12-2010

Data routing in multicore processors using dimension increment method

Arpita H. Kadakia
University of Nevada, Las Vegas

Follow this and additional works at: http://digitalscholarship.unlv.edu/thesesdissertations

Part of the Computer and Systems Architecture Commons, Hardware Systems Commons, Signal Processing Commons, and the Systems and Communications Commons

Repository Citation
http://digitalscholarship.unlv.edu/thesesdissertations/665

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
DATA ROUTING IN MULTICORE PROCESSORS USING DIMENSION INCREMENT METHOD

by

Arpita H Kadakia
Bachelor of Electronic Engineering
Vidyalankar Institute of Technology, India
2007

Diploma in Electronics and Communication
Bombay Institute of Technology, India
2004

A thesis submitted in partial fulfillment
of the requirement for the

Masters of Science Degree in Electrical Engineering
Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2010
THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Arpita H. Kaddakia

Entitled

Data Routing in Multicore Processors Using Dimension Increment Method

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

Henry Selvaraj, Committee Chair
Shahram Latifi, Committee Member
Emma E. Regentova, Committee Member
Lawrence L. Larmore, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

December 2010
ABSTRACT

Data Routing In Multicore Processors Using Dimension Increment Method

by

Arpita H Kadakia

Dr. Henry Selvaraj, Examination Committee Chair
Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

A Deadlock-free routing algorithm can be generated for arbitrary interconnection network using the concept of virtual channels but the virtual channels will lead to more complex algorithms and more demands of NOC resource.

In this thesis, we study a Torus topology for NOC application, design its structure and propose a routing algorithm exploiting the characteristics of NOC. We have choose a typical 16 (4 by 4) routers Torus and propose the corresponding route algorithm. In our algorithm, all the channels are assigned 4 different dimensions (n0,n1,n2 & n3). By following the dimension increment method, we break the dependent route circles, and avoid dead lock and live-lock and avoid the overhead of virtual channels.

Xilinx offers two soft core processors, namely Picoblaze and Microblaze. The Picoblaze processor is 8-bit configurable processor core. These soft processor cores offer designers tremendous flexibility during the design process, allowing the designers to configure the processor to meet the needs of their systems (e.g., adding custom instructions or including/excluding particular data path coprocessors) and to quickly integrate the processor within any FPGA. Unlike single chip Microprocessor/FPGA systems using hard-core processors, soft processor cores allow designers to incorporate varying numbers of processors within a single FPGA design depending on an application’s needs.
Soft processor cores implemented using FPGAs typically have higher power consumption and decreased performance compared with hard-core processors.

Key features of the Picoblaze processor, as well as other soft processor cores, include the user configurable options that allow a designer to tailor the processor’s functionality to their specific design.

The proposed design implements sixteen instances of a soft processor, Picoblaze, connected in a torus topology. Data is passed from one processor to another employing a routing algorithm which is based on dimension increment method. Thus we design an NOC with multiple microcontrollers and related logic, synthesize the process and test its performance in a simulation environment.
# TABLE OF CONTENT

ABSTRACT ........................................................................................................ iii

LIST OF FIGURES ......................................................................................... vii

ACKNOWLEDGEMENT .............................................................................. viii

CHAPTER 1 INTRODUCTION ....................................................................... 1

CHAPTER 2 NETWORK TOPOLOGY ................................................................. 4
  2.1 Benefits of adopting an NOC ................................................................. 5
  2.2 Topology ............................................................................................. 6
    2.2.1 Mesh Topology ............................................................................. 6
    2.2.2 Torus Topology ........................................................................... 8
  2.3 Flow Data ............................................................................................ 9
  2.4 Virtual Channel ................................................................................... 10
  2.5 Deadlock and Liveloak .............................................................. 12
    2.5.1 Prevention ............................................................................... 13
    2.5.2 Recovery .................................................................................. 13
    2.5.3 Avoidance ............................................................................... 14
  2.6 Deadlock Avoidance ........................................................................... 14
  2.7 Liveloak ............................................................................................. 18

CHAPTER 3 ARCHITECTURE AND WORKING OF PROPOSED NETWORK..... 19
  3.1 Justification for processor ............................................................... 19
  3.2 Use of Microprocessor within an FPGA ........................................... 21
  3.3 Picoblaze Microcontroller Features ............................................... 28
  3.4 Internal architecture of Picoblaze ..................................................... 30
    3.4.1 16 General purpose registers ................................................. 30
    3.4.2 ALU ......................................................................................... 30
    3.4.3 Reset ....................................................................................... 31
    3.4.4 Flags and program flow control .......................................... 31
    3.4.5 Input/output .......................................................................... 32
    3.4.6 Interrupt ............................................................................... 32
    3.4.7 Scratch Pad Memory ........................................................... 33
  3.5 Torus Structure .................................................................................. 37
  3.6 Proposed Torus Structure ............................................................... 38
  3.7 Routing Algorithm .......................................................................... 40
  3.8 Memory ........................................................................................... 42
  3.9 Torus Interconnection and placement of Routers ................................. 47
  3.10 Interconnection and Interaction between memory Picoblaze and routers...... 49

CHAPTER 4 EXPERIMENTAL RESULTS ......................................................... 52
  4.1 Simulation ....................................................................................... 53
  4.2 Synthesis ......................................................................................... 59
CHAPTER 5 CONCLUSION AND RECOMMENDATIONS .............................. 67
5.1 Recommendations ................................................................... 68

BIBLIOGRAPHY ........................................................................... 69
VITA ............................................................................................... 71
LIST OF FIGURES

