2009

Implementation and Evaluation of an NoC Architecture for FPGAs

Thuan Le
University of Windsor

Implementation and Evaluation of an NoC Architecture for FPGAs

By

Thuan Le

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
2009

Your file Votre référence
ISBN 978-0-494-82086-5
Our file Notre référence
ISBN 978-0-494-82086-5

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 such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

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

The Networks-on-Chip (NoC) approach for designing Systems-on-Chip (SoC) is currently emerging as an advanced concept for overcoming the scalability and efficiency problems of traditional bus-based systems. A great deal of theoretical research has been done in this area that provides good insight and shows promising results. There is a great need for research in hardware implementation of NoC-based systems to determine the feasibility of implementing various topologies and protocols, and also to accurately determine what design tradeoffs are involved in NoC implementation. This thesis addresses the challenges of implementing an NoC-based system on FPGAs for running real benchmark applications. The NoC used a mesh topology and circuit-switched communication protocol. An experimental framework was developed that allowed implementation of NoC-based system from a high level specification, using the Celoxica Handel-C hardware description language. Two test applications: charged couple device (CCD) and JPEG were developed in Handel-C to be used as our benchmark applications. Both benchmarks are computational expensive and require large quantities of data transfer that will test the NoC system. Implementation results show that the NoC-based system gives superior area utilization and speed performance compared to the bus-based system, running the same benchmarks.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Mohammed A. S. Khalid, where my research would not be possible without his advice and wisdom that guided me over the course of this research. Dr. Khalid, who first offered an undergraduate course in reconfigurable computing for embedded system had introduced me to this exciting field. For this very reason, it had motivated me to continue my academic career in this field. My appreciation also goes out to my thesis committee members, Dr. K. Tepe and Dr. A. Ngom, for their time to sit on my committee and for reviewing my thesis.

I want to thanks my parents, for their constant support, encouragement, and to remind me every day to make time to eat. Thanks to my brothers (Steven and David) and sisters (MyLai and Dianna) for all those time I pester around when my research feels overwhelming.

Finally, I need to acknowledge my friends and fellow graduate students at the University of Windsor. Dat, Gordon, Jason, Omesh, Geraldo, and David, thank you for your friendship and for amusing me with your various stories. Ngan, I always enjoyed our conversation and how we continue to motivate each other as we pursuit our own graduate study. Thanks to Omar, Marwan, Junsong and Aws for your company and making those long days at the office more enjoyable. Lastly, thanks to the rest of my colleagues: to Hongmei, Matt, Mike, Liton, Yasser, Lin Lin, Carl, Neil, Anthony, Ashkan and everyone else who made this great milestone in my life more enjoyable.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
ASIC	Application Specific Integrated Circuit
CAD	Computer Aided Design
CB	Crossbar
CCD	Charged Couple Device
CCDBM	Charged Couple Device Benchmark
CCN	Central Coordination Node
CLB	Configurable Logic Block
CPU	Central Processing Unit
CS	Circuit Switched
DOPB	Data On-Chip Peripheral Bus
DK	Design Kit
DLMB	Data local Memory Bus
DRM	Digital Radio Mondiale
DSP	Digital Signal Processing
FDCT	Forward Discrete Cosine Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GUI	Graphic User Interface
HC	Handel-C
I/O	Input/Output
IC	Integrated Circuit
ILMB	Instruction Local Memory Bus
IOB	Input/Output Block
IP	Intellectual Property
JHDL	Java-based Hardware Description Language
JPEG	Joint Photographic Experts Group
LE	Logic Element
LUT	Lookup Table
MB	MicroBlaze
MIN	Multistage Interconnection Network
MSI	Medium Scale Integration

MUX	Multiplexer
NoC	Network-on-Chip
OPB	On-chip Peripheral Bus
PC	Personal Computer
PCC	Packet Connected Circuit
PS	Packet Switched
PLD	Programmable Logic Device
QoS	Quality of Service
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
Rx	Receiver
SoC	System-on-Chip
SOPC	System on a Programmable Chip
SSI	Small Scale Integration
Tx	Transmitter
UART	Universal asynchronous Receiver/Transmitters
VC	Virtual Circuit
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Chapter 1

Introduction

The complexity of a system on silicon is comparable to other macro systems such as space shuttle or skyscrapers, when measured in terms of the number of basic elements intricately connected together, but at a micro level. As semiconductor technology evolves, electronics industry continuously pushes the envelope for greater functional and performance capabilities in new electronic systems. Thus, there is a continuing need for new design methodologies and design space exploration.

With the continuation of technology paradigm shifts of integrated circuit (IC) technology, complete embedded systems can be built onto a single chip. This paradigm shift is known as System-on-Chip (SoC) paradigm. Since the introduction of SoC concept, designers relied on a custom-designed ad-hoc mixture of buses and point-to-point links as communication mechanisms. Design reuse of standard interface modules is considered as a way to allow designers to keep pace with the technological advancement of SoC. But as SoC complexity scales, it becomes more difficult, if not impossible, to fully capture their functionality and still satisfy performance requirements.

A significant amount of research goes into the development of new on-chip communication methods and there is a growing interest in Networks-on-Chip (NoC). NoC approach is currently seen as a unifying concept for addressing two main challenges. First, traditional bus-based communication mechanisms do not scale well with increasing system complexity, thus performance will continue to deteriorate with increased

complexity. Second, design and verification times for complex systems continue to grow; therefore, there is a great desire for efficiency in design and verification [1], [2].

The development of field programmable gate arrays (FPFAs) and other programmable logic devices (PLDs) has provided designers with a flexible and rapid prototyping medium for embedded systems designs. Any digital circuit can be implemented on an FPGA, subjected to the limitations imposed by the logic capacity of the FPGAs. Using FPGAs, different design tradeoffs can be rapidly explored, allowing better design decisions to be made and reducing the overall development time of a system [3].

Since NoC is a relatively new research area, there is immense potential and opportunities for research. A great deal of theoretical research has been done in this area that provides good insight and shows promising results. However, these results may not have much practical value since the evaluation methods used rely on NoC simulation under assumptions and approximation of conditions rarely found when running real practical applications [2], There is a great need for research in hardware implementation of NoC-based systems to determine the feasibility of implementing various topologies and protocols, and also to accurately determine what design tradeoffs are involved in NoC implementation.

This thesis is primarily concerned with the challenges of implementing an NoC-based system for on-chip communication running real applications. The emphasis is on the design of NoC targeted for implementation on FPGAs, since FPGAs serve as an excellent platform for rapid prototyping and design space exploration.

1.1 Thesis Objectives

The main goal of this research was to design and implement an NoC-based system for image processing applications, on an FPGA. This research has the following major objectives:

1. Investigate the feasibility of NoC implementation on FPGAs.
2. Investigate and acquire real world benchmark applications with features that would severely test the NoC implementation.

3. Compare the results obtained by running these benchmarks on an NoC-based system and a bus-based system.

For the first objective, an experimental framework was developed that allowed implementation of NoC-based system from a high level specification. using the Celoxica Handel-C hardware description language. To address the second objective, two test applications: charged couple device (CCD) and JPEG were developed in Handel-C to be used for our benchmark applications. Both benchmarks are computational expensive and required large quantities of data transfer that will test the NoC system. Finally, for the third objectives, results from the NoC-based system and the bus-based system, running the same benchmarks are compared and evaluated in terms of speed performance and area utilization on an FPGA.

1.2 Thesis Organization

The outline of this thesis is as follows. Chapter 2 introduces the reader to the greater context of this research by presenting relevant background information and previous work that has been done by researchers in this area. Chapter 3 discusses NoC hierarchy and various architectures. Chapter 4 discusses the design process of basic NoC components and the functionality of each component block. In Chapter 5, implementation methodology of two benchmark applications on NoC-based system and bus-based system are discussed in detail. Chapter 6 presents the results obtained by implementing the NoC-based system and bus-based system on an FPGA. Finally, Chapter 7 concludes this thesis and discusses possible future work in this area.

Chapter 2

Background and Previous Work

In this chapter, the background and previous work that is relevant to this research is presented. This chapter begins with an overview of Network-on-Chip (NoC) for readers to build an understanding of basic concepts in this field. That is followed by sections describing FPGA technology and two soft-core microprocessors that are popular among the embedded systems community. Lastly, this chapter concludes with a presentation of previous work that is closely related to this research.

2.1 Overview of NoC

There are many research papers and books dealing with micro-networks, with many subtle differences in definitions, concepts, and theories. In this section, for the sake of clarity, we present a collection of concise definitions of relevant concepts and theory that holds true for most NoC systems.

2.1.1 NoC Building Blocks

Micro-networks such as NoC consist of many individual hardware components that have the task of carrying out two main functions; computation and communication. The communication infrastructure of the NoC allows the establishment of a communication channel set up by a router for transmitting information between computational components (commonly referred to as cores). An example of an NoC interconnection is shown in Figure 2.1, which consists of routers and links. A router is solely dedicated to

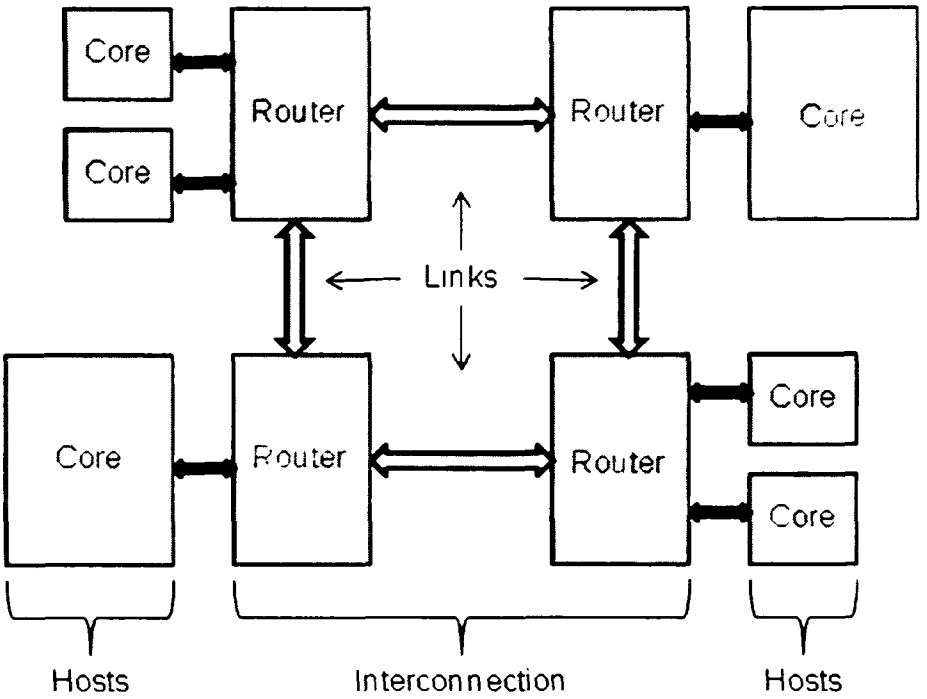


Figure 2.1: Hardware Components of an NoC [4]

carrying out the task of dispatching data inside the network, base on a routing algorithm. Links are physical connections between routers which can be latency insensitive and may contain buffering resources, if needed by a particular application.

2.1.2 NoC Architecture

The network architecture specifies the topology and the protocol which determine the routing and control flow scheme.

- Topology refers the logical layout of the interconnection network.
- Protocol refers to the switching mechanism in place for packaging data and a routing system for directing the flow of data through the network.

2.1.2.1 Network Topology

The network architectures are classified into four groups according to their topology [5]:

1. *Shared-medium networks*: The transmission medium is shared by all nodes and only one node at a time can have access to the transmission medium.

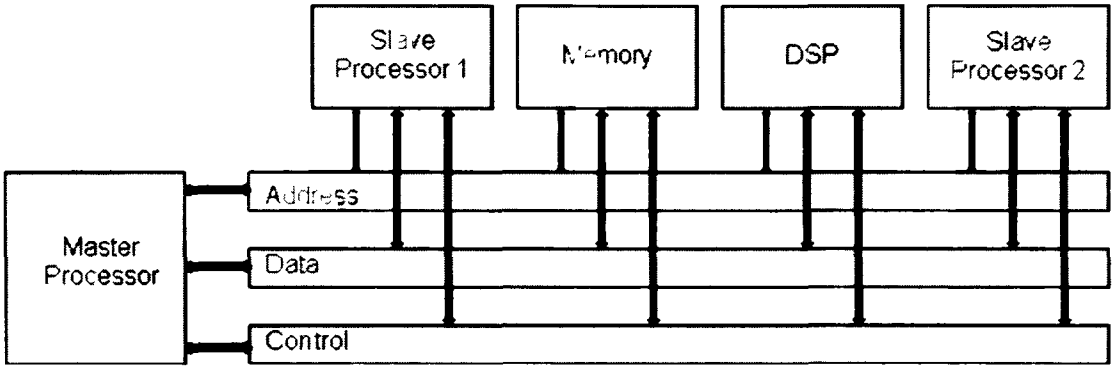


Figure 2.2: Shared-Medium Networks: a Traditional Bus-Based System

2. *Direct networks*: Each node has a router and point-to-point links to some or all of the other nodes.
3. *Indirect networks*: Each node is connected to a set of switches which can be programmed to implement given inter-node connections.
4. *Hybrid networks*: A mixture of the approaches listed above.

Shared-medium network is currently the most commonly used topology for Systems-on-Chips (SoCs) and has the simplest interconnection, in which all the communication devices share the transmission medium. A master processor will act as an arbitrator by prioritizing the slave processors to determine which node has access to the transmission medium. Therefore, when several slave processors are requesting the use of the bus, the processor with the highest priority gets primary access. Thus, it is important to have the arbitration process to quickly resolve the simultaneous requests. An example of shared-medium network is a bus-based systems, its simple topology carries low-overhead but does not scale well with high number of processors, as more buses are integrated to accommodate more processors. Thus, the shared-medium network is ideal for systems with small number of processors

Direct network architecture is designed to overcome the scalability problem of the shared-medium networks by allowing each router to directly connect to its neighboring routers. This network is a popular platform for building systems with many processors because increasing the number of processors also increases communication bandwidth. Performance trade-off for direct network architecture has been mainly between connectivity and wire cost. Higher connectivity will increase network performance but

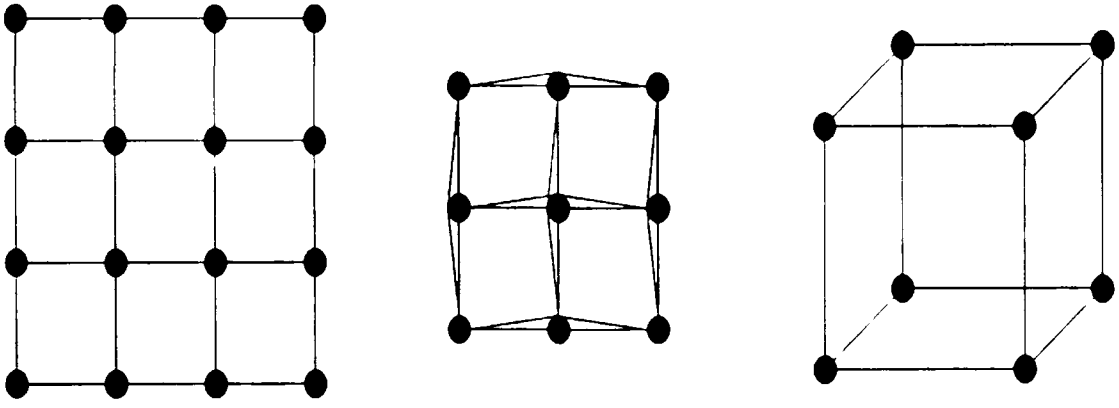


Figure 2.3: Direct networks: (a) Mesh, (b) Torus, and (c) Hypercube

also corresponds to higher area utilization and energy consumption [5]. Examples of popular direct network architectures are n-dimensional mesh, the torus and the hypercube depicted in Figure 2.3 These are the most practical implementations of orthogonal infrastructure, where the interconnection is arranged in n-dimensional orthogonal space. These prove to be the most simple in term of routing and hardware implementation.

The alternative to direct network is indirect network where the interconnection between nodes has to go through a set of switches or crossbars. Crossbar is a programmable component for establishing communication paths, connecting a node to all possible nodes is depicted in Figure 2.4. Crossbar can be very costly in terms of hardware requirements. The hardware complexity of connecting N nodes is $O(N^2)$.

Finally, the last architecture topology is the hybrid network which combines two or all three of the networks discussed above. Since each network class has their own merits, combining a mixture of these classes give the designer the flexibility of developing a network for addressing certain performance aspects. The main drivers for implementing a hybrid network are that they provide high-bandwidth and consume less energy. Therefore, hybrid network is seen as an ideal topology for applications that requires high performance but low energy consumption.

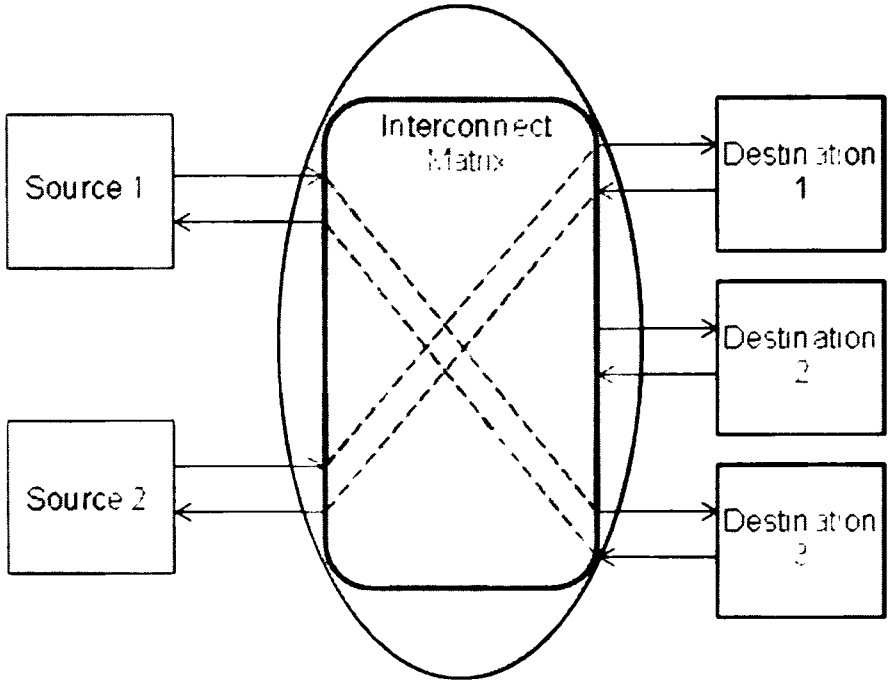


Figure 2.4: Indirect Networks (adapted from [1])

Each of the architecture topologies discussed, have positive and negative attributes that can be exploited for addressing performance issues. Studies conducted show a two dimensional mesh is considered to be the most suitable NoC for running most applications because a two dimensional mesh gives reasonable wire cost while still provide reasonably high bandwidth with predictable latency [6], [7].

2.1.2.2 Switching Protocol

Switching protocol determines how data flows through the NoC network. Deciding on a switching technique involves determining the granularity of data transfer. Strictly speaking, the switching techniques are classified in two basic modes of transporting data: packet switched (PS) and circuit switched (CS).

