameterizable NoC Router for FPGAs”**, M.A.Sc. Thesis, Department of Electrical and Computer Engineering, University of Windsor, Canada, 2009.

[22] Matt. Murawski, “**NoC Prototyping on FPGAs: Component Design, Architecture Implementation and Comparison”**, M.A.Sc. Thesis, Department of Electrical and Computer Engineering, University of Windsor, Canada, 2012.

VITA AUCTORIS

Krunal Jetly was born in Mumbai, Maharashtra, India in year 1986. He received his bachelor's Degree in Electronics and Communication in 2008 from Sardar Patel University in V.v.Nagar, Gujarat, India. He received his M.Eng degree in electrical engineering in 2009 from the University of Windsor in Windsor, Ontario, Canada. He is currently a candidate in the electrical and computer engineering M.A.Sc program at the University of Windsor. His research interests include field programmable-related technologies, hardware and software development for embedded system, and digital computing.