Figure 2.1 Mesh Topology ................................................................. 7
Figure 2.2 2D Torus Topology .......................................................... 9
Figure 2.3 Deadlock Configuration ................................................... 15
Figure 2.4 Stages of transverseing packet ....................................... 16
Figure 2.5 Dependency graph and Wait for and Hold graph .......... 17
Figure 3.1 KCPSM3 Block Diagram .................................................. 24
Figure 3.2 Macro in Isolation with XC Virtex 4 Device ................. 27
Figure 3.3 Assembler Files ............................................................. 29
Figure 3.4 Internal architecture of Picoblaze ................................. 30
Figure 3.5 Picoblaze Comparison .................................................... 36
Figure 3.6 Cores and Router Interconnects .................................... 37
Figure 3.7 4 by 4 Torus ................................................................. 39
Figure 3.8 Reconstructed Torus Structure ....................................... 41
Figure 3.9 Memory ....................................................................... 43
Figure 3.10 Source Selector ............................................................ 44
Figure 3.11 Destination Selector ..................................................... 45
Figure 3.12 Data Selector .............................................................. 46
Figure 3.13 Enable Selector ........................................................... 47
Figure 3.14 Router ID and their placements ................................. 48
Figure 3.15 Picoblaze and Router Connection Block Diagram .. 51
Figure 4.1 Simulation Results showing input and output of Picobalze_0000 ................................................................. 54
Figure 4.2 Output of router_0010 ................................................... 55
Figure 4.3 Channel28_down .......................................................... 56
Figure 4.4 Router_1010 ............................................................... 57
Figure 4.5 Picoblaze_1010 ............................................................ 58
Figure 4.6 Synthesis with Spartan 3E ............................................ 59
Figure 4.7 Synthesis with Virtex 4 .................................................. 60
Figure 4.8 Synthesis Summary Report ........................................... 61
Figure 4.9 Xilinx Clocking Wizard ................................................ 62
Figure 4.10 Place and Route ............................................................ 63
Figure 4.11 Device Schematic ......................................................... 64
Figure 4.12 Package Schematic ...................................................... 65
Figure 4.13 8 Leds on Virtex -4 ....................................................... 65
ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. Henry Selvaraj who served as my Advisor, guided me through the entire course of study. This thesis would have been beyond my capability without his countless help and cooperation. It has been a great experience working with him as a research student.

I am grateful to the Department of Electrical and Computer Engineering for the financial assistantship and other support they provided me during my studies. I am also thankful to Dr. Shahram Latifi, Dr. Emma E. Regentova and Lawrence L. Larmore for kindly accepting to serve in my graduate committee.

I am thankful to God for giving me such a great opportunity and my parents, for their understanding, love and affection throughout my life. I would like to give special thanks to my sister Saloni, my brother in law Hemal and my cute nieces for their immense support and help in this foreign land. Finally I would like to thank my fiancé Ankoor for his motivation and encouragement when it was most needed.
CHAPTER 1
INTRODUCTION

Companies like Sun Microsystems Inc, IBM, Compaq Computer Corp, Hewlett-Packard Co. and others will start to roll out high-end servers that take advantage of chip multiprocessing (CMP), a step forward from current systems that load up boxes with multiple discrete chip modules. Two processors in a single module outperform multiple discrete processors by 50% or more. By putting two CPUs on a single piece of silicon, engineers can take advantage of shorter distances and faster bus speeds when shuttling data between the two CPU cores [1].

Performance of CMPs can potentially scale with the number of the processors or cores on the die without re-design of individual processors. Therefore, the same processor design with small modifications can be used across multiple generations of product, amortizing the cost of design and verification. Also, scaling CMP performance does not require an increase in energy dissipation per instruction; on the contrary, CMP performance can be increased simultaneously with a reduction in energy spent per instruction.

The switch to chip multiprocessors also helps to reduce the effect of wire delays, which is growing relative to gate delay. Each processor in a CMP is small relative to the total chip area, and wire length within a processor is short compared to the die size. Interprocessor communication still requires long global wires; however, the latency of interprocessor communication is less critical for performance in a multi-processor system than latency between units within a single processor [2]. Also, these long wires can be pipelined and thus don’t affect clock cycle time and performance of an individual
processor in a CMP. CMPs have evolved largely due to the increased power consumption in nano scale technologies which have forced the designers to seek alternative measures instead of device scaling to improve performance. Increasing parallelism with multiple cores is an effective strategy. However, the power dissipation challenges do not disappear in the CMP regime. In fact, the power optimization problem in CMPs is quite complex as these architectures include multiple heterogeneous processing cores. Even though there exists a large body of work on power optimization in uni-processor architectures, there is still little understanding of the power performance challenges on CMPs.

Single-core Microprocessor approach is beginning to slow due to the following key factors, increasing power consumption (>100 W), increasing heat dissipation, diminishing performance gains from ILP & TLP. Whereas in multicore multiple smaller and energy efficient processing cores are integrated onto a single chip, improved overall performance through concurrent processing and the latencies associated with chip-to-chip communication disappear, shared data structures are much less of a problem.

Torus architectures have gained increasing importance in the design of distributed memory multiprocessor systems in recent years. This is because of their regularity, modularity, fault-tolerance, scalability, ease of construction, and other attractive interconnection properties. The architecture enables efficient communication between nodes due to its high bisection density, possessing a very high potential for parallel execution of programs.

1.1. Thesis Overview

Chapter 1 introduces the concept of Chip Multiprocessing. In Chapter 2, a detail review of Mesh and Torus topology along with Flow control, Virtual Channels, Deadlock
and Livelock is provided. Chapter 3 explains the Architecture of the proposed CMP. In
Chapter 4 the main idea of this thesis is presented- Verifying if data has reached from the
source Microprocessor to the destination Microprocessor following a novel routing logic.
The proposed architecture is tested using a test bench and the simulation waveforms are
presented along with synthesis results. Chapter 5 gives the conclusion and future
recommendations.
Chip multiprocessors - also called multi-core Microprocessors or CMPs for short - are now the only way to build high-performance Microprocessors, for a variety of reasons. Large uniprocessors are no longer scaling in performance, because it is only possible to extract a limited amount of parallelism from a typical instruction stream using conventional superscalar instruction issue techniques.

CMPs fill up a processor die with multiple, relatively simpler processor cores instead of just one huge core. The exact size of a CMPs cores can vary from very simple pipelines to moderately complex superscalar processors, but once a core has been selected the CMPs performance can easily scale across silicon process generations simply by stamping down more copies of the hard-to-design, high-speed processor core in each successive chip generation. In addition, parallel code execution, obtained by spreading multiple threads of execution across the various cores, can achieve significantly higher performance than would be possible using only a single core. While parallel threads are already common in many useful workloads, there are still important workloads that are hard to divide into parallel threads. The low inter-processor communication latency between the cores in a CMP helps make a much wider range of applications viable candidates for parallel execution than was possible with conventional, multi-chip multiprocessors; nevertheless, limited parallelism in key applications is the main factor limiting acceptance of CMPs in some types of systems [3].

Multiprocessor architectures and platforms have been introduced to extend the applicability of Moore’s law. They depend on concurrency and synchronization in both
software and hardware to enhance the design productivity and system performance. These platforms will also have to incorporate highly scalable, reusable, predictable, cost- and energy-efficient architectures. With the rapidly approaching billion transistors era, some of the main problems in deep sub-micron technologies which are characterized by gate lengths in the range of 60-90 nm, will arise from non-scalable wire delays, errors in signal integrity and unsynchronized communications. These problems may be overcome by the use of Network on Chip (NOC) architecture [4].