Since the early days of NoC, PS was used as a main switching technique where data are broken down into manageable packets, each containing data and routing information in the packet's header. These packets are injected into the network where they are independently routed before reaching its destination. Packets may not always follow the same paths, thus reaching its destination at different time [1], [8]. A major

drawback in employing PS is as more packets are being injected into the network, packet latency becomes unpredictable and also congestion control will become an issue In order to avoid this type of situation, each router has buffers to queue-up packets before routing to the next router [7] PS suffer from a rather significant area overhead because majority of the hardware is dedicated to buffers [10]-[13]

In contrast, CS establishes a dedicated connection (a virtual channel), a direct link for data transmission and does not require buffers to store packets for routing However, CS has a high initial latency incurred by building the virtual circuit before data can be transmitted CS is appropriate when data is sent very often, and then the initial latency becomes less relevant [1], [8], [9].

2.1.2.3 Performance

NoCs architecture are designed to meet certain performance demands, which include, but are not limited to, latency, throughput, energy consumption, scalability and area cost. While performance aspect such as latency and throughput are an important metrics for one application, it may not necessary be relevant for others that prioritize area and energy consumption.

Another performance metric that needs additional attention is Quality of Service (QoS) which has a direct relationship to the protocol used for routing. Typical parameter measured in relation to performance includes delay and bandwidth. Delay can be decomposed into several components depending on the different phases of the transfer process. Bandwidth corresponds to the measured bandwidth obtained by the application and only equals the capacity of the network under ideal assumptions which normally depends on traffic load and flow control protocols [4]

2.2 FPGA Technology

Field programmable gate arrays (FPGA) are semiconductor devices containing reconfigurable logic blocks and interconnections that can implement almost any digital circuit that fits within their logic capacity. FPGAs tend to run slower and consume more area and energy compared to application-specific integrated circuits (ASICs) But they

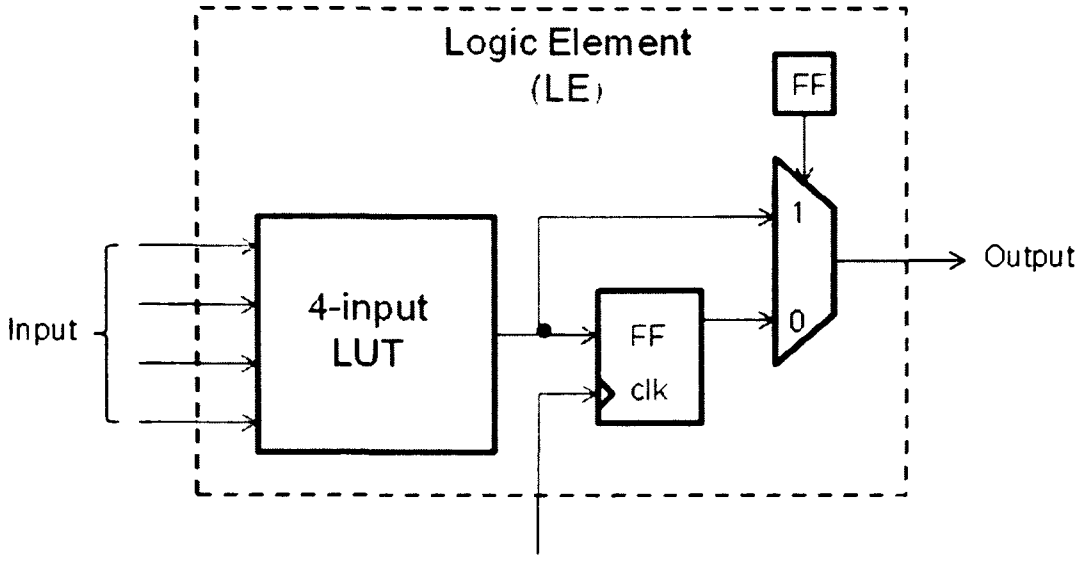


Figure 2.5: Schematic for a Generic FPGA Logic Element (LE) [14]

offer the circuit designer much greater flexibility. Their advantages over ASIC design included a shorter time to market, lower non-recurring engineering cost, and the ability to fix bugs by reprogramming. Thus, FPGAs are reconfigurable IC chips that serve as a good platform for design space exploration and prototyping.

The two largest manufactures of FPGAs are Altera Corporation [15] and Xilinx Incorporated [16]. The basic building block in an FPGA, referred to as a Logic Element (LE) is shown in Figure 2.5.

At the core of each LE is a block of programmable memory called a Lookup Table (LUT) shown in Figure 2.6. The LUT consists of an array of 1-bit memories connected to a multiplexed output pin. If the LUT has n inputs, then the memory array will have 2^n bits. This array can be programmed with the truth table of any possible n-input Boolean logic function, and the n multiplexer (MUX) select inputs decide which of the 2^n memory array bits appears at the LUT output [3].

A Logic Block consists of large number of LEs that are connected together using reconfigurable interconnections shown in Figure 2.7 as (L). FPGA routing architecture is made up of many programmable Switch Blocks (S) that are programmed to complete a circuit. The Logic Block connects to a set of horizontal and vertical wires to the Connection Block (C) where each output from the Logic Block are programmed to

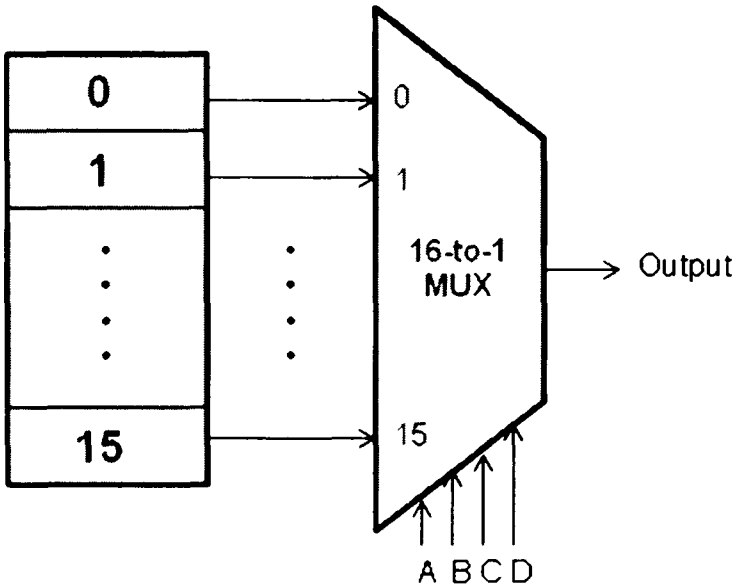


Figure 2.6: Schematic of a Lookup Table (LUT) [3]

connect to the wire segments of the routing architecture. At each junction lies a Switch Blocks which are programmable to connect to other wire segments to complete the circuit. Input/Output Blocks (I/O) which can be unidirectional or bidirectional are arranged around the perimeter of the FPGA chip, are used for receiving external signal and sending signal.

Xilinx Spartan3 xc3s1500lfg320 FPGA has been selected as the target device in this research; therefore a brief description of the Spartan3 architecture [17] is given. All of the devices in the Spartan-3 family contain four different types of logic resources: Configurable Logic Blocks (CLBs), block RAMs, Dedicated Multipliers, and I/O Blocks (IOBs). CLBs constitute the main logic resource for implementing sequential as well as combinatorial logic. CLB are blocks which consists four connected slices to implement user-defined logic functions. For our target FPGA device, there are 64 rows and 52 columns of CLB with a total of 3,328 slices. The target FPGA device has 32 block RAMs, each containing 16 kilobits of memory and 18,432 bits including the parity bits, totaling 589,824 memory bits. The block RAM has a dual port structure to provide single-port or simple dual-port memory operation. All Spartan-3 devices provide embedded multiplier and the target FPGA device has 32 dedicated multipliers that accept

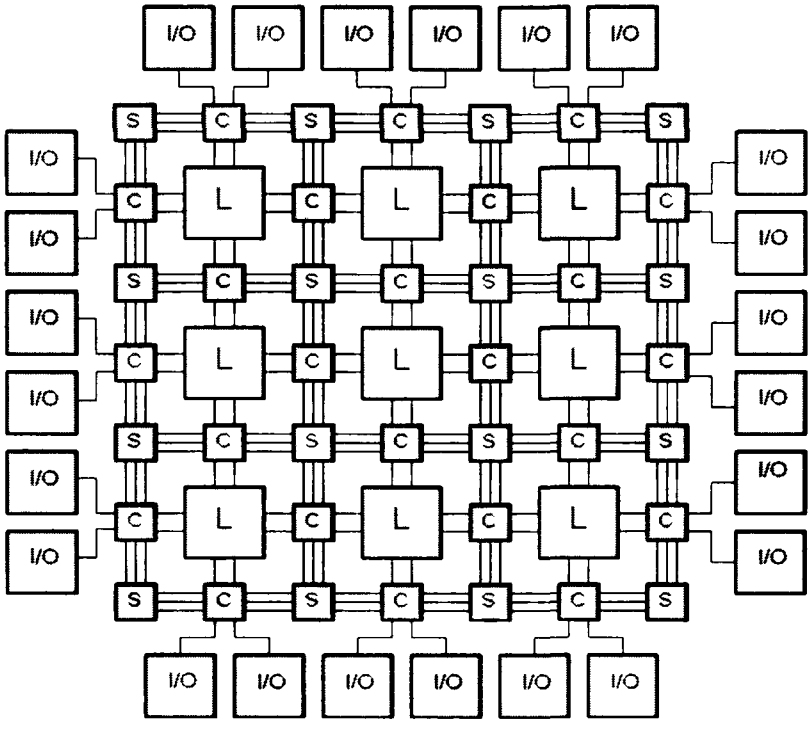


Figure 2.7: Generic FPGA Routing Architecture

two 18-bit words as inputs to produce a 36-bit product. Lastly, there are 487 IOBs that provide a programmable bidirectional interface between an I/O pin and the FPGA's internal logic [17].

2.3 Soft-core Processors

Historically, FPGAs were used to implement SSI and MSI level logic functions. Following Moore's law, FPGA logic capacity has increased substantially to the point that multi-million gate FPGAs are currently available. This enables one or more CPU's to be implemented on a single FPGA. Most FPGA vendors commonly provide their own soft-core processors targeting their own device as well as other intellectual property (IP) cores for commonly used functions such as digital signal processing, encryption, etc [3]. Altera [15] and Xilinx [16] provide soft-cores processors targeting their devices help facilitate the development of embedded systems on FPGAs.

2.3.1 Nios II Soft-core Processor

Nios II soft-core processor [18] is the flagship processor core of Altera Corporation [15] that is targeted for Altera's FPGAs. The Nios II soft-core processor is a general purpose reduced instruction set computer (RISC) processor that is optimized for embedded applications. This soft-core consists of three processor variants that can be selected based on a designer's specific needs: the Nios II/f fast core, which is designed for maximum performance, the Niox II/e economy core, which is the smallest processor core, and the NiosII/s standard core, which is a tradeoff between the fast core and the economy core. These cores each feature their own set of configurable options, and all of them provide support for up to 256 custom instructions and interfacing to peripheral devices using the automatically-generated Avalon bus [19]. Using Quartus II Cad tool suite with System on a Programmable Chip (SOPC) Builder from Altera Corporation, designers can develop their embedded systems designs to connect other soft-core processors and peripherals such as timers, memories, and universal asynchronous receiver/transmitters (UARTs) [20].

2.3.2 MicroBlaze II Soft-core Processor

MicroBlaze [21] is a 32-bit general purpose RISC soft-core processor optimized and targeted for Xilinx FPGAs. MicroBlaze soft-core processor is highly configurable, allowing embedded systems designer to select a specific set of features that includes 32-bit general purpose registers, a 32-bit instruction word with three operands, two addressing modes for data, instruction memories, and a 5-stage single issue pipeline. MicroBlaze also includes a large number of parameters, including an optional hardware barrel shifter, multiplier, divider, floating point unit (FPU), and others [21]. MicroBlaze system design relies on number of slave peripherals placed on the interface known as the On-chip Peripheral Bus (OPB) to interface with MicroBlaze with memories and other peripheral components [22]. In this research, we used MicroBlaze as our soft-core processor.

2.4 Related Work

Wiklund et al. [7] proposed a two-dimensional mesh, CS network called SoCBUS for real time embedded systems which is strictly an ASIC/SoC design. The SoCBUS architecture uses five port router that allocate four ports for connecting to adjacent routers and one port to connect to the local IP core. There are at least four phases for each router transaction where four phases occur in a successful routing attempt and additional phase for each failed attempt when a router is blocked. As this request finds its way through the network, the router is temporarily locked and cannot be used in other transactions. To decrease the chances of blocking, Wiklund et al. uses some of the packet switching routing technique in addition to the circuit switched routing technique called Packet Connected Circuit (PCC). PCC does not lock resources during routing transaction. Thus, no blocking will occur by introducing a buffer for holding request package during routing transaction. The test application is a typical real time embedded system that runs in a simulator; Telephone-to-VoIP Gateway. The authors do admits that SoCBus is not suitable for computing platforms where random traffic patterns are observed to exhibit a higher probability of blocking when running without scheduling. Therefore, Wiklund concluded that their CS NoC had a high latency for setting up a new circuit.

T. A. Batric et. al. [12], [23] argued that different types of networks will be require for addressing different application domains. Depending on topology and on the number of cores attached, router ports are parameterized at design stage. This approach allows for a very flexible network design and allows users to instantiate arbitrary network topologies. Each router contains a routing table that keeps the record of network resources and then the router arbitrary makes routing decision to fully utilize routing resources. This proposed routing algorithm does not scale very well where memory space grows exponentially with the additional of more inputs. The authors implemented a two-dimensional PS with five inputs and 5 outputs crossbar on a Xilinx Vertex2Pro FPGA synthesized in VHDL. The network using virtual cut-through switching as a routing algorithm has a low latency while maintaining a high throughput but it comes at a cost. The design suffers from a rather significant area overhead from buffers used in the router.

For example, 97% of total NoC area utilized is dedicated for buffers only. Therefore, this architecture is more suitable for networks with a known traffic pattern.

Hilton and Nelson [1] implemented a CS NoC on Xilinx Virtex FPGA and a Java-based HDL language (JHDL) was used in design description. The proposed network topology consists of a series of subnets; a router and a collection of network nodes. This topology allowed the placement of modules that communicate frequently in the same subnet for increasing the efficient of the overall communication system. Hilton et al. proposed a router structure where the number of ports used can be parameterized at build time and can also supports the dynamic removal and insertion of nodes in the system at run time. Establishing a connection path is done through the router where all requests are sent to the routing table and the success on establishing a connection is base on the availability of free port. The test application used a simple image binarization system for thresholding to quantize grayscale image into binary pixels of black and white values. The authors compare their results to the NoC system from T. A. Batric et. al [12] to point out some significant improvements. The simplicity of the circuit switched NoC architecture reduces hardware cost by over two times and it also increases the clock rate by almost three times. The authors also admit that "Depending on the FPGA used, router connectivity may be compromised when support for dynamic module replacement is desired." This clearly restricts the topology that this NoC is build-upon.

From Wiklund [7] and Batric [12], [23], the establishment of connections is independently done by router with the aid of a routing table for managing routing resources. Main drawback of such routing system is the great overhead cost dedicate for the use of buffers for each router. Wolkotte [9] proposed a regular two-dimensional mesh CS network that allocates the task of routing system to one core, called Central Coordination Node (CCN). The CCN perform scheduling of individual processes and communications during execution by configuring each router. The advantages of this scheduling system over Batric [12] and Wilund [7] is that since the connections path is predetermine by the CCN, blocking in the router do not occur. Therefore, there is no need for buffer in individual router. The ASIC design runs three wireless test applications: HiperLAN/2, UMTS and Digital Radio Mondiale (DRM). Comparing to a

PS equivalent. the area of the circuit switched router is three times less compared to the PS router due to the effect of the absence of buffers and extra controller in the router for packet switched.

2.5 Summary

In this chapter, the relevant background material and related previous work was presented. First, a short collection of concise definitions of relevant concepts and theories were presented. Next, the basic concepts of FPGA technology were discussed with a brief discussion of the features of the Xilinx Spartan-3 FPGA. Then, soft-core microprocessors targeting FPGA was discussed followed by examples of the two popular soft-core microprocessors: Nios II from Altera [15] and MicroBlaze from Xilinx [16]. Finally, the Chapter concluded with a discussion of some of the previous work that is closely related to this research. In Chapter 3, a detailed description of the NoC architecture hierarchy and abstraction layers is presented.

Chapter 3

NoC Architecture

In this chapter a detailed description of our proposed NoC architecture is presented. This chapter begins with a discussion of trade-offs involved in selecting different architectures. That is followed by a discussion of the role each abstraction layer plays in protocol implementation which provides a useful insight into switching techniques. This chapter concludes with a discussion of the many issues designers face when developing a NoC system.

3.1 Topology

In section 2.1 we discussed different topologies used for NoC implementation. When selecting a topology, designers must evaluate key performance trade-offs such as area utilization, speed performance and power consumption.

First, the proposed NoC system is developed targeting an FPGA device. Since the logic capacity of an FPGA is limited, the NoC system should be as small as possible to allow most of the hardware resources in FPGA to be dedicated for user defined logic for computational tasks, rather than communication.

Second, the performance of an NoC is generally measured by the ability of the system to process high volume of data. Higher performing NoC implies more connections, bigger channel width, and increased system capacity. But a high-performance NoC often translates into higher area cost and less reliable system.

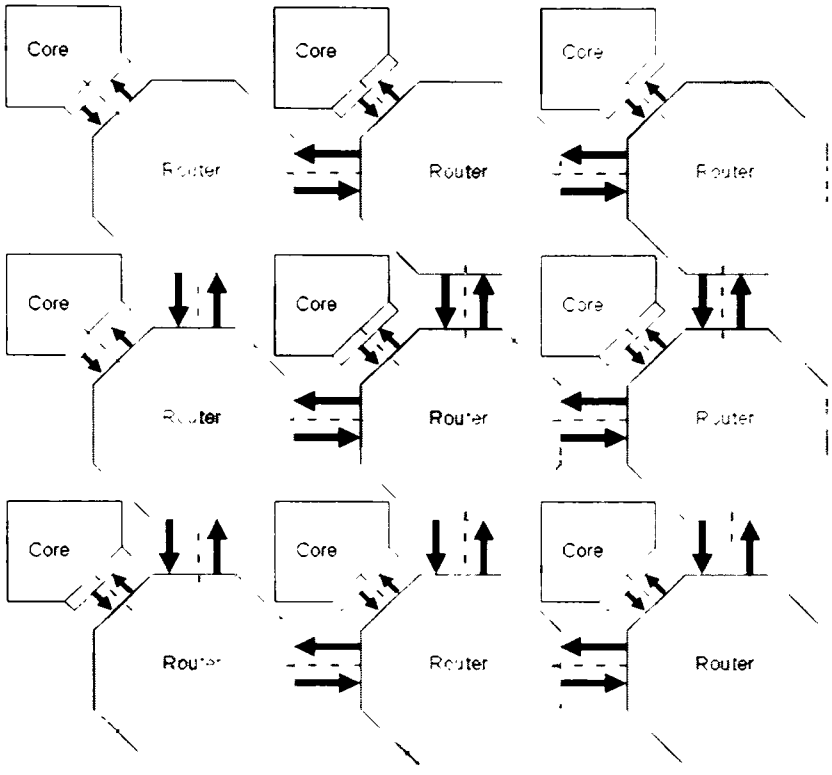


Figure 3.1: 2-Dimensional Mesh Topology

Therefore, a simple and small NoC architecture is desirable which satisfies performance requirements.