On a billion transistors chip, it may not be possible to send a global signal across the chip within real-time bounds. If the SoC (System-on-Chip) is synchronized by a global clock signal, the circuit will be more prone to EMI (electromagnetic interference). The traditional system designs are usually based on critical paths and clock trees. These critical paths and clock trees contribute to an increased amount of power consumption. Therefore, SoCs are not power efficient. Besides, it is difficult to manage these clock trees due to clock skew problems. As compared to synchronous designs, asynchronous designs are modular and do not suffer from issues such as clock skew, higher power consumption and EMI. However, designing asynchronous systems is a more complex task as compared to designing a synchronous system. Designing a glitch free circuit and managing clock arrival time are complicated in the case of an asynchronous system.

2.1 Benefits of adopting an NOC

Traditionally, ICs have been designed with dedicated point-to-point connections, with one wire dedicated to each signal. For large designs, in particular, this has several limitations from a physical design viewpoint. The wires occupy much of the area of the chip, and in nanometer CMOS technology, interconnects dominate both performance
and dynamic power dissipation, as signal propagation in wires across the chip requires multiple clock cycles.

NoC links can reduce the complexity of designing wires for predictable speed, power, noise, reliability, etc., thanks to their regular, well controlled structure. From a system design viewpoint, with the advent of multi-core processor systems, a network is a natural architectural choice. An NoC can provide separation between computation and communication, support modularity and IP reuse via standard interfaces, handle synchronization issues, serve as a platform for system test, and, hence, increase engineering productivity [5].

2.2 Topology

From the communication perspective, there have been various topologies for NOC architecture. These include mesh, torus, ring, butterfly, octagon and irregular interconnection networks.

2.2.1 Mesh Topology

Mesh Network is a network where all the nodes are connected to each other and is a complete network. In a Mesh Network every node is connected to other nodes on the network through hops. Some are connected through single hops and some may be connected with more than one hop.

While the data is traveling on the Mesh Network it is automatically configured to reach the destination by taking the shortest route which means the least number of hops. Data travels by hopping from one node to another and then reaches the destination node in a Mesh Topology Network [6].
An example of a Mesh Network is the Mobile Adhoc Network or MANet. The entire Mesh Network is continuously connected. Being completely connected does not mean that Mesh Network is dependent on each and every node of the network. Even if one node fails in the Mesh Network the network finds an alternate route to transfer the data. It is called the self healing technology where it receives data one way or the other. The Mesh Network is based on a very sensible concept and has lesser chances of a network breakdown. There are so many possible combinations of routes and hops a data transfer can take that it will reach the destination one way or the other. It is highly unlikely that all the nodes in a single Mesh Network will break down at any given point of time.

Figure 2.1 Mesh Topology
Advantages of mesh topology is the arrangement of the network nodes is such that it is possible to transmit data from one node to many other nodes at the same time, no traffic problem as there are dedicated links, robust as failure of one link does not affect the entire system, security as data travels along a dedicated line, points to point links make fault identification easy.

Disadvantages of mesh topology are the arrangement wherein every network node is connected to every other node of the network, many of the connections serve no major purpose. This leads to the redundancy of many of the network connections the hardware is expansive as there is dedicated link for any two nodes and each device should have (n-1) I/O ports, there is mesh of wiring which can be difficult to manage and installation is complex as each node is connected to every node

2.2.2 Torus Topology

A 2D torus is achieved by enriching a 2D mesh with additional channels that connect the external nodes in each row and column. Every node is connected to four neighbors. Torus is degree regular. Due to the wraparound edges the bisection is 4N/k and diameter decreases to 2|k/2| hops. The Wraparound edges have to be long enough to span the length of k nodes. These long edges can increase propagation delay that can bring a negative impact on speed. In our proposed torus the structure remains the routers are place next to each other as per certain rule discussed in chapter 3. These routers are placed such that there is no Deadlock in the network and even the overhead of virtual channels is avoided.
2.3 Flow Data

Flow control determines how network resources, such as channel bandwidth, buffer capacity, and control state, are allocated to a packet traversing the network. The flow control may be buffered or buffer less. The Buffer less Flow Control has more latency and fewer throughputs than the Buffered Flow Control. The Buffered Flow Control can be further categorized into Credit Based Flow Control, ACK/NACK Flow Control, STALL/GO Flow Control, T-Error Flow Control, and Handshaking Signal based Flow Control.

In Credit Based Flow Control, an upstream node keeps count of data transfers, and thus the available free slots are termed as credits. Once the transmitted data packet is either consumed or further transmitted, a credit is sent back. In Handshaking Signal Based Flow Control, a VALID signal is sent whenever a sender transmits any flit. The receiver acknowledges by asserting a VALID signal after consuming the data flit.
In the ACK/NACK protocol a copy of a data flit is kept in a buffer until an ACK signal is received. On assertion of ACK, the flit is deleted from the buffer; instead if a NACK signal is asserted then the flit is scheduled for retransmission.

In the STALL/GO scheme, two wires are used for flow control between each pair of sender (producer) and receiver (consumer). When there is an empty buffer space, a GO signal is activated. Upon the unavailability of buffer space, a STALL signal is activated. None of the present NOC implementations have employed this flow control scheme.

The T-Error Flow Control scheme is very complex as compared to other flow control mechanisms. It aims at enhancing the performance at the cost of reliability. Real time systems operating in a noisy environment must avoid the use of this flow control mechanism. None of the present NOC implementations has employed this flow control scheme.

2.4 Virtual Channel

The design of a virtual channel (VC) is another important aspect of NOC. A virtual channel splits a single channel into two channels, virtually providing two paths for the packets to be routed. There can be two to eight virtual channels. The use of VCs reduces the network latency at the expense of area, power consumption, and production cost of the NOC implementation. However, there are various other added advantages offered by VCs. Network Deadlock/livelock: Since VCs provide more than one output path per channel there is a lesser probability that the network will suffer from a Deadlock; the network livelock probability is eliminated (these Deadlock and livelock are different from the architectural Deadlock and livelock, which are due to violations in inter-process communications).
Performance improvement: A packet/flit waiting to be transmitted from an input/output port of a router/switch will have to wait if that port of the router/switch is busy. However, VCs can provide another virtual path for the packets to be transmitted through that route, thereby improving the performance of the network. Supporting guaranteed traffic: A VC may be reserved for the higher priority traffic, thereby guaranteeing the low latency for high priority data flits.