Finally, energy consumption depends on the number of active components in NoC independently routing data through the network and the power dissipation due to data in the NoC itself [8]. Topologies that consume less energy have shorter routes, and have components to stay inactive when not in use.

From extensive work [4], [7]–[9], a two-dimensional mesh depicted in Figure 3.1 is considered to be suitable for most NoC implementations because a two dimensional topology provide reasonable high bandwidth with predictable latency at reasonable wire cost [24]. For this reason, a two-dimensional mesh topology was chosen to simplify our design. Also, implementing a router based on partial crossbar system (discussed in Chapter 4) further decreases the energy consumption, when compared to using a full crossbar for implementing the router.

3.2 Protocol

Implementing a communication protocol is done by distributing the service to each abstraction layer. The higher the abstraction layers, the broader the scope in which the protocol operates over the network. This section will address the various tasks at each abstraction layer [4], [8].

1. *Physical layer*: task of the physical implementation of transmitting information over a physical link.
2. *Data link layer*: task of guaranteeing a reliable data transfers across inherently unreliable physical links.
3. *Network layer*: task of establishing network connection for the purpose of data transfer determined by the choice of switching and routing algorithms.
4. *Transport layer*: task of the decomposition of data for transmission at the source and assembly at the destination while addressing the flow control and congestion issues.

A circuit switched (CS) protocol is implemented and the following subsections explain the operations at each of the above abstraction layers.

3.2.1 Physical Layer

Physical layer has the sole responsibility of transmitting data through physical links. In our case, a unidirectional point-to-point link connects neighboring nodes. The port has three groups of signals depicted in Figure 3.2.

- *Data signal:* the width of this communication channel is parameterizable and transmits the data bits in parallel.
- *Valid signal and Request signal:* these physical wires are use in a handshake protocol to mark the beginning and the end of data transmission.

Each routing port consists of unidirectional output port and input port, each with three types of signals: *Data* channel, *Valid* signal and *Request* signal. For output port, the *Data* channel connects to the neighboring router input port and the two other signals

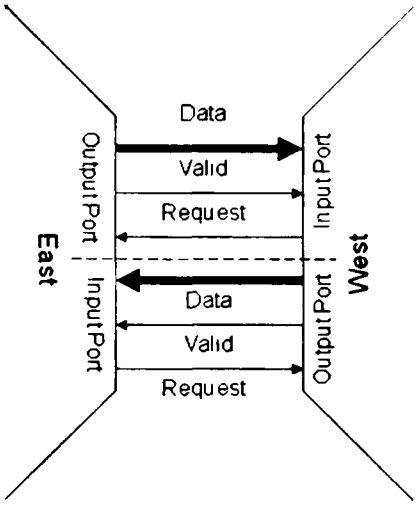


Figure 3.2: Port Interface

connects to two adjacent routers, one in each direction with the *Valid* signal following the direction of the *Data* channel and the *Request* signal the opposite.

3.2.2 Data Link Layer

The main task of data link layer is to guarantee reliable data transfers across the physical links and to provide error correction.

If errors were to ensue, there's a high probability of the error to occur during data transmission. Simplifying the data transmission protocol by developing a light weight handshake protocol can reduce the chance of error occurring. In the protocol, data transmission can only take place when the destination core has sent a request for receiving data. If data is transmitted when the destination core has not sent a request, the level *Valid* signal would indicates the data is invalid.

In NoC, especially NoC employing PS, there is a tendency to allow two or more transmitters to concurrently send data to the same core [8]. This can be minimized by deploying an indirect system that uses a crossbar system for routing since crossbars are programmed to establish a direct data path and the NoC should have a flow control measure in place to control traffic congestion.

3.2.3 Network Layer

The task of network layer is to establish the type of connection and to also determine the path to be taken to reach its final destination. The CS routing protocol can be described as deterministic [4] since the path is already determined by the virtual circuit. Unlike packet switched, storing data in buffers is not necessary since incoming data is forwarded immediately, thus insuring Quality-of-Service (discussed in 3.4).

3.2.4 Transport Layer

The transport layer tasks are the decomposition of data for transmitting at the source and assembly at the destination, while also addressing congestion and flow control. At this layer, there are two approaches that address the flow control: deterministic approach and statistical approaches.

PS protocol is referred to as the adaptive approach. The advantage of the adaptive approach is the increased efficiency in utilization of network resources during run time. In contrast, deterministic techniques may lead to under-utilization of network resources based on worst cases scenarios, whereby most of the resources may not be utilized during run time.

3.3 Data Transactions

Before the transmission of data, cores from each node must notify the network of its current state of its readiness for accepting data by deploying a light weight handshake protocol. The data transmission handshake protocol uses four phases and is summarized in Figure 3.3:

1. When a destination core is ready for accepting data, it will send out a *Request* signal, a request for receiving data.
2. The source core will transmit data, accompanying the data is the *Valid* signal to indicate the transmission of data is still in progress and valid.

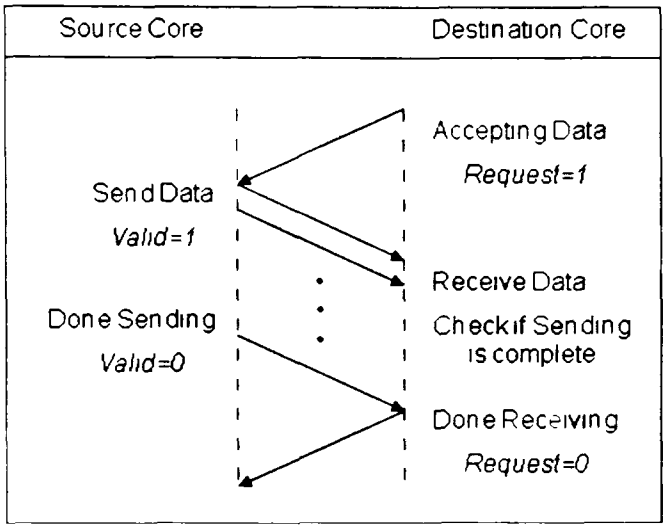


Figure 3.3: Data Transmission Handshake Protocol

3. When transmission is complete, the source core will send a *Valid* signal of 0, signaling the transmission of data is complete and wishes to start new transmission.
4. The destination core will send back an *Request* signal of 0 to signal that all data have been received and will resend the *Request* signal when cores is ready to process new data.

3.4 Quality of Service

Quality-of-Service (QoS) is a networking term that refers to the specification of network services that need to be provided in a specific application. Certain application such as DSP or video streaming will required a guarantee of high uninterrupted bandwidth because of the uniqueness of the application. It is difficult to actually predict the behavioral nature of the data in the network, thus making it nearly impossible to guarantee the required bandwidth without some margin of errors. To guarantee QoS, network should consider the following causes of failure to minimize the traffic disruption in the network [4], [8]:

1. *Deadlock:* data is prevented from reaching its destination because it is blocked at some intermediate resource.

2. *Livelock:* data is prevented from reaching its destination because it is in a cyclic path.
3. *Starvation:* data is prevented from reaching its destination because some resource does not grant access.

Deadlock is cause by packet being continuously blocked and it is the hardest problem to solve because packets that are blocked stay blocked while waiting for an event that cannot happen. Since deadlock is more associated with PS, employing CS will avoid any problems associating with deadlock.

Livelock occurs when the packets are being routed around their destination and are placed in a cyclic holding manner. Livelock can be avoided by allowing the packet to travel the shortest route.

Starvation occurs when the packet is discriminated against as low-priority packet data, thus never getting service. This can be avoided by allocating resources to process all packets equally such as employing a round robin or FIFO scheme. Also, starvation can be avoided by reserving some resources for processing low-priority packets.

Majority of these failures are associated with PS. In CS, the only cause for failure is livelock, which can be easily solve by placing cores with high interaction closer together and route path with the shortest distance. Therefore, circuit switched can better offer higher QoS.

3.5 Summary

A detailed description of NoC architecture hierarchy and design principles was presented. The selection of topology and protocol was made to satisfy some general goals. In this research, a 2-dimensional mesh topology was chosen because its simple topology is less area intensive while still providing reasonably high performance. We chose CS as our switching protocol because it provides a high degree of QoS compared to PS. In Chapter 4, the proposed NoC design is described in detail.

Chapter 4

The Proposed NoC Design

In this chapter, a detailed description of the proposed NoC components such as routers, configuration blocks, network adaptors, and links is presented. First, the router internal structure and lay-out is described. This is followed by a description of virtual circuit (VC) set up by the configuration block. Then we describe the integration of computational hardware blocks with the NoC system using the network adaptor. Lastly, we briefly discuss the issue of protecting signal integrity in NoC-based systems.

4.1 Router

The development of a router can presents some unique challenges where designers have to evaluates trade-offs to determine a configuration that is best suited for their needs. Our research goal is to develop a generic router that is highly configurable and area efficient, for NoC implementation. Although Sethurman et al. [25] developed a packet switched (PS) router called LiPaR, a light-weight scalable parallel router, area can be further reduced by employing a circuit switched (CS) scheme where buffering is not necessary. Our goal was to develop a highly modular and scalable router that is light weight with low area cost. The block diagram of a generic CS router is depicted in Figure 4.1

We implemented a router with 5 bidirectional ports: *North Port*, *East Port*, *South Port*, *West Port*, and *Core Port*. The first four ports are for connecting to neighboring routers and the *Core Port* is for connecting to the local port that could be connected to a computational hardware block. Each port consists of *IN* and *OUT* unidirectional port consisting of three physical links: *Input* link, *Valid* link, and *Request* link.

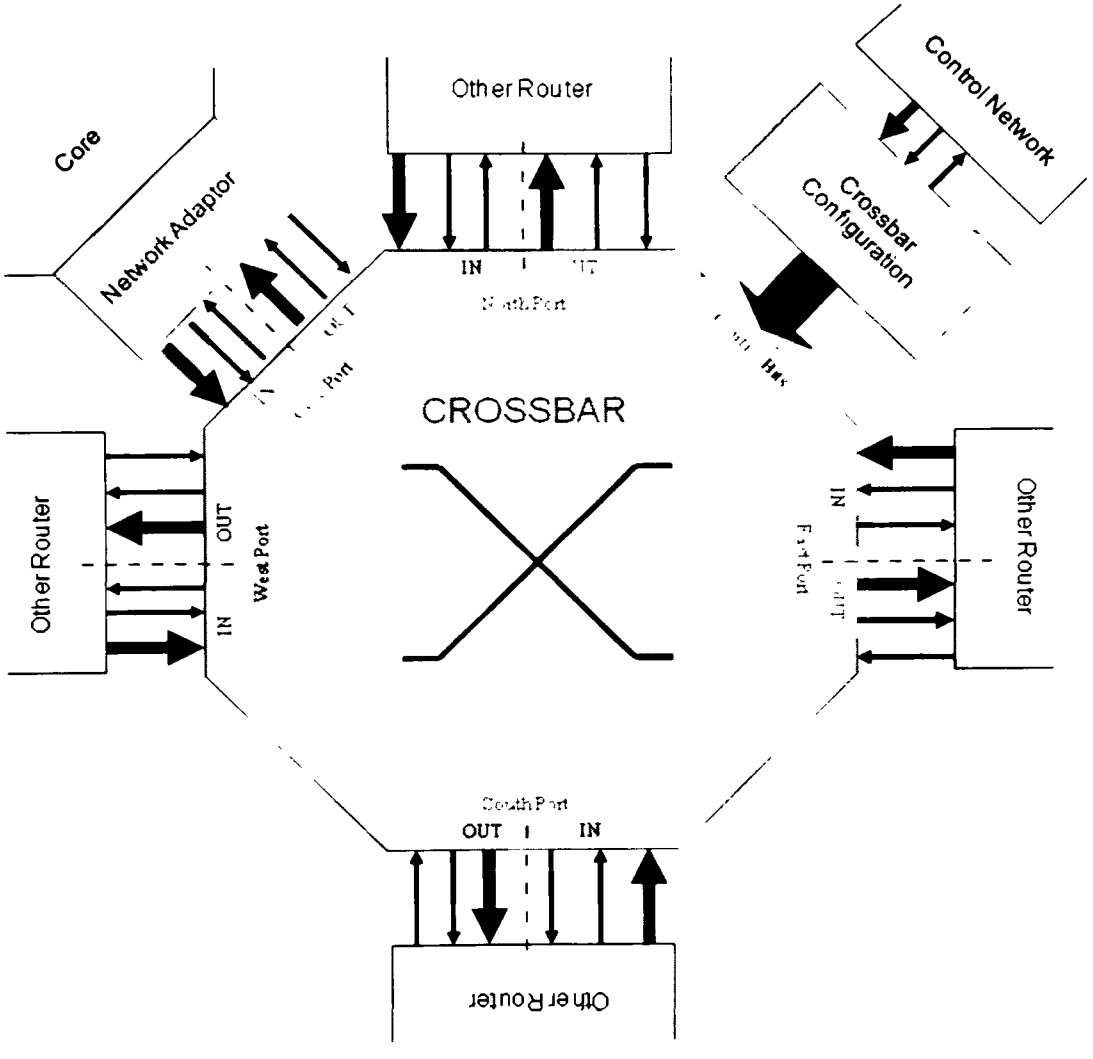


Figure 4.1: Block diagram of a Router [9]

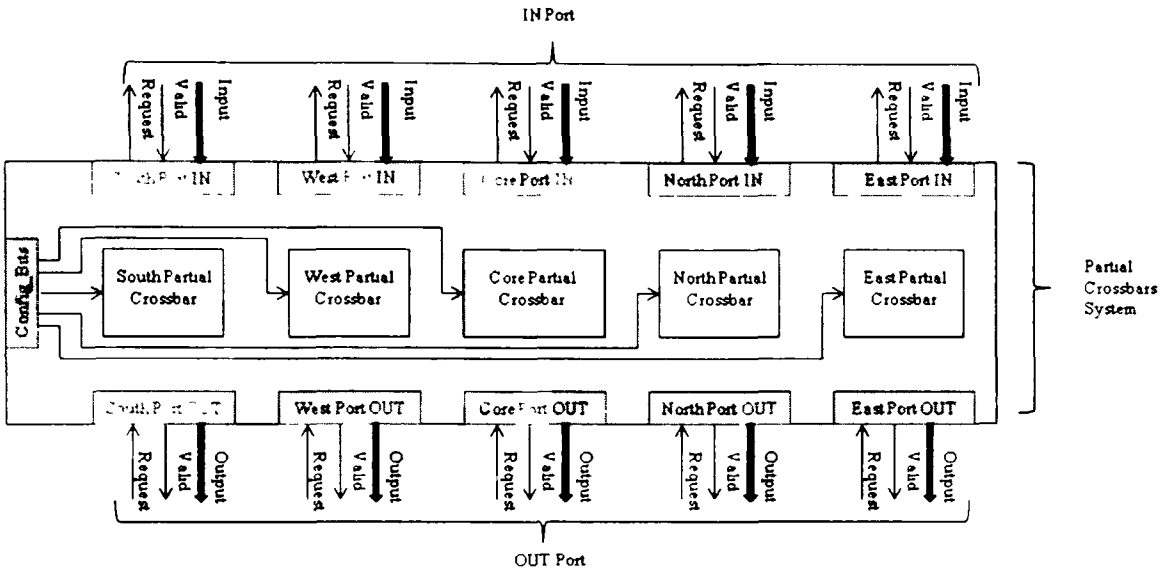


Figure 4.2: Cascading Intermediate Partial Crossbars

The functionality of the router can be viewed as the multiplexing and demultiplexing of data to direct the flow of incoming and outgoing signals. Crossbars provide a programmable interconnection for establishing paths between ports, where the multiple buses from all incoming ports have a corresponding bus to connect to all outgoing ports, constituting a full crossbar. Because a full crossbar is a very complex structure, it is possible to implement the functionality of a full crossbar by using a number of smaller crossbars. This is called a partial crossbar. Compared to full crossbar, partial crossbars are significantly smaller and consume much less energy [8].

4.1.1 Partial-Crossbar Design

Partial crossbar is an effective alternative to full crossbar for reducing structure complexity and promoting energy efficiency. To implement a crossbar using partial crossbars, each individual partial crossbar is cascaded into a *multistage interconnection network* (MIN) as shown in Figure 4.2, a partial crossbar for each output port.

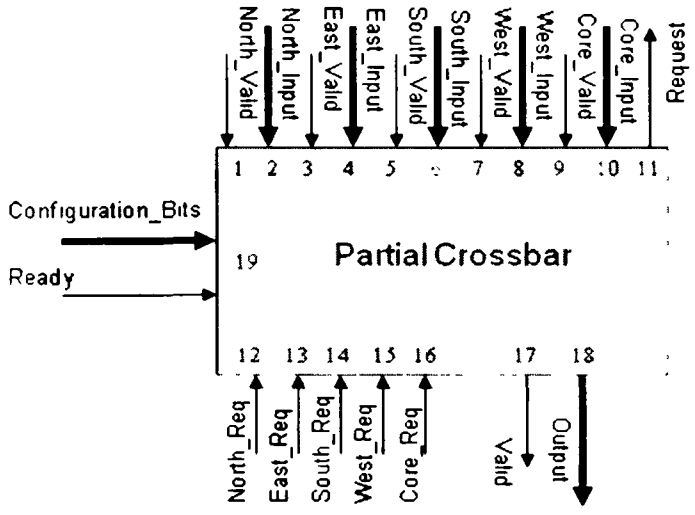


Figure 4.3: Partial Crossbar

Table 4.1: Connections Assignment for Partial Crossbar

Connection Number	Description
1	Input north *valid* signal
2	Input north *data* signal
3	Input east *valid* signal
4	Input east *data* signal
5	Input south *valid* signal
6	Input south *data* signal
7	Input west valid signal
8	Input west *data* signal
9	Input core *valid* signal
10	Input core *data* signal
11	Output partial crossbar *request* signal
12	Input north *request* signal
13	Input east *request* signal
14	Input south *request* signal
15	Input west *request* signal
16	Input core *request* signal
17	Output partial crossbar *valid* signal
18	Output partial crossbar *data*
19	Input crossbar configuration information

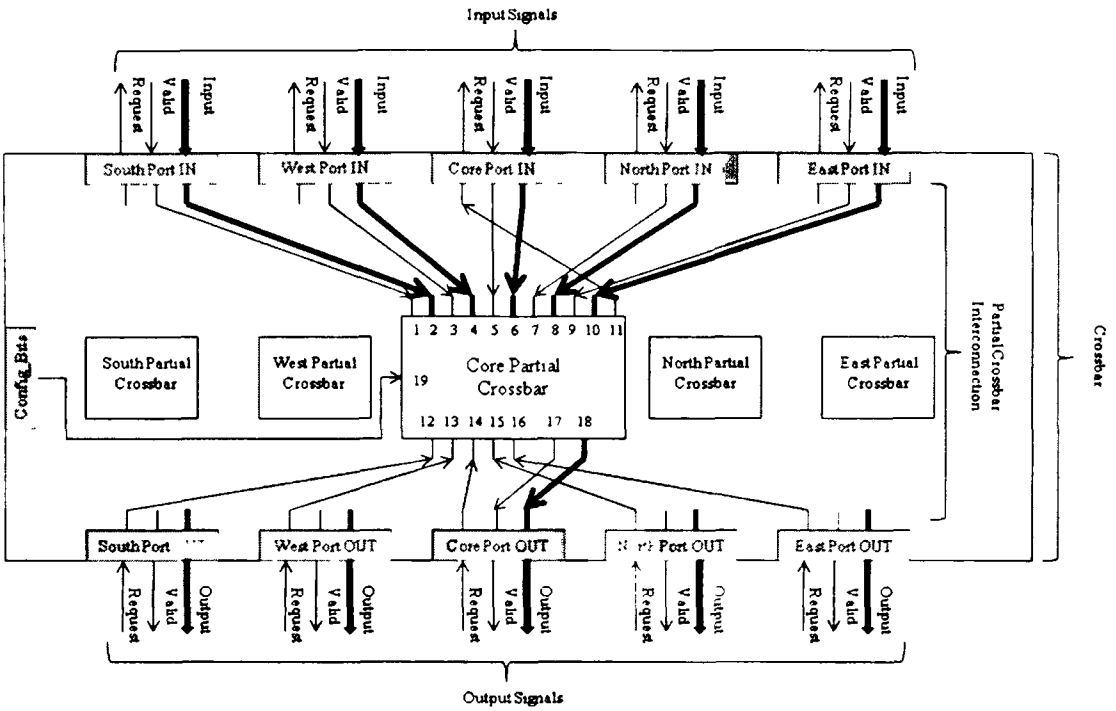


Figure 4.4: Crossbar Interconnection

The block diagram of the partial crossbar is depicted in Figure 4.3 with Table 4.1 giving a brief description of the connection assignments. Before partial crossbar is operational, it will first receive a routing scheme (from *Config_Bits,* connection assignment 19) to program its interconnection according to the scheme. Partial crossbar does not perform any information processing which reduces latency and the unused paths are inactive which helps to reduce energy consumption.

Figure 4.4 demonstrates how a specific routing scheme is implemented in the partial crossbars. For clarity, the figure shows only the connections for the core partial crossbar, while all partial crossbars are connected in the same manner. As mentioned earlier, each partial crossbar is only responsible to route data from the input port, to its output port.

All the *Data* signals from the input port; North, East, South, West, and Core are connected to the partial crossbar on connection assignment 2, 4, 6, 8, and 10 respectively. The *Valid* signals from the input port connect to the partial crossbar on connection assignment: 1, 3, 5, 7, and 9 respectively and the *Request* signals from the output port

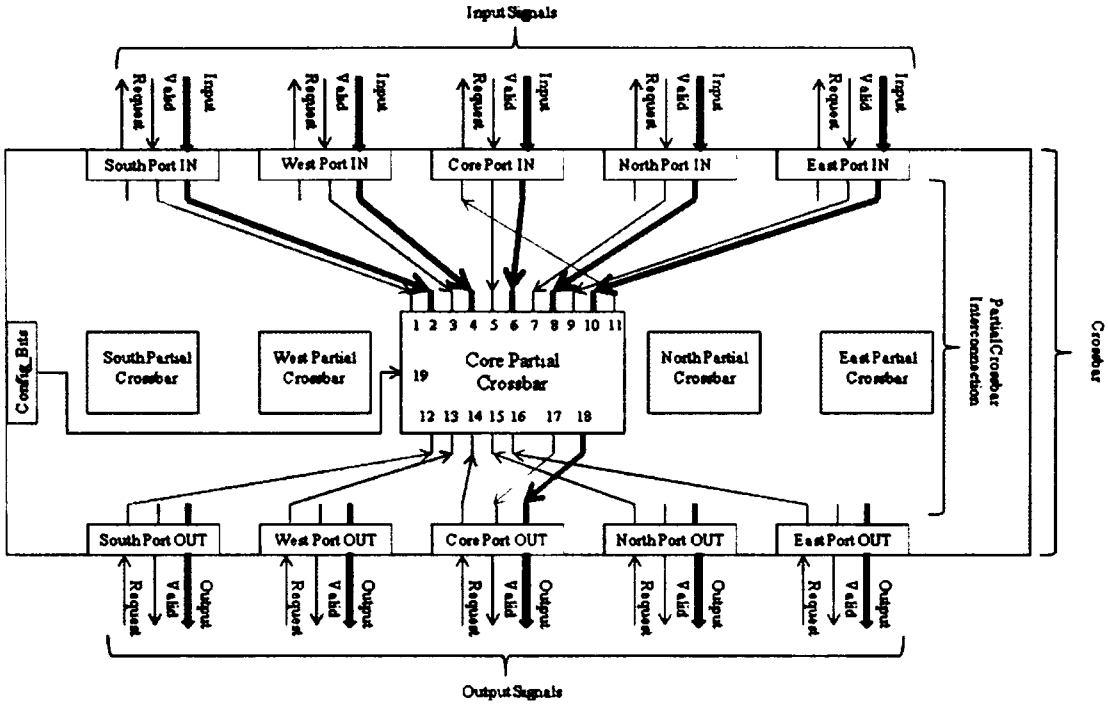


Figure 4.5: Routing South Port to Core Port

connect to the partial crossbar on connection assignment: 12, 13, 14, 15, and 16 respectively.

In the example depicted in Figure 4.5, the core partial crossbar is configured to receive data from the south input port and routed it to the core output port. All the connections are involved in this example is shown in red. The partial crossbar takes the *Data* signal and the *Valid* signal from South input port; connections assignment 1 and 2 respectively, and connects it to the core output port, connections assignment 17 and 18 respectively. The partial crossbar connects the *Request* signal from the south output port, connection assignment 12, to the core acknowledge input port, connection assignment 11. This MIN interconnection allows parallel transactions of all five ports, which eliminates the need for buffering resulting in the decrease of data latency and area cost.

4.1.2 Crossbar Configuration

In the previous section, we described the interconnection, layout and functionality of the partial crossbar. In this section, the process of configuring these partial crossbars is

described. Before the transmission of data, each router receives a configuration control word that holds the routing information for the router, shown in Figure 4.6.

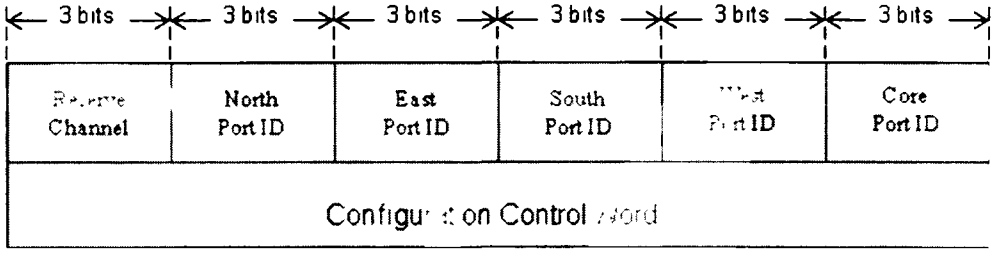


Figure 4.6: Configuration Control Word

Table 4.2: Port Conversion Table

Port	Port ID	Binary Representation
North	0	000
East	1	001
South	2	010
West	3	011
Core	4	100

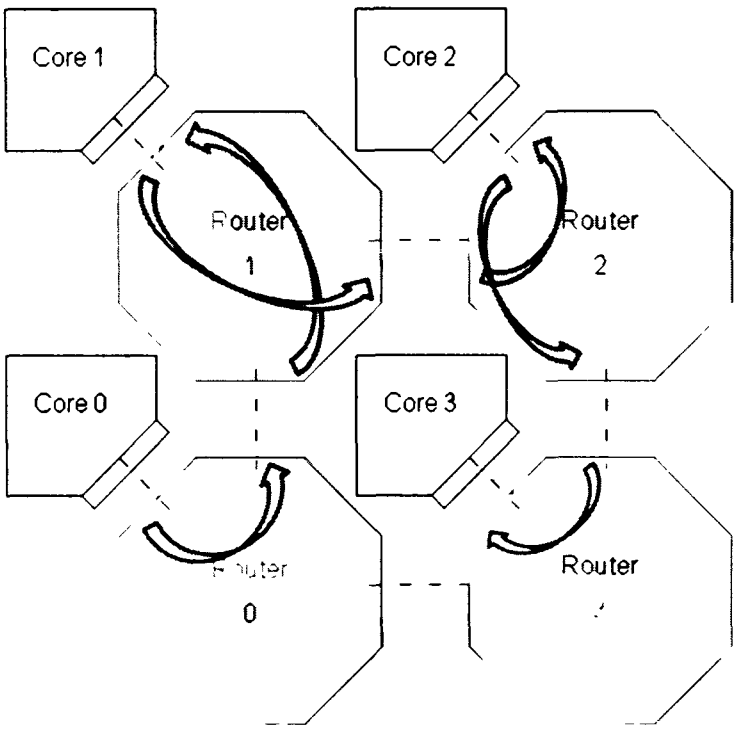


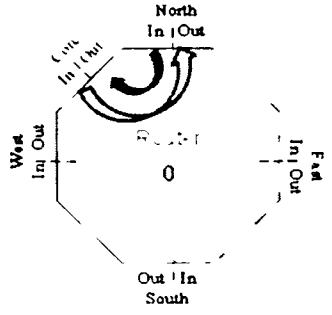
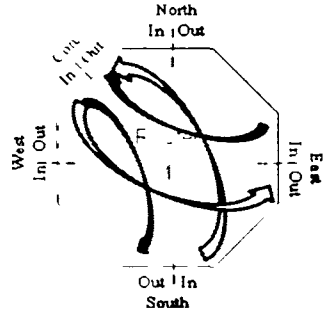
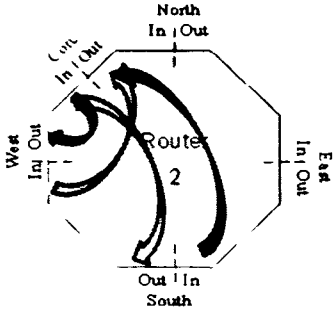
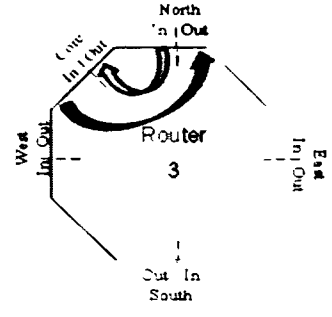
Figure 4.7: Example of a Test Scenario

Each configuration control word is 18 bits which is divided into 6 fields of 3 bits each, depicted in Figure 4.6. Each field is designated for a partial crossbar, containing an ID port to instruct the partial crossbar as to which input and output ports to use.

For a better understanding of how the configuration control word is determined, an example, illustrated in Figure 4.7, is used to demonstrate this process. In this example, data needs to be transmitted from *Core 0* to *Core 1*. Data from *Core 1* needs to travel to *Core 2* and data from *Core 2* travels to *Core 3*. This example will help us show how the configuration bit values are determined.

Table 4.3 shows the operations that are taking place in each router. A white arrow represents the path the data needs to travel while a solid black arrow represents the reserved communication channel.

Table 4.3: Configuration Bits Value for Test Scenario

Router	Scenario	Configuration Bits Value
0		
1		
2		
3		

Take the first scenario; data from *Core 0* needs to travel to *Core 1* which has to go through the north port. In the sub-packet for the north partial crossbar. the port ID for core ("100") is placed while the other ports service is not needed so we placed an "X" for don't care. *Core 0* will be expecting communication signals from *Core 1*, so in the reserve channel sub-packet, a port ID of "000" is placed because the communication signals coming from *Core 1* have to go through the north port.

In the second scenario, more operations are taking place. First, data coming from *Core 0* needs to go to *Core 1* and data coming out of *Core 1* need to travel to *Core 2*. A binary representation of "010" is placed in the *Core* sub-packet to establish a connection between the south input port to the core output port. Second, *Core 1* will need to communication with *Core 0* in the south direction through the south port. Then, the core ID is placed in the south partial crossbar sub-packet. Third, data from *Core 1* need to be route to *Core 2* in an east direction. The core ID is placed in the east partial crossbar sub-packet. Lastly, a communication channel is reserved for the east port since the communication signals from *Core 2* is coming from the east port and the rest is left unused.

In the third scenario, incoming data from *Core* 1 is coming from the west port will need to be routed to *Core 2* and also data from *Core 2* will need to be travel to *Core 3* through the south port. Thus, west ID and core ID is placed in the core sub-packet and south sub-packet respectively. *Core 2* will be sending communication signals to *Core 1*, so the core ID is place in the west sub-packet. Lastly, the reserve channel will reserved the communication signals coming from *Core 3* from the south port.

Finally, data from *Core 2* comes in from the north port and the communication signals from *Core 3* will also be going through the north port. Therefore, north ID is placed in core sub-packet and the core ID is place in the north sub-packet. This concludes our example which shows the process of configuring each router.

4.2 Configuration Block

The configuration block passes the configuration information it received from the PC host to the router. Using a handshake protocol, the PC host will signal the configuration block for a new configuration. The configuration block recognize this by stopping all operations by sending a low *Ready* signal to the router and send a request to the PC host for the configuration bits. NoC can only resume its operation only after all routers are configured.

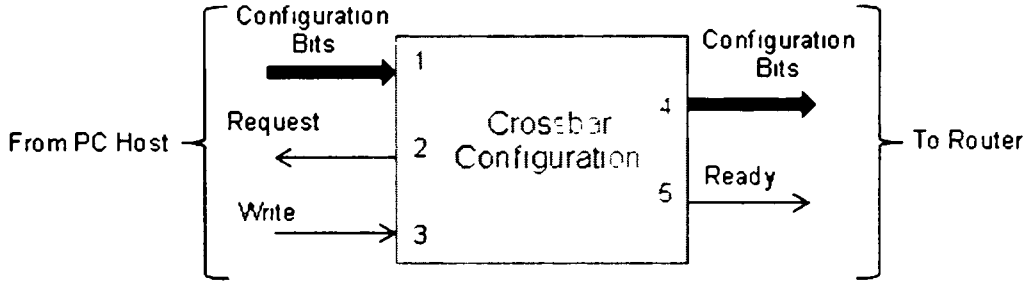


Figure 4.8: Crossbar Configuration Block

Table 4.4: Connection Assignment for Configuration Block

Connections Number	Description
1	*Write* signal from PC host to signal write operation
2	*Request* signal to PC host to request a write operation
3	*Configuration Bits* value send by the PC host
4	*Ready* signal to signal router configuration is complete
5	*Configuration Bits* value send by the configuration block

4.2.1 Primary latency

As discussed in chapter 2, the disadvantage of implementing a circuit switched protocol is the initial latency incurred for establishing the virtual circuit. This causes the blocking of all routers because the physical link is now reserved for setting up the new connection. For our application which requires large quantity of data and frequent data transfers, the

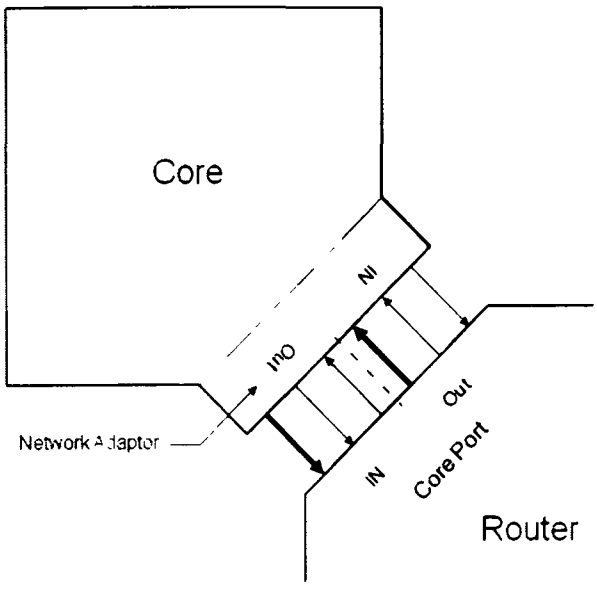


Figure 4.9: Block Diagram of Network Adaptor

initial latency becomes insignificant. Our proposed NoC system have nine routers that need to be configure and the execution time to configure all nine routers is 10.8 milliseconds, at a clock speed of 50 MHz.

4.3 Network Adaptor

The task of a network adaptor is to integrate the core computational hardware block into the NoC system. The network adaptor is responsible for communicating with the router to receive data as input to the core and to transmit data as output from the core.

4.4 Links

Links are physical wires that transfer data from one node to another. Links are very susceptible to the degradation of signal integrity caused by intrinsic and extrinsic noise from the circuit, when driving the signals on a long wire. Although they may not be a major component, links are the backbone of the infrastructure that need to be optimized for signal integrity. Therefore, the protection of physical wires from noise and minimizing power consumption are important design goals that should be considered at the physical level. These issues are more important when developing NoC for ASIC implementation.

4.5 Summary

In this chapter, detailed descriptions of the functionality of each NoC blocks are given. The implementation and functionality of the router was described and examples were used to illustrate the operation of the router. Lastly the important functions of the network adaptor and physical links were described. In chapter 5, the design methodology for the proposed NoC implementation is described. We also describe the application mapping process for evaluating the proposed NoC.

Chapter 5

Experimental Evaluation Framework

This chapter starts with a discussion of the design methodology for implementing an Network-on-Chip (NoC) system. This methodology also facilitates rapid prototyping and exploration of various aspects of NoC implementation. This is followed by a description the Celoxica DK tool, used for NoC implementation. Then, we describe how a Microblaze-based system is specified using the Celoxica Handel-C hardware description language. Lastly, the applications mapping process for mapping two test applications into the target systems, bus-based system and NoC-based system is presented.

5.1 NoC Specification and Implementation Methodology

For this research, it is important to define a framework that helps guide this research for exploring the design space for NoC implementation. As discussed earlier in Chapter 3, any NoC architecture implementation can be described at several layers of abstraction. Figure 5.1 illustrates the design space for NoC architectures.