Reduced wire cost: In today’s technology the wire costs are almost the same as that of the gates. It is likely that in the future the cost of wires will dominate. Thus, it is important to use the wires effectively, to reduce the cost of a system. A virtual channel provides an alternative path for data traffic, thus it uses the wires more effectively for data transmission. Therefore, we can reduce the wire width on a system (number of parallel wires for data transmission). For example, we may choose to use 32 bits instead of 64 bits. Therefore, the cost of the wires and the system will be reduced.

Virtual channels are the representation of the partitioned buffer queue inside a switch. Buffers can be placed in the input port of a switch and give us the input buffered switch, centrally within the switch which give us a centrally buffered switch and finally at both input and output ports of the switch which give us an input-output buffered switch. The packets traverse through the network using the same communication lines, and use the switches as intermediate stops until their destination. With the structure of virtual channels is provided to the incoming packets of a switch, an alternative path to select in case that a previous packet is blocked inside a buffer. This alternative path is selected through the flow control mechanism that is implemented in the switch, with the use
information that each packet carries in its header, so that can properly directed to its destination [7].

2.5 Deadlock and Livelock

Deadlock is a very common problem that happens in different communication levels, in our case in the interconnection network of a High Performance Computer. It is the situation that occurs when different processes wait one another to release specific resources. With that way there is cyclic dependency between these different processes for the same resources, creating like that a circular chain. Deadlock can be catastrophic and paralyze the network, is very important to eliminate any possibility that a Deadlock will occur. There are four necessary conditions for a Deadlock to occur, knows as Coffman conditions. These conditions are mutual exclusion, hold and wait condition, no pre-emption condition and circular wait condition.

Deadlock can be avoided if certain information about processes is available in advance of resource allocation. For every resource request, the system sees if granting the request will mean that the system will enter an unsafe state, meaning a state that could result in Deadlock. The system then only grants requests that will lead to safe states. In order for the system to be able to figure out whether the next state will be safe or unsafe, it must know in advance at any time the number and type of all resources in existence, available, and requested. One known algorithm that is used for Deadlock avoidance is the Banker's algorithm, which requires resource usage limit to be known in advance. However, for many systems it is impossible to know in advance what every process will request. This means that Deadlock avoidance is often impossible. A total ordering on a minimal set of resources within each dimension is required, if we would like to use these resources in full capacity. In contrary some resources along the dimension links have to
stay free so that can remain below the full capacity and avoid Deadlock. To allow full access to the network resources of the network, we have either to duplicate the physical links or duplicate the logical buffers associated with each link. This results respectively to physical channels or virtual channels. Routing algorithms based on this technique, called Duato’s protocol, can be defined that allow alternative paths provided by the topology, to be used for a given pair of source-destination nodes in addition to the escape resource set. One of those allowed paths must be selected, preferably the most efficient one. For Deadlock there are three known solution techniques, Prevention, Recovery and Avoidance. Each one of them refers to a different approach for the Deadlock [8].

2.5.1 Prevention

The system itself is built in such a way that there are no Deadlock. That means that the system makes sure, that at least one of the necessary for Deadlock conditions will never occur. This is done for example in circuit switching where the resources are granted before the transmission starts. It is very conservative approach and may lead to very low resource utilization [8].

2.5.2 Recovery

Deadlock recovery does not impose any restrictions to the routing mechanism, but rather allows Deadlock to occur. Deadlock recovery attends to give a solution to the problem after that has caused, forcing the agents that hold resources to release them, allowing with that way other agents to use those resources and break the Deadlock.

2.5.3 Avoidance

Deadlock avoidance is the technique where certain information about agents is available in advance of resource allocation. For every resource request, the system sees if
granting the request will mean that the system will enter an unsafe state, meaning a state that could result in Deadlock. The system then only grants request that will lead to safe states. In order for the system to be able to figure out whether the next state will be safe or unsafe, it must know in advance at any time the number and type of all resources in existence, available, and requested. One known algorithm that is used for Deadlock avoidance is the Banker's algorithm. However, for many systems it is impossible to know in advance what every process will request. This means that Deadlock avoidance is often impossible [8].

2.6 Deadlock Avoidance

A Deadlock is a situation where in two or more competing actions are waiting for the other to finish, and thus neither ever does. It is often seen in a paradox like 'the chicken or the egg'. In computer science, Deadlock refers to a specific condition when two or more processes are each waiting for each other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resources known as a software, or soft, lock [8].
Deadlock occurs in an interconnection network when a group of agents, usually packets are unable to make progress because they are waiting on one another to release resources, usually buffers on channels. If a sequence of waiting agents forms a cycle, as it shown in image 3, then the network is Deadlocked. This can have catastrophic sequences for the network. When some resources of the network are been occupied with Deadlocked packets other packets that coming block on these resources and cannot proceed to their destination.

To achieve the Deadlock Avoidance, the routing mechanism applied has to restrict the allowed paths for the packets that keep Deadlock free the global network state. An approach for the solution of this is to put an order on the resources that want to be accessed by the packets, in the minimum way for having network full access. Assigning
the resources partially or totally to the packets, so that cannot exist the possibility that a circular dependency will appear. With that way we are applying escape paths to the packets, no matter where they are inside the network, avoiding the probability that they will come in a Deadlock situation.

Critical resources on the Deadlock avoidance, in network level, are the connection lines and the buffers associated with them. There must be an order in the access of the resources by the packet, while these are travelling from their source to the destination.

![Figure 2.4 Stages of transverseing packet](image)

When a packet inserted in the network at the phase 0 is entering in a switch. Through the communication lines goes to the phase 1 where the next switch is, and continues until it reach its destination. While not exist recirculation of packets, once a packet have reserved an output channels from the first phase, it cannot request any other output channel from the same phase, thus there are no dependencies between the output channels of the same phase. Similarly a packet that has reserved an output channel on a given phase, cannot request for an output channel at a previous phase. With that way we only
have dependencies from this phase to the next phase. Sequence of that is that we don't have cyclic dependency between channels and we avoid Deadlock.

While using a flow control method, like store and forward or virtual cut through, the agents are packets and the resources are the packet buffers. At any given time each packet can only occupy one packet buffer. When a packet request for a new packet buffer, it should release the old packet buffer a short time later. In our case the resources will be the virtual channels that will replace the packet buffers as entities.

The lines (agents) and the virtual channels (resources) are related with “Wait for” and “Hold” relations. If a line holds a buffer, then that buffer is waiting from the line to be released. If that not happen, a Deadlock occur.