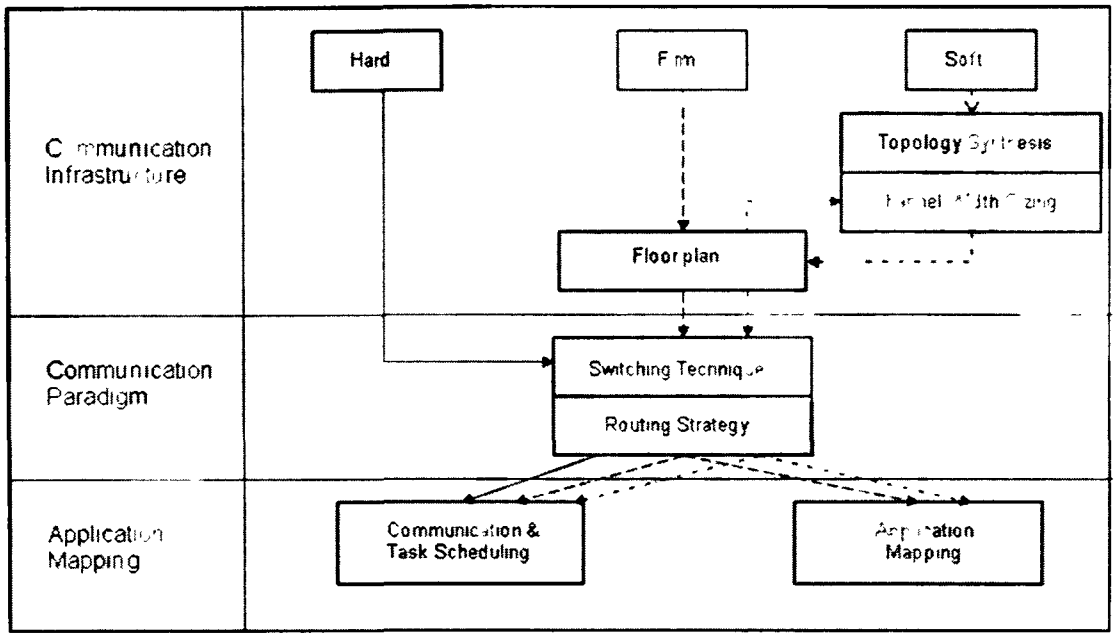


Figure 5.1: Design Space for NoC Architectures [2]

5.1.1 Communication Infrastructure

Communication infrastructure determines the communication architecture for providing optimal performance, with particular attention to design trades-offs to meet the performance needs of the chosen application. Although customized NoC is highly desirable for great improved performance, they bring other issues into focus such as physical links optimization (driving uneven wire length) and low scalability (irregular floor planning) [2]. While low complexity architecture such as the mesh can provides an excellent platform for rapid prototyping for many applications, its performance may not be adequate for some higher end applications. Also, communication infrastructure is involved with network floor planning, the placement of various NoC components for optimal performance and routability.

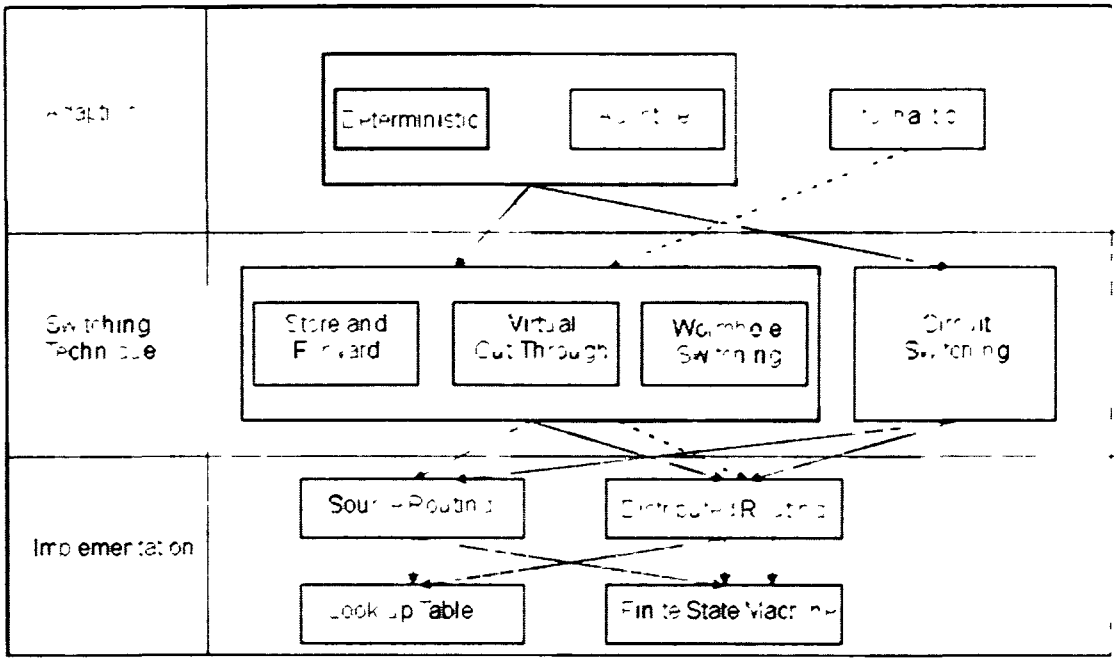


Figure 5.2: Design Choices for Communication Paradigm [2]

5.1.2 Communication Paradigm

Network does not dictate the behavior of the traffic flow, rather the traffic flow dictates how the network behaves. The performance of the network is greatly affected by the selection of the routing schemes and routing strategies as depicted in Figure 5.2.

Compared to adaptive routing, deterministic routing requires fewer resources while guaranteeing an orderly data arrival. Deterministic routing is free of deadlock and livelock and is more appropriate for traffic that is predictable. On the other hand, adaptive routing provides better area utilization and lower latency by allowing alternate paths based on the network congestion [27]. But this routing strategy is highly susceptible to deadlock and livelock discussed in Chapter 3.

Deterministic routing can provide high QoS with low area overhead, is a promising alternative to adaptive routing such as wormhole switching which allocate more resources such as buffers for routing. A drawback of deterministic routing such as circuit switched (CS) is they are static in nature which works well when traffics is predictable. Therefore, further research should be done exploring hybrid routing

schemes, a combination of deterministic and adaptive routing that can take advantage of both systems.

5.1.3 Application Mapping

Application mapping is involved with integrating the application into the NoC system and task scheduling. Designers have to decide the optimal arrangement of IP cores into the network for optimizing certain design metrics such as communication congestion, high bandwidth, power consumption, and area utilization. For optimal mapping, the task scheduling and the IP mapping should be ideally performed in parallel [2].

5.1.4 Celoxica Handel-C

Our main research goal is to design an NoC-based system to be used for comparison with a bus-based system using the Celoxica's Handel-C.

Handel-C language is used to illustrate the advantages of using a C-like language to speed-up the design process. Celoxica Design Kit (DK), a compiler for Handel-C, allows designers to develop an algorithm in software for targeting hardware implementation. In the design process, Celoxica's DK graphical user interface (GUI) along with third party FPGA design tools (e.g. Xilinx EDK and ISE) are used together to produce automatically, equivalent data path and control logic expressed in the form of an EDIF netlist. This EDIF output can then be taken as the input to physical design tools targeting FPGAs provided by Altera Corp. [15] or Xilinx Inc. [16], or alternatively, to physical design tools targeting ASICs [28]

Celoxica's Handel-C environment does not come without some drawbacks. Issues were encountered during the design process while transporting design data from the DK environment to the Xilinx environment, following Celoxica's design flow. This problem was easily be resolved by applying missing patches needed for the Xilinx EDK tool.

5.2 Xilinx Microblaze Soft-Core Processor

Microblaze (MB) [21] was been selected as the soft-core processor for this research that is optimized and targeted for Xilinx's FPGAs family. Therefore a brief description of the Microblaze architecture [22] is given. The MB processor has four bus interfaces:

1. Instruction Memory Bus (IOPB) for the connection of internal or large external instruction memory
2. Data OPB bus (DOPB) for the connection of Slave (OPB) peripherals, internal memory and large external data memory
3. Instruction Local Memory Bus (ILMB) for the connection of fast local block RAMS that acts as an instruction memory for the Microblaze Core
4. Data Local memory Bus (DLMB) for connection of fast local block RAM that acts as data memory for the Microblaze core

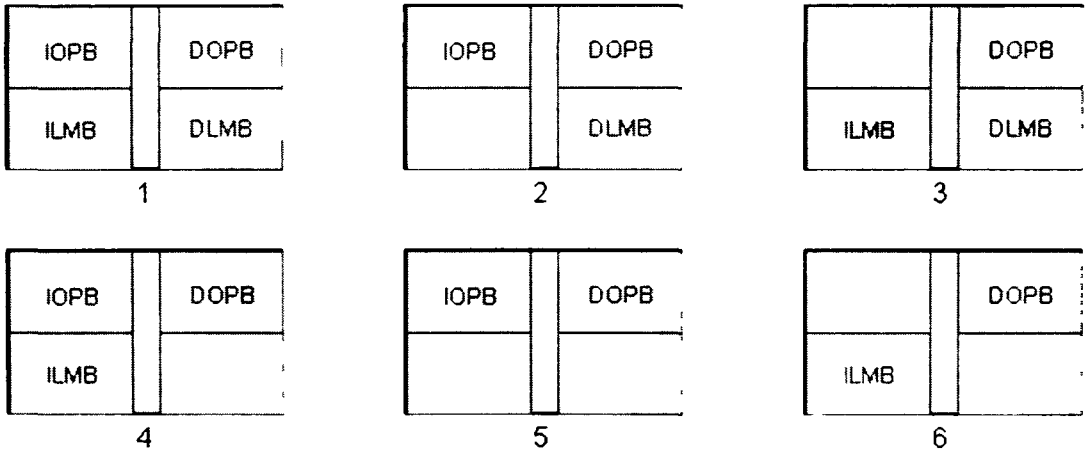


Figure 5.3: Microblaze Bus Configurations

There are six different configurations with which the Microblaze core can operate; they are shown in Figure 5.3.

1. IOPB, DOPB, ILMB, and DLMB are all enabled
2. Only IOPB, DOPB, and DLMB are enabled
3. Only DOPB, ILMB, and DLMB are enabled
4. Only IOPB, DOPB, and ILMB are enabled

5. Only IOPB and DOPB are enabled
6. Only DOPB and ILMB are enabled

The designed system used configuration 3 which is more suitable for applications where the code can fit into the on-chip block RAM and more memory is required for data.

5.2.1 On-Chip Peripheral Bus

The on-chip peripheral bus (OPB) is one element of IBM's CoreConnect architecture, and is a general purpose synchronous bus designed for easy connections of on-chip peripheral devices. It includes the following features:

1. 32-bit or 64-bit data bus
2. Up to 64-bit addressing
3. Supports 8-bit, 16-bit, 32-bit, and 54-bit slaves
4. Supports 32-bit and 64-masters

The OPB is designed for easy connection of on-chip peripheral devices. It provides a common design point for various on-chip peripherals. Celoxica provides libraries to ease the design and integration of OPB devices in an embedded processor system. This library allows the designer to connect Handel-C coded OPB peripherals to the Microblaze processor based system.

5.2.2 Specifying a Microblaze-Based System Using Handel-C

The architecture of the Microblaze-based system used in this research using Handel-C at the top-level of the design is demonstrated in Figure 5.4. OPB Bus at the top-level is represented in the form of a macro that communicates with the actual Microblaze OPB bus through the OPB_HC Bridge. In this manner, OPB peripherals are designed at a higher level of abstraction.

Choosing Handel-C slave peripherals to be at the top-level rather than using Microblaze at the top-level enables the designer to address some design disadvantages such as:

1. No direct access to the data and instruction OPB buses

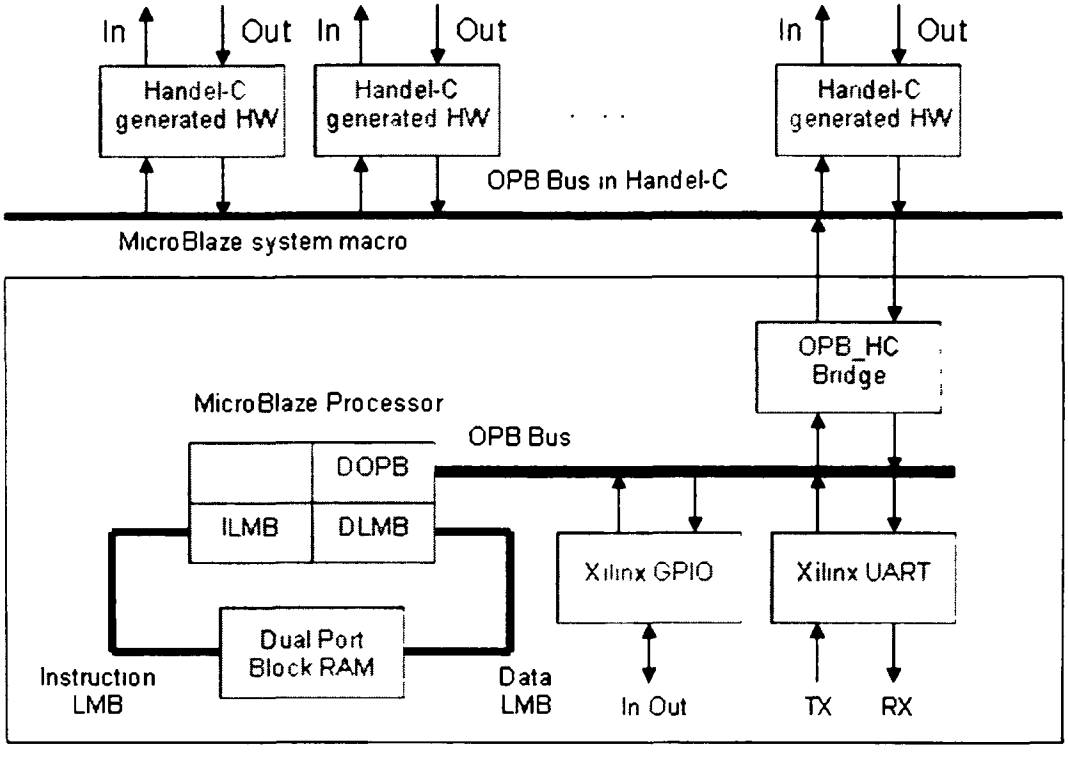


Figure 5.4: Integration of MB and Handel-C Slaves with Slaves at the Top-level

2. A set of support files must be created and current support files for current OPB peripherals must be updated when a new OPB peripheral is added.

Figure 5.4 shows the architecture of the system being designed. The Microblaze is represented by a rectangle with four quadrants and the space outside the large rectangle is hardware specified in Handel-C environment with full access to the OPB and input/output to/from the processor.

5.3 Charged-Couple Device Benchmark

A Charged-Coupled Device (CCD) is a special senor that captures an image using a light-sensitive silicon solid-state device composed of many small cells. Light falling on a cell is converted into a small amount of electric charge, which is then measured by the CCD electronics and stored as a number. The number usually ranges from 0, meaning no light, to 256 or 65,535, meaning very intense light per pixel.

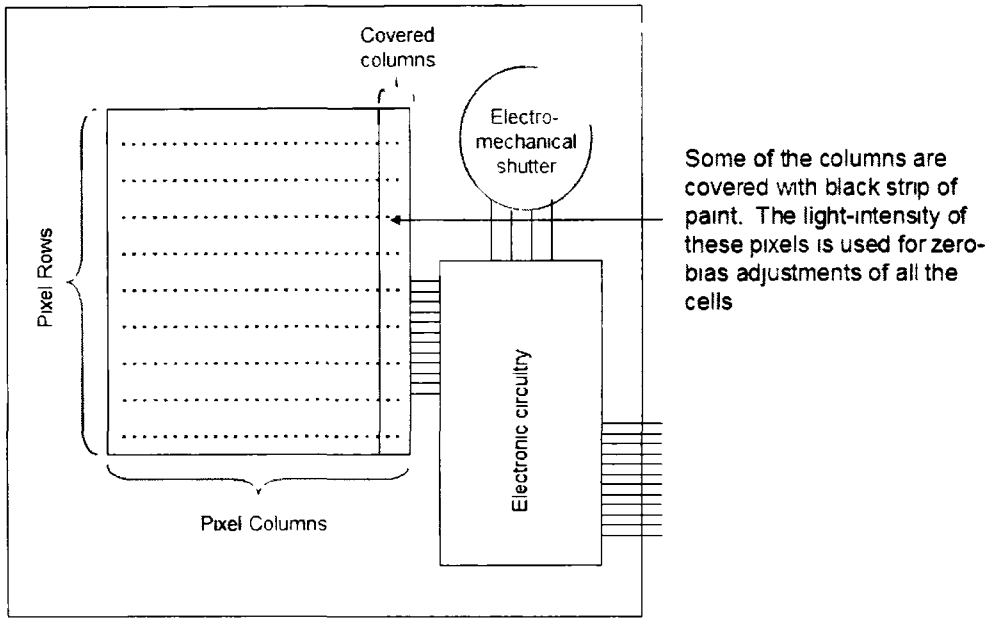


Figure 5.5: Internal of a CCD [26]

Due to manufacturing errors, the converted values from the light-sensitive cells need to be calibrated. This error, called the zero-bias error, is typically the same across columns but different across rows. For this reason, some of the left most columns of a CCD's light-sensitive cells are blocked by a strip of black paint shown in Figure 5.5. The actual intensity registered by these blocked cells should be zero.

If these blocked cells doe not registered a zero, the zero-bias error can be correct by taking the average of these blocked cells of the same row and subtracting each element from the same row. This process is repeated until all the rows have been corrected for zero bias errors.

After zero bias errors correction, the image is compressed to reduce the number of bits needed for storing the image in memory. Compression allows us to store more images in limited amount of memory. Image data is divide into many 8x8 blocks where each block is encoded using Forward Discrete Cosine Transform (FDCT) for its high compression ratios [26].

After compression, the next step is quantizing to further compress the image by reducing the bit precision of encoded data, thus achieving higher compression ratios at the expense of lower image quality. The high-level functionality of the CCD can be described

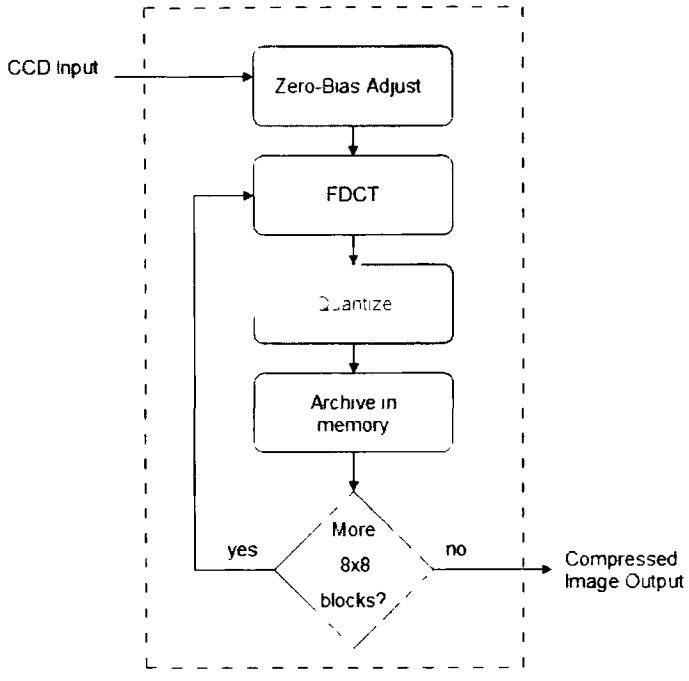


Figure 5.6: Flow-Diagram for CCD

using the flow-diagram from Figure 5.6. The four functions are Zero-Bias adjust, FDCT, quantize, and archive in memory.

5.3.1 Pure software system

The software prototype is developed and executed on a PC which consists of 5 programs describing the functionality of the CCD. Figure 5.7 described the flow-diagram of the high-level model of the executable model composing of five models; CCD, CCDPP, CNTRL, CODEC, and UART. Readers can refer to appendix A for a full description, describing the execution of the program simulating a real CCD with the executable models in C language [26].