A representation of the relations between agents and resources can be done through the dependence graph and the wait-for graph. In both above image we can see how connections A and B occupying some resources while they are waiting for some others like A occupies channels u and v and waits for channel w which is occupied by the connection B. Similarly the connection B holds channels w and x and waits for channel u.

Figure 2.5 Dependency graph and Wait for and Hold graph

A representation of the relations between agents and resources can be done through the dependence graph and the wait-for graph. In both above image we can see how connections A and B occupying some resources while they are waiting for some others like A occupies channels u and v and waits for channel w which is occupied by the connection B. Similarly the connection B holds channels w and x and waits for channel u.
If we focus on the Hold relations that lead to the buffers u and w from the lines A and B in and we redraw these lines to the opposite direction. Here we can see, from the dotted arrows that appear a circulation between the resources. This circulation shows us that the configuration is Deadlocked.

In order to occur Deadlock, the lines have to acquire buffer resources and wait on others, with a way that creates a cycle in the wait for graph. This cycle is a necessary nut not sufficient condition for a Deadlock. If we can manage to eliminate the cycles from the resource dependence graph we can we eliminate the possibility of a circular dependence on the wait for graph and as a sequence we avoid to Deadlock the network [8].

2.7 Livelock

Livelock is a condition that occurs when two or more processes continually change their state in response to changes in the other processes. The result is that none of the processes will complete. An analogy is when two people meet in a hallway and each tries to step around the other but they end up swaying from side to side getting in each other's way as they try to get out of the way.

In our Proposed Torus Architecture we avoid the problems of virtual channel and by following the dimension increasing method avoid Deadlock and Livelock. Chapter 3 talks in detail about the router architecture and routing logic.
CHAPTER 3

ARCHITECTURE AND WORKING OF THE PROPOSED NETWORK

In computing, a processor is the unit that reads and executes program instructions. Processors were originally developed with only one core. The core is the part of the processor that actually performs the reading and executing of the instruction. Single-core processors can only process one instruction at a time. A multi-core processor is composed of two or more independent cores. It can be described as an integrated circuit which has two or more individual processors (called cores in this sense). In our proposed network sixteen instances of an 8-bit Microprocessor, Picoblaze [9] have been implemented. These sixteen instances are connected such that each can transfer data to any of the other fifteen processors using a routing logic. The proposed Architectural design can be divided into three main parts are memory, torus Structure and 16- Picoblaze Instances

3.1 Justification for Microprocessor?

There are literally dozens of 8-bit Microprocessor architectures and instruction sets. Modern FPGAs can efficiently implement practically any 8-bit Microprocessor, and available FPGA soft cores support popular instruction sets such as the PIC, 8051, AVR, 6502, 8080, and Z80 Microprocessors. The Picoblaze Microprocessor is specifically designed and optimized for the Xilinx Spartan-3 family and with support for Spartan-6, and Virtex-6 FPGA architectures. Its compact yet capable architecture consumes considerably less FPGA resources than comparable 8-bit Microprocessor architectures within an FPGA. Furthermore, the Picoblaze Microprocessor is provided as a free, source-level VHDL file with royalty-free re-use within Xilinx FPGAs. Some standalone Microprocessor variants have a notorious reputation for becoming obsolete. Because it is
delivered as VHDL source, the Picoblaze Microprocessor is immune to product obsolescence as the Microprocessor can be retargeted to future generations of Xilinx FPGAs, exploiting future cost reductions and feature enhancements. Furthermore, the Picoblaze Microprocessor is expandable and extendable.

Before the advent of the Picoblaze and Microblaze™ embedded processors, the Microprocessor resided externally to the FPGA, limiting the connectivity to other FPGA functions and restricting overall interface performance. By contrast, the Picoblaze Microprocessor is fully embedded in the FPGA with flexible, extensive on-chip connectivity to other FPGA resources. Signals remain within the FPGA, improving overall performance. The Picoblaze Microprocessor reduces system cost because it is a single-chip solution, integrated within the FPGA and sometimes only occupying leftover FPGA resources.

The Picoblaze Microprocessor is resource efficient. Consequently, complex applications are sometimes best distributed across multiple Picoblaze Microprocessors with each controller implementing a particular function, for example, keyboard and display control, or system management.

### 3.2 Use of Microprocessor within an FPGA

Microprocessors and FPGAs both successfully implement practically any digital logic function. However, each has unique advantages in cost, performance, and ease of use. Microprocessors are well suited to control applications, especially with widely changing requirements. The FPGA resources required to implement the Microprocessor are relatively constant. The same FPGA logic is re-used by the various Microprocessor
instructions, conserving resources. The program memory requirements grow with increasing complexity [10].

Programming control sequences or state machines in assembly code is often easier than creating similar structures in FPGA logic. Microprocessors are typically limited by performance. Each instruction executes sequentially. As an application increases in complexity, the number of instructions required to implement the application grows and system performance decreases accordingly. By contrast, performance in an FPGA is more flexible. For example, an algorithm can be implemented sequentially or completely in parallel, depending on the performance requirements. A completely parallel implementation is faster but consumes more FPGA resources.

A Microprocessor embedded within the FPGA provides the best of both worlds. The Microprocessor implements non-timing crucial complex control functions while timing critical or data path functions are best implemented using FPGA logic. For example, a Microprocessor cannot respond to events much faster than a few microseconds. The FPGA logic can respond to multiple, simultaneous events in just a few to tens of nanoseconds. Conversely, a Microprocessor is cost-effective and simple for performing format or protocol conversions.

Picoblaze is an efficient 8-bit Microprocessor architecture which can be synthesized in Spartan 3 FPGAs (also in Virtex II and Virtex-4). Picoblaze is similar to many Microprocessor architectures but it is specifically designed and optimized for Xilinx FPGAs. There are also larger Microprocessors that can be synthesized into FPGAs such as the 32-bit Microblaze Microprocessor. Picoblaze and Microblaze are typically referred to as soft processor cores since they are synthesized from an HDL and use the
programmable logic and routing resources of an FPGA for their implementation, as opposed to a dedicated processor hard core such as the PowerPC that is incorporated in some Virtex II and Virtex-4 FPGAs.

The Picoblaze was designed to be efficiently mapped to available FPGA resources including block memories and distributed memories as well. Although it can be used for processing of data it is used for applications requiring a complex but non time critical state machine. KCPSM3 is totally embedded into the device and requires no external support. Any logic can be connected to the module means that additional features can be provided to give extended flexibility.

The Picoblaze Microprocessor is optimized for efficiency and low deployment cost. It occupies just 96 FPGA slices, or only 12.5% of an XC3S50 FPGA and a miniscule 0.3% of an XC3S5000 FPGA. In typical implementations, a single FPGA block RAM stores up to 1024 program instructions, which are automatically loaded during FPGA configuration. Even with such resource efficiency, the Picoblaze Microprocessor performs a respectable 44 to 100 million instructions per second (MIPS) depending on the target FPGA family and speed grade.

The Picoblaze Microprocessor core is totally embedded within the target FPGA and requires no external resources. The Picoblaze Microprocessor is extremely flexible. The basic functionality is easily extended and enhanced by connecting additional FPGA logic to the Microprocessor's input and output ports [10].

The Picoblaze Microprocessor provides abundant, flexible I/O at much lower cost than off-the-shelf controllers. Similarly, the Picoblaze peripheral set can be customized to meet the specific features, function, and cost requirements of the target application.
The Picoblaze Microprocessor is delivered as synthesizable VHDL source code, the core is future-proof and can be migrated to future FPGA architectures, effectively eliminating product obsolescence fears. Being integrated within the FPGA, the Picoblaze Microprocessor reduces board space, design cost, and inventory.

The Picoblaze FPC is supported by a suite of development tools including an assembler, a graphical integrated development environment (IDE), a graphical instruction set simulator, and VHDL source code and simulation models. Similarly, the Picoblaze Microprocessor is also supported in the Xilinx System Generator development environment.

Picoblaze (see Figure 3.1) consists of two parts: 1) the processor core (KCPSM3 – which stands for Ken Chapman Programmable State Machine version 3 and 2) the program memory from which instructions are fetched and executed by the processor core. (Note that the program memory is referred to as an instruction ROM or program ROM in Picoblaze documentation since the processor core cannot write to the program memory.)

There are also two VHDL files that are used to construct the complete Picoblaze with program. The KCPSM3.vhd file is optimized for Spartan 3 by calling design primitives specific to Spartan 3 (also for Virtex II and Virtex-4) and, as a result, this VHDL file should not be modified by the user. The KCPSM3 require approximately 96 slices in a Spartan 3 [29]. The program memory, on the other hand, is a VHDL file specific to the user’s desired program to be executed by the Picoblaze core and is generated automatically by the assembler (KCPSM3.exe) from your assembly language program, name.psm. (Note that the prefix for the .psm file must be 8 characters or less.)
The program memory is implemented in a single Block RAM in the FPGA configured to function as a 1K×18-bit ROM. The program to be executed is typically initialized in the Block RAM during the download of the overall design (including Picoblaze and other user logic) and, as a result, is normally assembled prior to synthesis.

Figure 3.1 KCPSM3 Block Diagram(Source:KCPSM3 user manual)
## Figure 3.2 Signal Declarations

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_PORT[7:0]</td>
<td>Input</td>
<td>Input Data Port: Present valid input data on this port during an INPUT instruction. The data is captured on the rising edge of CLK.</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>Input</td>
<td>Interrupt Input: If the INTERRUPT_ENABLE flag is set by the application code, generate an INTERRUPT Event by asserting this input High for at least two CLK cycles. If the INTERRUPT_ENABLE flag is cleared, this input is ignored.</td>
</tr>
<tr>
<td>RESET</td>
<td>Input</td>
<td>Reset Input: To reset the PicoBlaze microcontroller and to generate a RESET Event, assert this input High for at least one CLK cycle. A Reset Event is automatically generated immediately following FPGA configuration.</td>
</tr>
<tr>
<td>CLK</td>
<td>Input</td>
<td>Clock Input: The frequency may range from DC to the maximum operating frequency reported by the Xilinx ISE® development software. All PicoBlaze synchronous elements are clocked from the rising clock edge. There are no clock duty-cycle requirements beyond the minimum pulse width requirements of the FPGA.</td>
</tr>
<tr>
<td>OUT_PORT[7:0]</td>
<td>Output</td>
<td>Output Data Port: Output data appears on this port for two CLK cycles during an OUTPUT instruction. Capture output data within the FPGA at the rising CLK edge when WRITE_STROBE is High.</td>
</tr>
<tr>
<td>PORT_ID[7:0]</td>
<td>Output</td>
<td>Port Address: The I/O port address appears on this port for two CLK cycles during an INPUT or OUTPUT instruction.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_STROBE</td>
<td>Output</td>
<td>Read Strobe: When asserted High, this signal indicates that input data on the IN_PORT[7:0] port was captured to the specified data register during an INPUT instruction. This signal is asserted on the second CLK cycle of the two-cycle INPUT instruction. This signal is typically used to acknowledge read operations from FIFOs.</td>
</tr>
<tr>
<td>WRITE_STROBE</td>
<td>Output</td>
<td>Write Strobe: When asserted High, this signal validates the output data on the OUT_PORT[7:0] port during an OUTPUT instruction. This signal is asserted on the second CLK cycle of the two-cycle OUTPUT instruction. Capture output data within the FPGA on the rising CLK edge when WRITE_STROBE is High.</td>
</tr>
<tr>
<td>INTERRUPT_ACK</td>
<td>Output</td>
<td>Interrupt Acknowledge: When asserted High, this signal acknowledges that an INTERRUPT Event occurred. This signal is asserted during the second CLK cycle of the two-cycle INTERRUPT Event. This signal is optionally used to clear the source of the INTERRUPT input.</td>
</tr>
</tbody>
</table>

(Source: KCPSM3 user manual)
KCPSM3 is supplied as a .vhd file and as a pre compiled sot macro which is handled by the place and route tools to merge with the logic of the design. We can Program the Block memory by written a .psm file as below:

```vhdl
CONSTANT source_inputport,01
CONSTANT destination_inputport,02 ;
CONSTANT data_inputport,03 ;
CONSTANT destination_outport,05
CONSTANT data_outport,06 ;
CONSTANT incomming_data,07;
CONSTANT outgoing_data,08;
;
start:
    STORE S6,source_inputport;
    INPUT s1,destination_inputport ;
    OUTPUT s1, destination_outport;
    INPUT s2,data_inputport ;
    OUTPUT s2, data_outport;
    INPUT s4,incomming_data;
    OUTPUT s4,outgoing_data;
```

Once the program has been simulated and verified, it can be assembled by `KCPSM3.exe`. This will produce a file named `in_test.vhd` which contains the VHDL for a Block RAM. We need a top level VHDL model to instantiate and interconnect the Picoblaze core (`kcpsm3.vhd`) and the program memory (`in_test.vhd`) instantiation which is
initialized with the machine language for the assembled program. Note that you will need to have the files *ROM_form.vhd* and *ROM_form.coe* in your directory where you are compiling the program.