5.3.2 Bus-based system

Microblaze system relies on a number peripherals connected to the Microblaze OPB bus. To implement the CCDBM, four function blocks were created as separate entities in Handel-C as slave peripherals for Microblaze, and connected to the OPB bus. The four

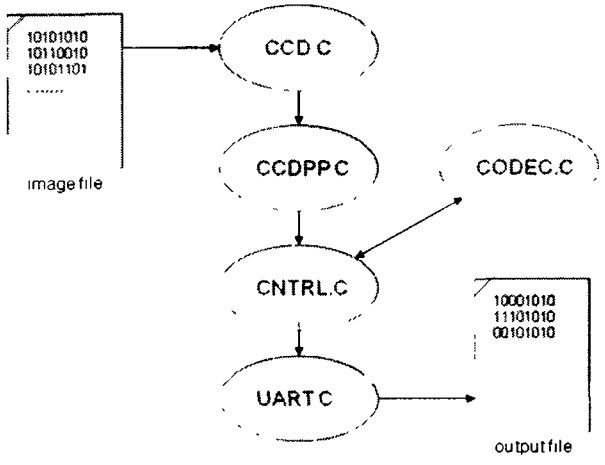


Figure 5.7: Flow-Diagram of the Executable Model of the CCD [29]

slave peripherals are; Zero-Bias peripheral, FDCT peripheral, Quantize peripheral, and Memory Archiving peripheral are depicted in Figure 5.8.

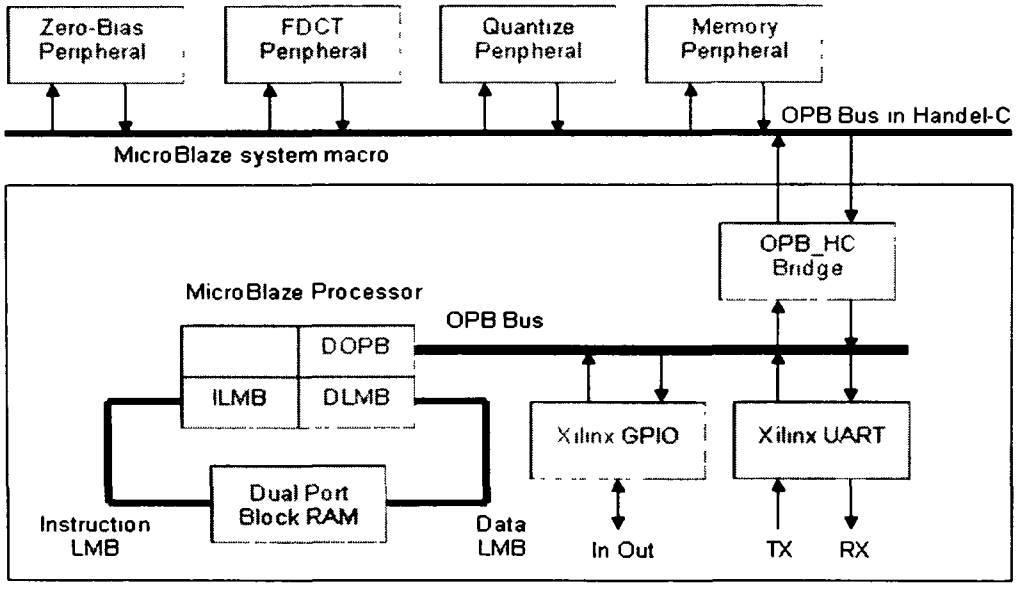


Figure 5.8: Integration of MB and CCDBM Slaves Peripherals

The slave peripherals are connected to the Microblaze OPB bus and communicate with the Microblaze processor through the OPB-HC Bridge which is a Handel-C generated hardware that brings the OPB signals to the Handel-C environment. This is

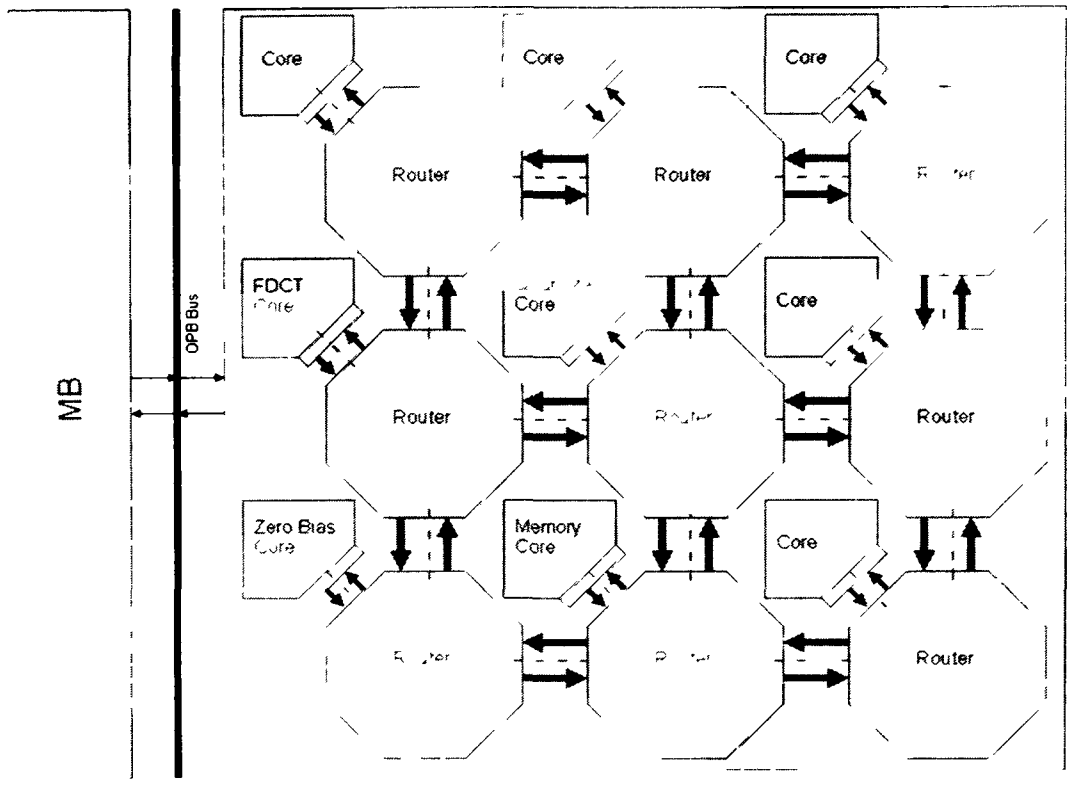


Figure 5.9: Integration of MB and NoC

how a bus-based implementation of a Microblaze system was implemented, for comparison with the NoC-based system.

5.3.3 NoC-Based Implementation of a Microblaze System

In the bus-based system, function blocks are created as separate slave peripherals. In contrast, in NoC-based system, the function blocks are mapped into hardware cores within the NoC and the whole NoC is treated one peripheral as shown in Figure 5.9. In the NoC, the four cores are; Zero-Bias core, FDCT core, Quantize core, and Memory Archiving core.

For simplicity, the bottom left router is use as reference node or a starting node. The first core which is the Zero-Bias core is connected to the reference router. From a clockwise direction, the next three cores then follow: FDCT core, Quantize core and Memory core. How the cores are arranged is determined by the designer. But in general,

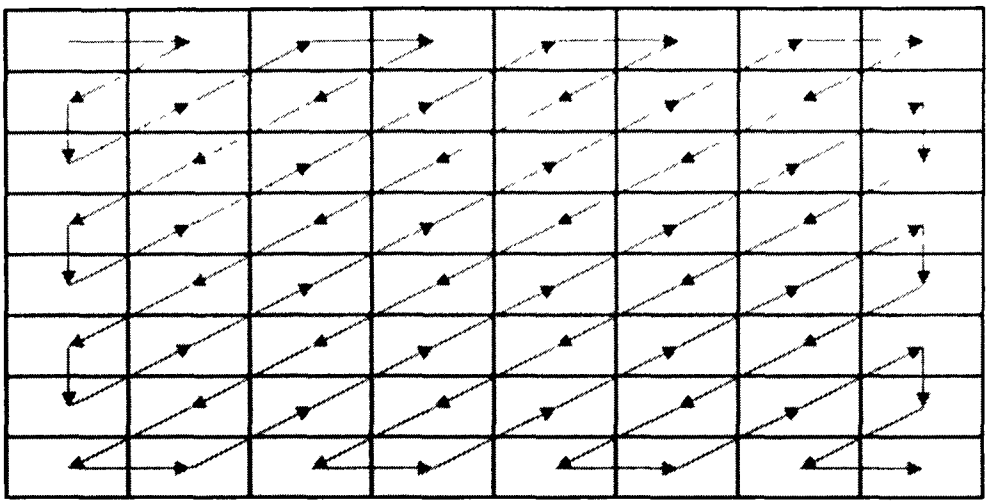


Figure 5.10: JPEG Encoding Sequence for a Block of 8 x 8 Pixels

it is a good idea to place cores that have high interaction close together because driving long wires increases latency and power consumption.

5.4 JPEG Benchmark

The whole process of the CCD benchmark is to capture the image and compress the image for encoding. Now, the compressed image needs to be encoded that can be serially transmitted and for storage by other external devices such as a computer. Therefore, this step is to encode the data into JPEG format where the values are serialized in a zigzag pattern shown in Figure 5.10.

5.4.1 Pure software system

An encoding module was created to complete the encoding process of an image in a digital camera. This module imports the compressed data that goes through the CCD process and following the zigzag pattern, the data is encoding into JPEG format. Once the data is JPEG encoded, it is ready to be store in memory and its format can be read by external device.

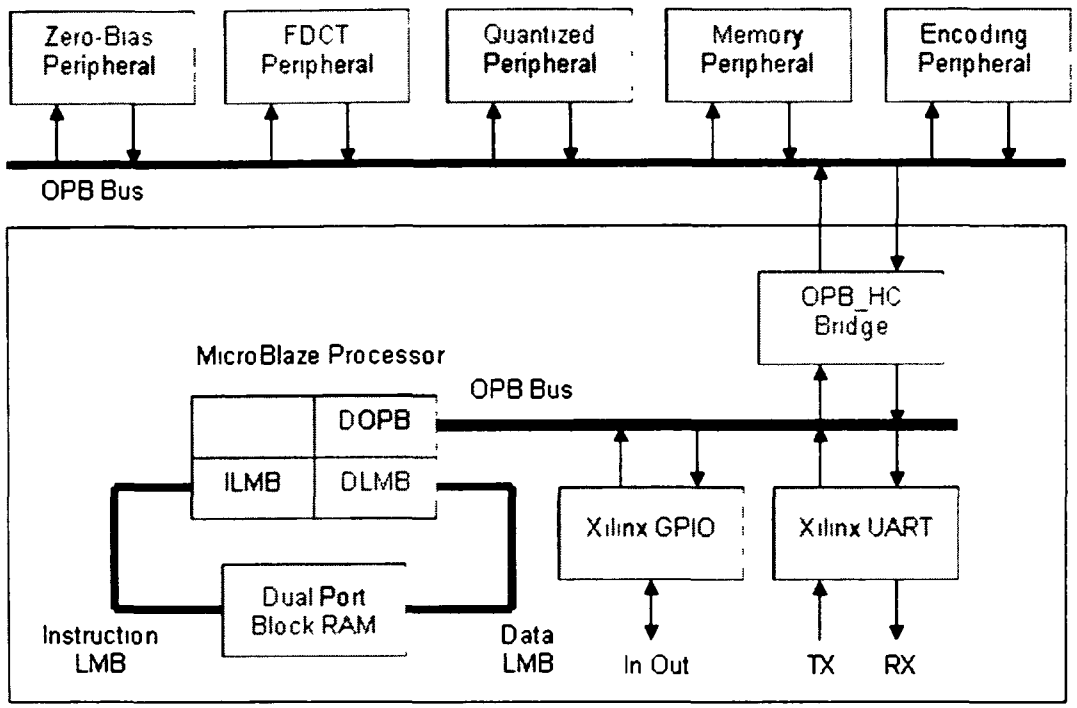


Figure 5.11: Integration of MB and JPEG Slave Peripherals

5.4.2 Bus-based system

To complete the JPEG benchmark, in addition to the CCD peripherals, an encoding peripheral was created addition to the CCD benchmark shown in Figure 5.11. The advantage of using Celoxica's Design Kit (DK) graphical user interface (GUI) along with third party FPGA design tools (e.g. Xilinx EDK and ISE) is the simplicity of incorporating new peripheral on existing system. For creating a new peripheral, the hardware component is designed which includes libraries provided by the DK tools to connect the peripheral to the OPB bus. Then, changes are made to the memory addresses to include the new peripheral. This demonstrates the power and ease of using an ESL tool in designing a processor-based system.

5.4.3 NoC system

For the NoC system, a JPEG encoding core was created and placed on an empty core. Since most of the data transactions for the encoding core are made between the memory

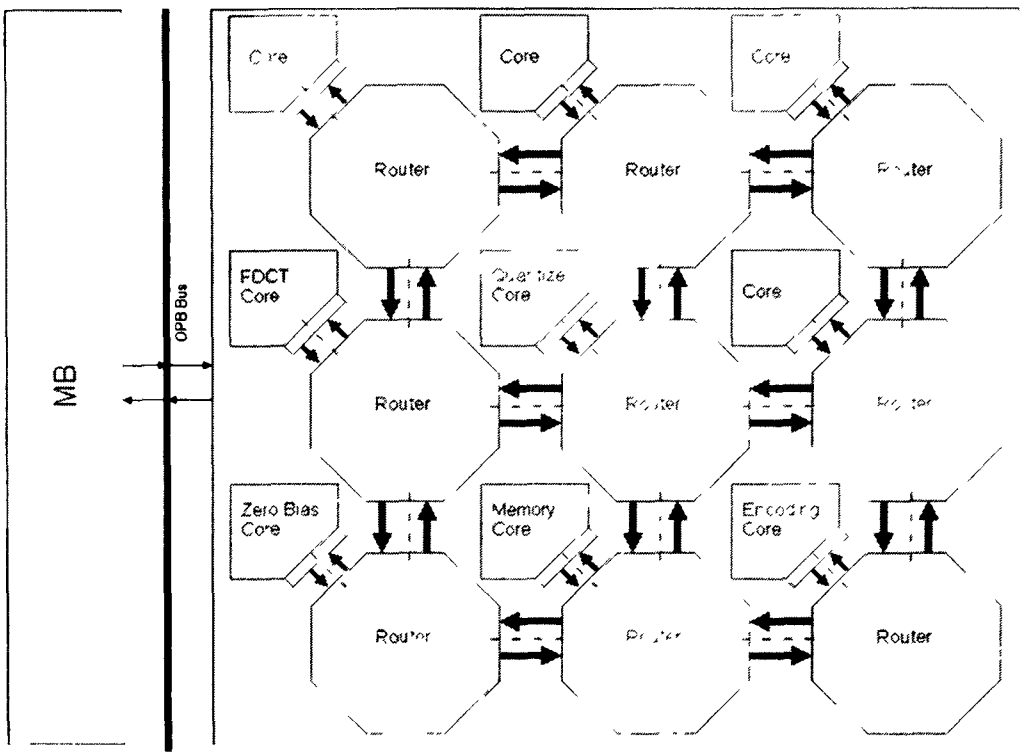


Figure 5.12: NoC System for JPEG

core and the encoding core. Logically, the encoding core is placed next to the memory core shown in Figure 5.12.

5.5 Summary

In this chapter, the design methodology for NoC implementation was first presented, followed by a description of the design tools and the environment in which the NoC was designed and implemented. A detailed description of the implementation of bus-based and NoC-based Microblaze systems in the Celoxica Handel-C environment was presented. Finally, the application mapping process for the two test applications was presented. These two applications were mapped into three types of systems: purely software-based system, bus-based system, and NoC-based system. The next chapter discusses the results obtained from application mapping.

Chapter 6

Comparison of NoC-Based System and Bus-Based System

In this chapter, the implementation results for NoC-based system and bus-based system are presented and compared. First, the processing cycle of the two test applications implemented on the NoC-based system and bus-based system is described. Next, the area cost of implementing the NoC-based system and the bus-based system targeting an FPGA is discussed. Finally, speed comparison is presented in two parts; first a comparison between an NoC-based system and a purely software system; and second a comparison between an NoC-based system and a bus-based system.

6.1 Processing Cycle

An NoC-based system increases performance by allocating more resources for communication tasks through pipelining. The following section describes the processing cycle of the two systems; a bus-based system and an NoC-based system running two applications; CCDBM and JPEG.

6.1.1 CCDBM Processing Cycle

Figure 6.1 shows the CCDBM processing cycle for a bus-based system. In the bus-based system, the processing cycle starts when Zero-Bias block receiving raw data from the host PC and performing zero-bias adjustment. After the raw data is adjusted, the zero-bias

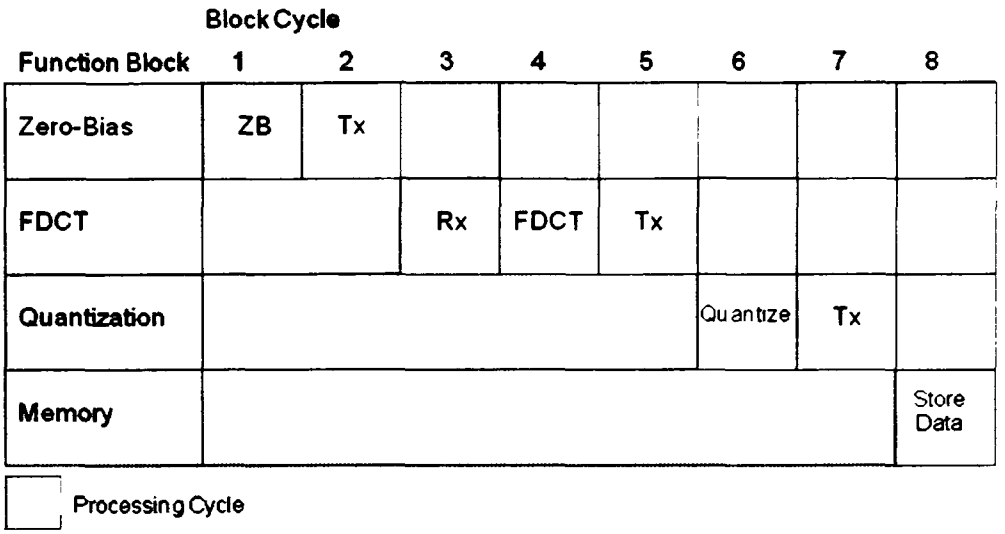


Figure 6.1: Bus-Based Processing Cycle for CCDBM

data is transmitted to Forward Discrete Cosine Transform (FDCT) block in the next block cycle.

In a bus-based system, only one peripheral is activated at any given time. Therefore, data is first temporally stored until the next function block is activated. After the FDCT block has received the adjusted data, the FDCT block will run its computational hardware and the data is transmitted to the next function block on the next processing cycle. On block cycle 6 and 7, the Quantized block is activated for data quantization and the quantized data is transmitted on block cycle 7, to be stored in the Memory Archive block. It takes a total of 8 block cycles to complete the processing cycle of an 8 x 8 block. And this procedure continues until all 8 x 8 blocks are processed.