Picoblaze and the KCPSM3 assembler support a total of 57 instructions, each belonging to one of 7 groups. There are approximately 21 types of instructions and, hence, the instruction set is fairly small and easy to learn. Instructions can be upper or lower case characters but are converted by the assembler to upper case. Comments begin with a semicolon, and everything on the line following the semicolon is ignored by the assembler. Note that the KCPSM3.vhd is compatible with the ModelSim simulator and can be simulated with the assembled program memory and any other user designed logic.

![Figure 3.2 Macro in Isolation with XC Virtex 4 Device](Source: KCPSM3 user manual)
3.3 Picoblaze Microprocessor Features

A 16 byte-wide general-purpose data registers, 1K instructions of programmable on-chip program store, automatically loaded during, byte-wide Arithmetic Logic Unit (ALU) with CARRY and ZERO indicator flags, 64-byte internal scratchpad RAM, 256 input and 256 output ports for easy expansion and enhancement, automatic 31-location CALL/RETURN stack, predictable performance, always two clock cycles per instruction, up to 200 MHz or 100 MIPS in a Virtex-II Pro FPGA, fast Interrupt response; worst-case 5 clock cycles, optimized for Xilinx Spartan-3 architecture—just 96 slices and 0.5 to 1 block RAM, support in Spartan-6, and Virtex-6 FPGA architectures, assembler, instruction-set simulator support.

Moreover PSM.EXE is provided to simplify the generation of programs. This assembler is written in C and compiled with Microsoft Visual Studio 6.0. It is a simple DOS program that can be run under a DOS window. Programs are best written with Notepad or WordPad type tools. The file is saved with a .psm file extension. KCPSM3 supports a program up to a length of 1024 instructions utilizing one block memory. Requirements for larger program space are typically addressed by using multiple KCPSM3 processors, each with an associated block memory to distribute the various system tasks. The KCPSM3 assembler actually reads four input files and generates 15 output files. Assembler error messages are displayed to help determine the reason for an error. The assembler also displays the line it was analyzing when it detected a problem. All output files are overwritten each time the assembler is executed. Filename.bin – This file is the binary code for the program memory in hex format and it is suitable for program debugging.
Filename.vhd – This file is a VHDL module for program memory generated by the assembler and suitable for synthesis and simulation. Filename.fmt – This file is the original program reformatted for easier reading. Looking at this file is also a good way to see that everything has been interpreted as intended. Filename.mcs – This file is the binary code for the program memory in Intel’s MCS-86 format. This file can be used to program an external memory if needed. Filename.log – This log file shows the assembler process performed and any error messages generated during the process.

Figure 3.3 Assembler Files (Source: KCPSM3 user manual)
3.4 Internal architecture of Picoblaze

3.4.1 16 General purpose registers.

There are 16 General purpose registers of 8-bits specified as ‘s0’ through to ‘sF’ which may be renamed in the assembler code. All operations are completely flexible about the use of registers with no registers reserved for special tasks or having any priority over any other register. There is no accumulator as any register can be adopted for this task.

3.4.2 ALU

The Arithmetic Logic Unit (ALU) provides many simple operations expected in an 8-bit processing unit. All operations are performed using an operand provided from any register (sX). The result is returned to the same register. For operations requiring a second operand, a second register can be specified (sY) or a constant 8-bit value (kk) can be supplied. The ability to specify any constant value with no additional penalty to program size or performance enhances the simple instruction set i.e. the ability to ‘ADD
1’ is the same as a dedicated INCREMENT operation. Addition (ADD) and Subtraction (SUB) have the option to include the carry flag as an input (ADDCY and SUBCY) for the support of arithmetic operations requiring more than 8-bits. LOAD, AND, OR and XOR bit-wise operators provide ability to manipulate and test values. Comprehensive SHIFT and ROTATE group. COMPARE and TEST instructions enable register contents to be tested without altering their contents and determine PARITY.

3.4.3 Reset

The RESET input forces the processor back into the initial state. The program will execute from address ‘000’ and Interrupts will be disabled. The status flags and CALL/RETURN stack will also be reset. Note that register contents are not affected.

3.4.4 Flags and program flow control

The results of ALU operations determine the status of the ZERO and CARRY flags. The ZERO flag is set whenever the ALU result has all bits reset (0016). The CARRY flag is set when there is an overflow from an arithmetic operation. It is also used to capture the bit moved out of a register during shift and rotate instructions. During a TEST instruction, the carry flag is used to indicate if the 8-bit temporary result has ODD PARITY. This status of the flags can be used to determine the execution sequence of the program using conditional and non-conditional program flow control instructions. JUMP commands are used to specify absolute addresses (aaa) within the program space. CALL and RETURN commands provide sub-routine facilities for commonly used sections of code. A CALL is made to an absolute address (aaa) and an internal program counter stack preserves the associated address required by the RETURN instruction. The stack supports up to 31 nested subroutine levels.
3.4.5 Input/Output

KCPSM3 effectively has 256 Input ports and 256 Output ports. The port being accessed is indicated by an 8-bit address value provided on the ‘PORT_ID’. The port address can be specified in the program as an absolute value (pp), or may be indirectly specified as the contents of any of the 16 registers (sY). During an ‘INPUT’ operation the value provided at the Input port is transferred into any of the 16 registers. An Input operation is indicated by a pulse being Output on the READ_STROBE. It is not always necessary to use this signal in the Input interface logic, but it can be useful to indicate that data has been acquired by the processor. During an ‘OUTPUT’, the contents of any of the 16 registers are transferred to the Output port. An Output operation is indicated by a pulse being Output on the WRITE_STROBE. This strobe signal will be used by the interface logic to ensure that only valid data is passed to external systems. Typically, WRITE_STROBE will be used as clock enable or write enable (see ‘READ and WRITE STROBES’).

3.4.6 Interrupt

The processor provides a single Interrupt input signal. Simple logic can be used to combine multiple signals if required. Interrupts are disabled (masked) by default, and are then enabled and disabled under program control. An active Interrupt forces KCPSM3 to initiate a ‘CALL 3FF’ (a subroutine call to the last program memory location) from where the user can define a suitable jump vector to an Interrupt Service Routine (ISR). At this time, a pulse is generated on the INTERRUPT_ACK Output, the ZERO and CARRY flags are automatically preserved and any further Interrupts are disabled. The
‘RETURN’ instruction ensures that the end of an ISR restores the status of the flags and specifies if future Interrupts will be enabled or disabled.