In contrast, an NoC system allows for more parallelism. The processing cycle for CCDBM implemented on a NoC-based system is depicted in Figure 6.2.

The Zero-Bias block starts after receiving raw data from the host PC. On block cycle 1, Zero-Bias block will perform zero-bias adjust and then the data is transmitted on the next block cycle. Since all the function blocks are active, the Zero-Bias block can transmits the data directly to FDCT block without the need of the data to be temporally stored, saving one block cycle.

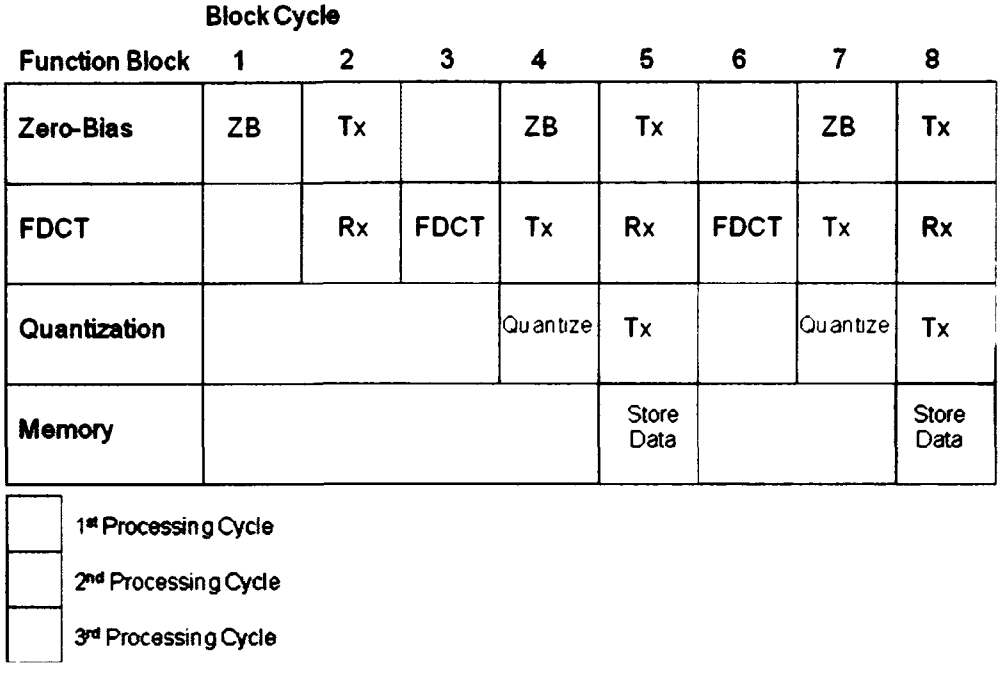


Figure 6.2: NoC-Based Processing Cycle for CCDBM

On block cycle 3, the FDCT function block performs its computational hardware and the data is transmitted to Quantized block on block cycle 4 while Zero-Bias block performs zero-bias adjust on a new set of data block. As the first processing cycle is completed, a new processing cycle have already been started and this cycle continue until all 8 x 8 blocks are processed.

In the bus-based system, it takes 8 block cycles to processed 8 operation blocks while in the same block cycles for NoC system, it has processed 19 operation blocks; an NoC speed up of 2.375.

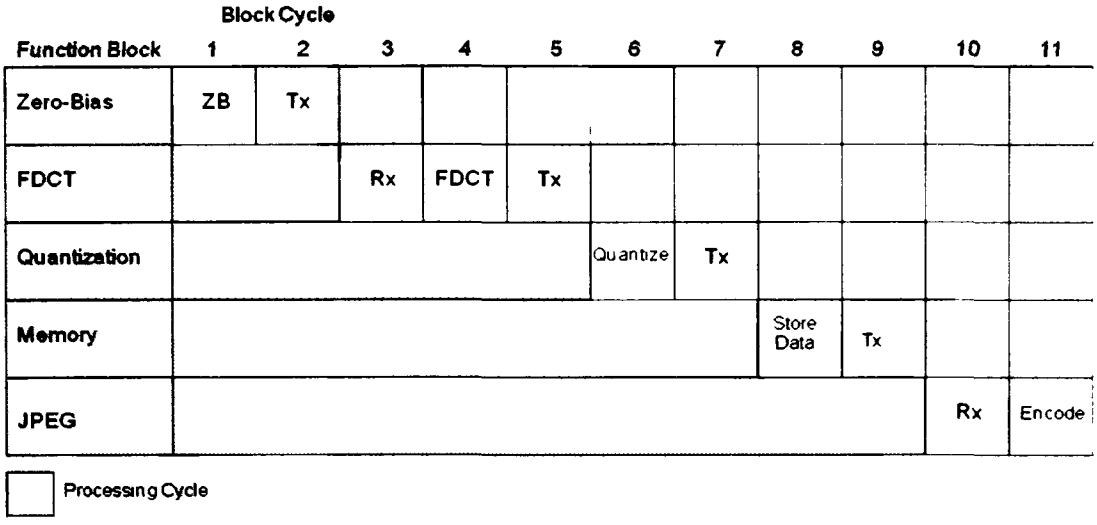


Figure 6.3: Bus-Based Processing Cycle for JPEG

6.1.2 JPEG Processing Cycle

The processing cycle for JPEG application is implemented on a bus-based system and on an NoC system are depicted in Figure 6.3 and Figure 6.4 respectively. JPEG application is made up of 5 function blocks that include the 4 function blocks from the CCDBM with an additional of a JPEG function block. We can see that with the addition of the JPEG function block, the processing cycle is increased by 3 to a total of 11 block cycle.

In the NoC system, the addition of the JPEG block increased the processing cycle by 2 to a total of 7 processing cycle. Comparing both systems, the bus-based system takes 11 block cycle to completed one processing cycle. While in an NoC system, it has processed 33 operation blocks in the same amount of block cycles, an NoC speed up of 3. What we can conclude from this that the NoC-based system increase performance by allocating more resources to the communication aspect of the network for pipelining. Also, we can see a greater NoC speed up for larger system with greater number of function blocks.

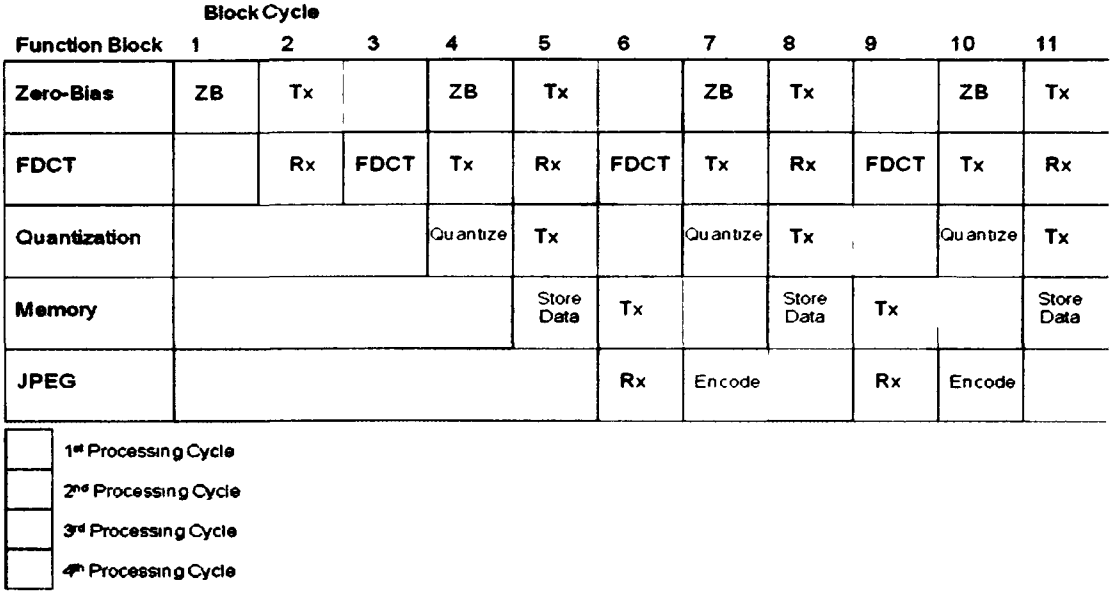


Figure 6.4: NOC-Based Processing Cycle for CCDBM

6.2 Implementation Results

The implementation results are presented in Table 6.1 which shows the speed performance and area utilization of each system including area utilization of each individual computational hardware block.

6.2.1 Area Results

As mention earlier, MB is a 32-bit embedded soft-core RISC processor and from Table 6.1, at 2,061 slices, MB takes up 17% and 37% of the total logic hardware for the bus-based system and the NoC-based system respectively.

In the CCDBM application implemented on the bus-based system, the biggest peripheral in terms of area utilization is Zero-Bias peripheral at 4,189 slices. The reason for the large area cost is not only the Zero-Bias peripheral performs zero-bias adjustment, it also stores the raw data. The next largest peripheral is FDCT at 2,290 slices, where most of the computational operation for image compression takes place. This is followed by Quantize and Memory Archive peripherals at 1,281 slices and 1,985 slices respectively.

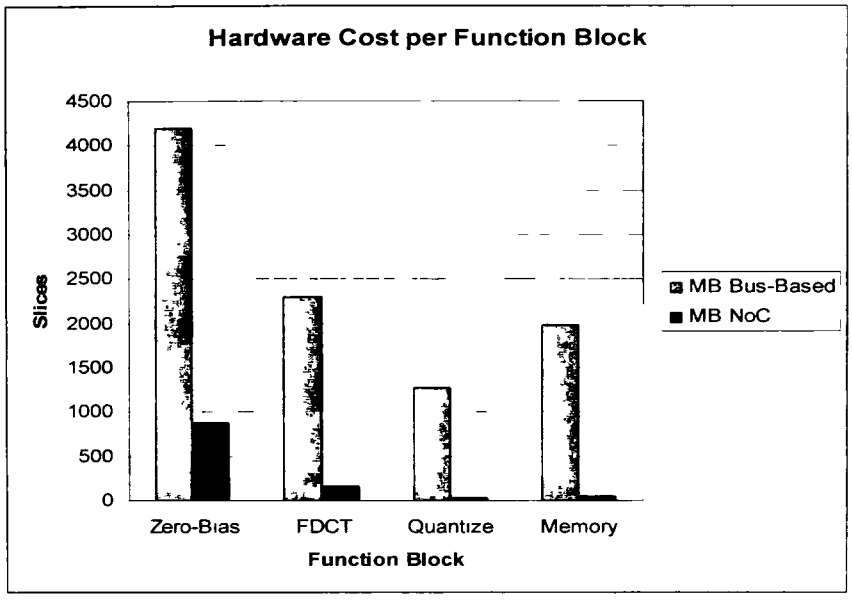
Table 6.1: Synthesis Results of Three Systems

	CCDBM			JPEG		
System	Bus-Based	NoC	Software	Bus-Based	NoC	Software
Channel Width	32 bit	16 bit	n.a.	32 bit	16 bit	n.a.
MB (slice)	2061	2061		2061	2061	-
NoC (slice)	-	2407			2407	-
Zero-Bias (slice)	4189	880	-	4189	880	-
FDCT (slice)	2290	153	-	2290	153	-
Quantize (slice)	1281	34	-	1281	34	-
Memory (slice)	1985	50	-	1985	101	
JPEG (slice)	-	-	-	1506	118	-
Total (slice)	11806	5585	-	13310	5599	
Hardware freq.	50 MHz	50 MHz	-	50 MHz	50 MHz	-
Execution time	18.65 ms	7.53 ms	12 ms	23.55 ms	7.54 ms	12.79 ms
NoC Speed-up	2.47		1.59	3.12	-	1.69

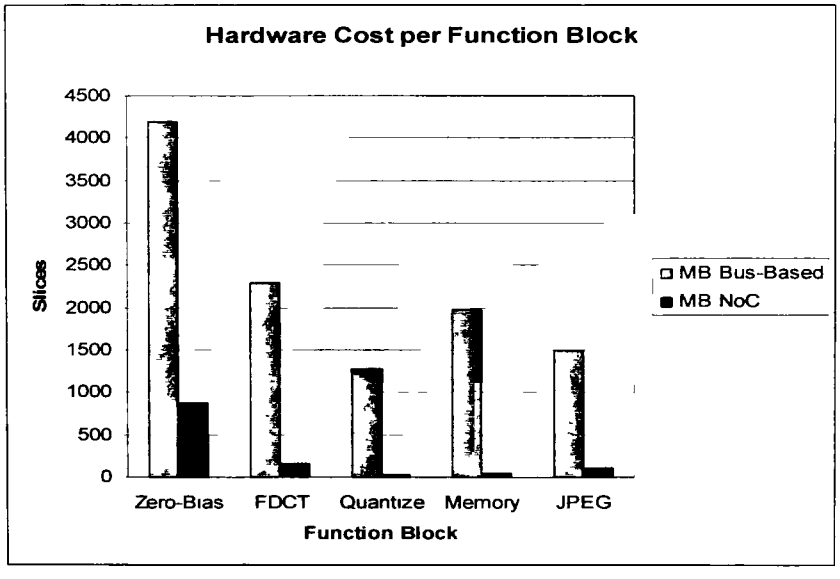
For NoC-based implementation of CCDBM, the NoC peripheral is the only peripheral that housed the whole NoC system. At 2,407 slices, it is comparable to industrial designs. The four cores; Zero-Bias, FDCT, Quantize and Memory Archiving takes up 880, 153, 34, and 50 slices respectively. At 5,585 slices of the total area utilized for NoC based system, it is much smaller compared to the bus-based system at 11,806 slices.

Figure 6.5a compares the area cost of the computational block of both systems implementing CCDBM. Both graphs clearly show that the peripherals in the bus-based system have larger area cost compared to the NoC-base system cores. For example, the Quantize peripheral in the bus-based system is 37 times larger than the Quantize core in the NoC-based system, while both computational blocks perform the same functions. The reason for the great variation in area cost is because implementing peripherals comes with great area overhead.

Looking at area utilization for JPEG implementation depicted in Figure 6.5b, it reinforces our finding that implementing peripherals comes with great area overhead. The total area utilization for JPEG implementation in the bus-based system and NoC-based system is 13,310 and 5,599 slices respectively.



a) CCDBM Application



b) JPEG Application

Figure 6.5: Area Comparison

6.2.2 Speed Results

We have demonstrated the advantage in area utilization of implementing an NoC-based system over bus-based system. In this section, we will focus on speed performance comparison of the two systems. Also, we compare speed performance between a hardware implementation of an NoC-based system to a purely software system running the same applications.

From Table 6.1, the software execution time of a purely software system implementing the CCDBM is 12 milliseconds compared to 7.53 milliseconds from NoC system, an NoC speed up of 1.59. For JPEG implementation, software execution time is 12.79 milliseconds and 7.54 milliseconds for NoC system, an NoC speed up of 1.69. This shows that hardware implementation is faster than the software system even though the MB runs at only 50 MHz compared to a CPU with clock frequency in GHz for software implementation on a PC.

For hardware comparison, the NoC-based system execution time is 7.53 milliseconds while the execution time for bus-based system is 18.65 milliseconds, an NoC speed up of 2.47 for CCDBM implementation. For JPEG implementation, the execution time is 7.54 milliseconds for NoC-based system and 23.55 milliseconds for bus-based system, an NoC speed up of 3.12. Even though the NoC system is not optimized since it is written in a high level language. This research has shows that the NoC-based system gives superior performance compared to the bus-based system.

6.3 Summary

In this chapter, the implementation results for the NoC-based system and the bus-based systems were presented. The advantages of using cores in NoC-based system over peripherals in bus-based system can have a tremendous impact on area cost for FPGA implementation. The total area cost for CCDBM applications in NoC system and bus-based system is 5,585 and 11,806 slices respectively and for JPEG applications, 5,599 and 13,310 slices respectively. Finally, in our speed performance analysis, we have

shown that NoC-based system is superior to the bus-based system. This is because the NoC architecture encourages a higher level of pipelining compared to bus-based system.

The next chapter concludes this thesis by providing a summary of the research contributions. In addition, a discussion of some of the work my colleagues are currently working and future work that remains to be done in this area is also presented.

Chapter 7

Conclusions and Future Work

As the complexity of system on chip continues to grow at its present pace, new challenges will surface that require solutions through the development of new technology, design techniques and methodologies. Communication infrastructure has always been overlooked during the development of electronic systems but with the increase in complexity, communication infrastructure will play a crucial role in future development. It is evident that a new design methodology is required and the adoption of Networks-on-Chip is inevitable in the near future.

NoCs present a new communication architecture where the infrastructure allows for greater dimension of parallel processing and resources utilization compared to a bus-based system. NoC's greater flexibility and robustness allows for more opportunities for hardware customization for addressing various application needs. Significant amount of theoretical work supports the vast potential of NoC and more practical studies are needed to evaluate NoCs driven by real applications.

This thesis explored design and implementation of an NoC-based system for FPGAs, running two real world applications. We compared the results to a bus-based system running the same applications. After much exploration and literature review, we concluded that a mesh circuit-switched NoC architecture is most suited for FPGA implementation. In Chapter 4, the design and implementation of NoC components such as, router, crossbar, network adaptor and links were discussed in detail. In Chapter 5,

application mapping on an NoC-based system and bus-based system was presented. In Chapter 6, the results from the NoC-based system and bus-based system implementing real test applications were presented.

7.1 Summary of Research Contributions

The following contributions were made over the course of this research:

1. A preliminary case study was conducted in which the feasibility of applying a NoC-based approach for FPGA implementation was investigated.
2. We succeeded in creating and implementing a working a NoC platform in Handle-C, a high level language to facilitate rapid prototyping for meeting today's communication challenges.
    a. The simple mesh topology can significantly reduce network complexity while still providing reasonable area utilization, reduced data latency, and faster speed compared to a bus-based system.
    b. The implementation of circuit-switched as a switching protocol eliminates most of the traffic disruption (i.e. deadlock, livelock and starvation) in the network. Thus, it can better provide higher Quality of Service.
3. A case study was conducted that evaluated and compared the area utilization and speed performance of an NoC-based system and a bus-based system running real applications. The results from the prototypes provide more accurate performance estimation and prediction which gives this research more practical value.

7.2 Future Work

Through the progression of this research, many interesting topics continue to surface during the development of the NoC. Because of time constraints, these topics are out of the scope of this research but they can provide an excellent opportunity for future work to further the design space exploration of NoC. Follow-up research can use these benchmarks and NoC-based system that was developed, for implementing and evaluating different NoC architectures.

First, most router implementation in a mesh topology consists of ports connecting to its immediate horizontal and vertical neighbours. Although this is a typical router configuration, router can be further expand to include diagonally connections. This can further increase bandwidth for routing and also diagonal connection can reduce the number of hops need to be taken for longer routes. Few concerns are increasing physical wires will increase area cost, more wires consume more energy and the increase complexity of the router system for incorporate more routing ports.

Second, circuit-switched are best for applications where traffics are mostly static and predictable, while packet-switched works well with dynamic traffic. Therefore, a more sophisticate protocols can be developed by combining a mixture of both circuit-switched and packet-switched protocols that can be used for any types of traffic.

Finally, our proposed NoC system was developed using a high level language and could be further improved by optimizing hardware components. Latency, speed performance and noise immunity will be improved while requiring less area. The network should be tested thoroughly and the evaluation metrics should expand to include energy consumption by utilizing new tools.

Appendix A

Simulating a real CCD

This program is from Vahid and Givargis book on embedded system design [26]. The high-level model starts with the CCD module which simulated a real CCD on a digital camera by capturing an image. In this case, read it from a parameterized file that contained pixel data of an image along with the zero-bias error. The CCD model initializes our model just prior for image processing by simple reading the image directly from a file.

The next model is CCDPP which performs the zero-bias adjustment processing. This module also exports three procedures called *CcdppInitialize*, *CcdppCapture*, and *CcdppPopPixel*. The *CcdppInitialize* procedure performs any necessary initializations. The *CcdppCapture* procedure is called to actually capture an image. Note that this procedure calls on the *CcdCapture* and *CcdPopPixel* procedures of the CCD module to obtain an image. As it is obtaining the image pixels, it also performs the zero-bias adjustment. The *CcdppPopPixel* procedure is called to get the pixels out of the CCDPP

The next module is called CODEC which models the forward DCT encoding. The CODEC module exports the procedures *CodecInitialize*, *CodecPushPixel*, *CodecPopPixel*, and *CodecDoFdct*. The *CodecInitialize* procedure resets an index that is used by the push and pop procedures for traversing two buffers. The *CodecPushPixel* is called 64 times to fill an input buffer, called *ibuffer*, which holds the original block of 8x8 pixels that is to be encoded. The *CodecDoFdct* is called to actually perform the transform. Therefore, to encode a block of 8x8 pixels, we call *CodecPushPixel* 64 times,

and *CodecDoFdct* once followed by 64 calls to *CodecPopPixel.* This module simply implements the FDCT for the given equation:

$$C(h) = \text{if } (h==0) \text{ then } 1/\text{sqrt}(2) \text{ else } 1.0$$

$$F(u,v) = \tfrac{1}{4} * C(u) * C(v) \sum_{x=0..7} \sum_{y=0..7} D_{xy} * \cos(\pi(2x + 1)u/16) * \cos(\pi(2y + 1)v/16)$$

For performance purposes, the cosine function is recomputed to 64 possibilities and are store in a table because the only variables in the cosine argument expression are the integers x and u and each of these variables can take one of 8 values. from 0 to 7

The last module that will complete the implementation of the digital camera is the heart of the system, called CNTRL, short for controller. This module exports three procedures name *CntrlCompressImage* and *CntrlSendImage.* The *CntrlCompressImage* procedure uses the other modules that were use so far, namely the CCDPP and the CODEC to capture and perform FDCT and quantization on an image. Part of what this procedure has to do is to break the image into windows, or what can be referred to as blocks of 8x8 pixels. Once a block is FDCT encoded, it is quantized and stored in memory. The *CntrlSendImage* procedure simply transmits the encoded image, serially, using the UART module.

A.1 CCD Module in C

```
#include <stdio.h>
#define SZ_ROW      64
#define SZ_COL      (64 + 2)
static FILE *imageFileHandle;
static char buffer[SZ_ROW][SZ_COL];
static unsigned rowIndex, colIndex;

void Ccdinitialize (const char *imageFileName) {
        imageFileHandle = fopen(imageFileName, "r");
        rowindex = -1;
        colIndex = -1;
}

void CcdCapture (void) {
        int pixel;
        rewind(imageFileHandle);
        for (rowIndex=0; rowIndex<SZ_ROW; rowIndex++) {
                for (colIndex=0; colIndex<SZ_COL; colIndex++) {
                        if (fscanf(imageFileHandle, "%i", &pixel) ==1) {
                                buffer[rowIndex][colindex] = (char)pixel;
                        }
                }
        }
        rowIndex=0;
        colIndex=0;
}

char CcdpopPixel (void) {
        char pixel;
        pixel = buffer[rowIndex][colIndex];
        if (++colIndex==SZ_CP) {
                colIndex=0;
                if (++rowIndex==SZ_ROW) {
                        colIndex=-1;
                        rowIndex=-1;
                }
        }
        return pixel;
}
```

A.2 CCDPP |Module in C

```
#define SZ_ROW      64
#define SZ_COL      64
static char buffer[SZ_ROW][SZ_COL];
static unsigned rowIndex, colIndex;

void Ccdppinitialize() {
        rowIndex=-1;
        colIndex=-1;
}

void CcdppCature(void){
        char bias:
        CcdCapture()
        for (rowIndex=0; rowIndex<SZ_COL; rowIndex++) {
                for (colIndex=0;colIndex<SZ_COL; colIndex++) {
                        buffer[rowIndex][colIndex]=CcdPopPixel();
                }
                bias=(CcdPopPixel() + CcdpopPixel()) /2;
                for (colIndex=0; colindex<SZ_COL; colIndex++) {
                        buffer[rowIndex][colIndex]-=bias;
                }
        }
        rowIndex=0;
        colIndex=0;
}

char CcdppPopPixel(void) {
        char pixel;
        pixel=buffer[rowIndex][colIndex];
        if (++colIndex==SZ_COL) {
                colIndex=0;
                if( ++rowIndex==SZ_ROW) {
                        colIndex=-1;
                        rowIndex=-1;
                }
        }
        return pixel;
}
```

A.3 UART Module in C

```
#include <stdio.h>
static FILE *outputFileHandle;

void UartInitialize(const char *outputFileName) {
        outputFileHandle=fopen(outputFileName, "w");
}

void UartSend(char d) {
        fprintf(outputFilehandle, "%i/n", (int)d);
}
```

A.4 CODEC Module in C

```
static const short COS_TABLE[8][8]= {
        { 32768. 32138, 30273, 27245, 23170, 18204, 12539, 6392},
        { 32768. 27245, 12539, -6392, -23170, -32138, -30273, -18204},
        { 32768. 18204, -12539, -32138, -23170, 6392, 30273, 27245},
        { 32768, 6392, -30273. -18204, 23170, 27245, -12539, -32138},
        { 32768, -6392. -30273, 18204, 23170, -27245. -12539, 32138},
        { 32768, -18204, -12539. 32138, -23170, -6392, 30273, -27245},
        { 32768, -27245. 12539, 6392. -23170, 32138, -30273. 18204},
        { 32768, -32138, -30273, 27245. 23170, -18204, 12539, -6392},
};

static short ONE_OVER_SQRT_TWO=23170, ibuffer[8][8]. obuffer[8][8], idx;
static double COS(int xy. int uv) { return COS_TABLE[xy][uv]/32768.0;)
static double C(int h) { return h? 1.0:ONE_OVER_SQRT_TWO/32768.0;)

static int FDCT(int u, int v, short img[8][8]) {
        double s[8], r=0; int x;
        for (x=0; x<8; x++) {
                s[x] = img[x][0]*COS(0,v) + img[x][1]*COS(1,v) + img[x][2]*COS(2,v)
                +  img[x][3]*COS(3,v) + img[x][4]*COS(4,v) + img[x][5]*COS(5,v)
                + img[x][6]*COS(6,v) + img[x][7]*COS(7,v);
        }
        for (x=0; x<8; x++) r+=s[x]*COS(x,u);
        return (short)(r*.25*C(u)*C(v));
}

void CodecInitialize (void) {idx=0}

void CodexpushPixel (short p) {
        if(idx==64) idx=0;
        ibuffer[idx/8][idx%8]=p; idx++;
}

short CodecPopPixel (void) {
        short p;
        if( idx==64 ) idx=0;
        p=obuffer[idx/8][idx%8]; idx++;
        return p;
}

void CodecDoFdct(void) {
```

```
        int x, y;
        for(x=0; x<8; x++) {
                for(y=0; y<8; y++) obuffer[x][y] ; FDCT(x,y. ibuffer);
        }
        idx=0;
}
```

A.5 CNTRL module in C

```
#define SZ_ROW      64
#define SZ_COL      64
#define NUM_ROW_BLOCKS (SZ_ROW/8)
#define NUM_COL_BLOCKS (SZ_COL/8)
static short buffer[SZ_ROW][SZ_COL}, i, j, k, l, temp;
void CntrlInitialize (void) {}
void CntrlCaptureImage (void) {
        CcdppCapture ();
        for(i=0; i<SZ_ROW; i++) {
                for(j=0; j<SZ_COL; j++) {
                        buffer[i][j]=CcdppPopPixel();
                }
        }
}
void CntrlCompressImage (void) {
        for (i=0; i<NUM_ROW_BLOCKS; i++) {
                for (j=0; j<NUM_COL_BLOCKS; j++) {
                        for (k=0; k<8; k++) {
                                for (l=0; l<8; l++) {
                                        CodecPushPixel((char)buffer[i*8+k][j*8+l]);
                        CodecDoFdct();
                        for (k=0; k<8; k++) {
                                for (l=0; l<8; l++) {
                                        buffer[i*8+k][j*8+1]=CodecPopPixel ();
                                        buffer[i*8+k][j*8+1] >>=6;
                                }
                        }
                }
        }
}
void CntrlSendImage(void) {
        for(i=0; i<SZ_ROW; i++) {
                for(j=0; j<SZ_COL; j++) {
                        temp=buffer[i][j];
                        UartSend(((char*)&temp)[0]);
                        uartSend(((char*)&temp)[1]);
                }
        }
}
```

A.6 MAIN Module in C

```
int main (int argc, char * argv[]) {
        char *uarOutputFileName=argc > 1? argv[1]:"uart_out.txt";
        /* initialize the modules*/
        UartInitialize(uartOutputFileName);
        CcdInitialize(imageFileName);
        CcdppInitialize();
        CodecInitialize();
        CntrlInitialize();
        /* simulate functionality*/
        Cntrl CaptureImage();
        CntrlCompressImage();
        CntrlSendImage();
}
```

References

[1] C. Hilton, B. Nelson, "PNoC: a flexible circuit-switched NoC for FPGA-based systems," In Proc IEE Computer. Digit Technology, Vol. 153, No. 3, May 2006.

[2] U. Y Ogras, R. Marculescu, H. G. Lee, P Choudhary, D. Marculescu, M. Kaufman, P. Nelson, "Challenges and Promising Results in NoC Prototyping using FPGA", In IEEE Computer Society, 2007.

[3] I. D. L. Anderson, "A Cad Tool for Design Space Exploration of Embedded CPU Cores for FPGAs," M.S. thesis, University of Windsor, 2007

[4] Siguenza-Tortosa, "Proteo: The Development of a Practical Network-on-Chip," Ph.D. dissertation, Tampere University of Technology, Nov. 30, 2005.

[5] J. Duato, S. Yalmanchili and L. Ni. "Interconnection networks: An Engineering Approach," Morgan Kaufmann, San Francisco, CA, 2003.

[6] S. P Sample and M. R. Butts, "Optimized Emulation and Prototyping Architecture," U.S. Patent, US 6,625,793 B2, Sept. 23, 2003.

[7] D. Wiklund, D. Liu, "SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems," In Proc. International Parallel and Distributed Processing Symposium (IPDPS'03), p. 78a, 2003.

[8] G. D. Micheli, L. Benini, "Technology and Tools Networks on Chips," Morgan Kaufmann, 2006.

[9] P. T. Wolkotte, G J. M. Smit, G. K. Rauwerda, L. T. Smit, "An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip," In Proc IEEE 19thParallel and Distriubuted Processing Symposium, 2005.

[10] W. Dally and B. Towles, "Route Packets, Not Wires: OnChip Interconnection Networks," In Proceedings of Design Automation Conference. DAC 2001, pp. 684-689, 18-22 June 2001.

[11] C. Bobda, M. Majer. D. Koch, A. Ahmadinia, and J. Teich, "A Dynamic NoC Approach for Communication in Reconfigurable Devices," In Proc. Field Programmable Logic & Applications Conference, 2005.

[12] T. A. Bartic, J.-Y Mignolet, V Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Highly Scalable Network on Chip for Reconfigurable Systems," In Proc. International Symposium on System-on-Chip, pp. 79–82, November 2003.

[13] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," ACM Computing Surveys, Vol.38. Article 1, March 2006.

[14] EECS instructional and Electronics Groups Homepage at University of California Berkeley. http://inst.eecs.berkeley.edu/. January 2007.

[15] Altera Corporation Website. October 2008, http://www.altera.com/.

[16] Xilinx Incorporated Website. October 2008, http://www.xilinx.com/.

[17] Xilinx Incorporated. "Spartan-3 FPGA Family: Complete Data Sheet," January 2005.

[18] Nios II Website. January 2007, http://www.altera.com/products/ip/processors/nios2/ni2/index/html.

[19] Altera Corporation. "Quartus II Version 5.0 Handbook," Version 5.0.0, May 2005.

[20] Altera Corporation. SOPC builder. January 2007, http://www.altera.com/producst/software/ products/sopc/sop-index.html.

[21] Xilinx Incorporated. "Microblaze Processor Reference Guide," UG081 (v5.1) April 2, 2005.

[22] Celoxica Limited, "Platform Developer's Kit MicroBlaze Manual," September 2007.

[23] T. A. Bartic, D.Verkest, S.Vernalde, T. Marescaux, R. Lauwereins, "Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs," In Proc. Field Programmable Logics, p795-805, 2002.

[24] D. Wiklund and D. Liu, "Design of a system-on-chip switched network and its design support," In Proc of the Design Automation Conference, 2001.

[25] B. Sethuraman, P Bhattacharya, J. Khan, R. Vemuri, "LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip," ACM GLSVLSI, April 17, 2005.

[26] F Vahid, T. Givargis, "Embedded System Design a Unified Hardware/Software Introduction," Hoboken, NJ: Wiley, 2002.

[27] P.P Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, G. D. Micheli "Design, Synthesis, and Test of Networks on Chips," IEEE Design and Test of Computers, Vol.22, pp. 404-413, September, 2005.

[28] EETimes Website, "Sharp, Celoxica leverage C variants to speed design," December 08, 2000, http://EETimes.com.

[29] C. Wayne Brown and B. J. Shepherd. "Graphics File Formats – Reference and Guide," Connecticut: Manning Publications Company, 1995.

[30] L. Benini and D. Bertozzi, *"Network-on-chip Architectures and Design Methods,"* In Proc. IEE Computer Digit Technology, Vol. 152, pp. 261–272, 2005.

[31] K. C. Chang, J. S. Shen, T. F Chen, "A Low-Power Crossroad Switch Architecture and Its Core Placement for Network-On-Chip," ACM ISLPED, August8-10, 2005.

[32] J. C. P Ortiz, "Design of Components for a NoC-Based MPSoC Platform," Eindhoven University of Technology, June 30, 2005.

[33] C. R. Hilton, "A Flexible Circuit-Switched Communication Network for FPGA-Based SoC Design," Brigham Young University, August, 2005.

[34] N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, "A Virtual Channel Router for On-Chip Networks," In Proc. IEEE SoC Conference, September, 2004.

[35] J. D. Owens, W J. Dally, R. Ho, D. N. Jayasimha, S. W Keckler, L. Peh, "Research Challenges for On-Chip Interconnection Networks," In Proc. IEEE Computer Society, 2007.

[36] K. Ravindran, N. Satish, Y Jin, K. Keutzer, "An FPGA-based Soft Multiprocessor system for IPV4 Packet Forward," In Proc. Field Programmable Logic, pp 487-492, August, 2005.

[37] D. Wiklund, D. Liu, "Switched Interconnect for System-on-a-Chip Designs," In Proc. IP Europe Conference, 2000.

[38] I. Saastamoinen, "Interconnect IP Node for Future System-on-Chip Designs," IEEE International Workshop on Electronic Design, Test and Applications, 2002.

[39] J. Hu, R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures," In Proc. DATE Conference, March, 2003.

[40] J. Liu, L Zheng, H. Tenhunen, "A Circuit-Switched Network Architecture for Network-on-Chip," In Proc. IEEE SOC Conference, pp. 12-15 Sept., 2004.

[41] J. Hu, R. Marculescu, "Energy-aware mapping for Tile-based NoC Architectures under performance Constraints," In Proc. Asia and South pacific Design Automation Conference, pp 223-233. January, 2003.

[42] L. Benini and G. D. Micheli, "Networks on Chips: A new Soc Paradigm," IEEE Computer. 35(1):pp. 70-80, January. 2002.

[43] P. H. Pham, Y. Kumar, C. Kim, "High Performance and Area-Efficient Circuit-Switched Network on Chip Design," In Proc. IEEE International Conference on Computer and Information Technology, 2006.

[44] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, "Performance evaluation and Design Trade-offs for Network-on-Chip Interconnection Architectures," IEEE Transaction onComputers, Vol. 54(8), pp. 1025-1040. Aug., 2005.

[45] D. Kim, K. Lee, S. Lee, H. J. Yoo, "A Reconfigurable Crossbar Switch with Adaptive Bandwidth Control for Networks-on-Chip," In Proc. ISACS. Vol.3, pp. 2369-2372. 2005.

[46] A. Ahmadinia, et al., "A Practical Approach for Circuit Routing on Dynamic Reconfigurable Devices," In Proc. RSP. pp. 84-90, 2005.

[47] S. Sathe, D. Wiklund, D. Liu, "Design of a Switching Node (Router) for on-chip Networks," In Proc. ASICON 2003 Conference, March, 2003.

[48] P. Kundu, L. Peh, "On-Chip Interconnects for Multicores," IEEE Computer Society. 2007.

[49] M. Saldana, L. Shannon, P. Chow. "The Routability of Multiprocessor Network Topologies in FPGAs," In Proc. ACM/SIGDA, 2006.

[50] W Dally, "Virtual-channel flow control," IEEE Transactions on Parallel and Distributed systems, Vol. 3, pp. 194-205, March, 1992.

[51] T. S. T. Mak, P Sedcole, P Y. K. Cheung, W. Luk, "On-FPGA Communication Architectures and Design Factors," FPL, 2006.

[52] V D. Ngo, H. N. Nguyen, H. W Choi, "Analyzing the Performance of Mesh and Fat-Tree Topologies for Network on Chip Design," EUC LNCS, pp. 300-316, 2005.

[53] Celoxica Limited, "Using Handel-C with DK," 2006.

[54] Celoxica Limited, "Platform Developer's Kit PAL Manual," 2006.

[55] Celoxica Limited, "Platform Developer's Kit Cosim Manager Manual," 2006.

[56] EEC Electrical and Computer Engineering at University of Toronto. October, 2008, http://www.eecg.toronto.edu/~lemieux/sega/ispd97/.

[57] Altera Corporation. "Avalon Bus Specification Reference Manual," Version 2.3, July 2003.

[58] Celoxica Limited, "DK4 Handel-C Language Reference Manual," 2005.

[59] Celoxica Limited, "DK Libraries Manual," Version 4.0 SP1, 2005.

[60] Celoxica Limited, "DK deisgn Suite user guide," Version 4.0 SP1, 2005.

[61] Celoxica Limited, "Platform Developer's Kit RC host library and FTU 3 Manual" 2005.

VITA AUCTORIS

Thuan Le was born in Hong Kong on April 25, 1983. He received his B.A.Sc. degree in electrical engineering in 2006 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.