3.4.7 Scratch pad memory

This is an internal 64 byte general purpose memory. The contents of any of the 16 registers can be written to any of the 64 locations using a STORE instruction. The complementary FETCH instruction allows the contents of any of the 64 memory locations to be written to any of the 16 registers. This allows a much greater number of variables to be held within the boundary of the processor and tends to reserve all of the I/O space for real Inputs and Output signals. The 6-bit address to specify a Scratch pad memory location can be specified in the program as an absolute value (ss), or may be indirectly specified as the contents of any of the 16 registers (sY). Only the lower 6-bits of the register are used, so care must be taken not to exceed the 00 - 3F16 range of the available memory.

The KCPMS3 macro is provided as a kcpsm3.vhd. This code is suitable for implementation and simulation of the macro. This code should not be modified in any way. The Picoblaze Microprocessor is supplied as a VHDL source file, called KCPSM3.vhd, which is optimized for efficient and predictable implementation in a Spartan-3, Spartan-6, and Virtex-6 FPGA. The code is suitable for both synthesis and simulation and was developed and tested using the Xilinx Synthesis Tool (XST) for logic synthesis and ModelSim for simulation. Designers have also successfully used other logic synthesis and simulation tools. The VHDL source code must not be modified in any way.

VHDL component declaration of KCPSM3

component kcpsm3
Port (  
  address : out std_logic_vector(9 downto 0);
  instruction : in std_logic_vector(17 downto 0);
  port_id : out std_logic_vector(7 downto 0);
  write_strobe : out std_logic;
  out_port : out std_logic_vector(7 downto 0);
  read_strobe : out std_logic;
  Interrupt : in std_logic;
  Interrupt_ack : out std_logic;
  reset : in std_logic;
  clk : in std_logic;
  in_port : in std_logic_vector(7 downto 0));
end component;

VHDL Component Instantiation of KCPSM3

processor: kcpsm3

    port map(  
        address => address,
        instruction => instruction,
        port_id => port_id,
        write_strobe => write_strobe,
        out_port => out_port,
        read_strobe => read_strobe,
        in_port => in_port,
        Interrupt => Interrupt,
        Interrupt_ack => Interrupt_ack,
reset => reset,

clk => clk);

The KCPSM3 Assembler will generate a VHDL file in which a block ram and its initial contents are predefined. This VHDL can be used for implementation and simulation of the processor.

The Picoblaze program ROM is used within a VHDL design flow. The Picoblaze Assembler generates a VHDL file in which a block RAM and its initial contents are defined. This VHDL file can be used for both logic synthesis and simulation of the processor. The name of the program ROM, shown as "in_test" in the following figures, is derived from the name of the Picoblaze Assembler source file. For example, if the Assembler source file is named phone.psm, then the Assembler generates a program ROM definition file called in_test.vhd. To speed development, a VHDL file called embedded_KCPSM3.vhd is provided. In this file, the Picoblaze macro is connected to its associated block RAM program ROM. This entire module can be embedded in the design application, or simply used to cut and paste the component declaration and instantiation information into the user’s design files.

VHDL Component Declaration of in_test

component in_test

Port (    address : in std_logic_vector(9 downto 0);

        instruction : out std_logic_vector(17 downto 0);

        clk : in std_logic);

end component;

VHDL Component Declaration of in_test
program: in_test

port map(  address => address,
            instruction => instruction,
            clk => clk);

3.5 Torus structure

As technology scales and chip integrity grows, on-chip communication is playing an increasingly dominant role in System-on-Chip (SoC) design. NoC is a new generation of communication infrastructures for SoC. It consists of a grid of cores where each core can be a SoC, an IP, a DSP, etc. Figure 3.6 shows a mesh NoC consisting of 16 cores. Each core is connected to a router by a local network interface. Each router is connected to its neighbors through a bidirectional channel [12].
For pursuing super IC processor with high speed and performance, more and more heterogeneous processing cores and reused IPs have been integrated on a single chip. When there are hundreds, even thousands of homogenous or heterogeneous IPs the intercommunication among different IP blocks becomes the focus in designing high performance and reliable on-chip-systems. When the amount of IPs becomes larger, the bus and point-to-point technology will pose a high cost, including interconnect delay, power consumption and traffic collision.

The routing and arbitral algorithms of Torus are more complex than those of Mesh since there are more loop links in Torus. In general, XY route, which is widely used in Mesh, is also available in Tours when virtual channels are added to build virtual networks. However, the virtual channels will lead to more complex algorithms and more demands of NOC resource. In our design we choose a typical 16 (4 by 4) routers Torus and propose a corresponding routing algorithm. All the 32 channels are divided into 4
different dimensions. By following the dimension increment method, we break the
dependent route circles, and avoid dead-lock and live-lock.

3.6 Proposed Torus structure

2D Torus can be considered as an extension of 2D Mesh by adding a long circle route
between two boundary nodes. It makes the degree of every router become 4. (Some
routers’ degree is 2 or 3 in Mesh). We consider a regular $m \times n$ 2D Torus where $m = n$.

![Figure 3.7 4 by 4 Torus](image-url)
One target of routing algorithm design is to avoid dead-lock and live-lock. In 2D Mesh, the prohibition of specific turns is a traditional technique to avoid dead-lock based on the idea of breaking certain loops. There are three types of algorithms, West First, North Last, and Negative First. Since Torus has many more loops, routing algorithm of traditional Torus structure is more complex than that of Mesh structure. In order to reduce the complexity of the algorithm and save on chip resource, we redefine the Torus structure.

Number the 16 routers, every router is represented by a 4 bit binary denotation, and every bit represents one dimension, recorded as \((n3n2n1n0)\), the 16 routers are recorded as 0000,0001………………….,1111 respectively. Every route node has only 4 channels to connect other 4 route nodes. These four channels are dual direction (including Input and Output). The logical structure does not correspond to the North, South, West and East as definition 2 and 3 defined in physical level. In our Torus structure, each router denotations \(n3, n2, n1, n0\) corresponds to each of four dimensions.

For two neighboring routers \((a3a2a1a0\) and \(b3b2b1b0\)) to be placed next to each other their denotations must satisfy the following:

\[ a3a2a1a0 \texttt{xor} b3b2b1b0 = r3r2r1r0, \text{then}\ r3+r2+r1+r0=1. \]

The 32 channels are divided into 4 different dimensions. The channels that belong to the same dimensions cannot be connected to the same routers. Based on this logic we can get a redefined Torus structure with new router denotations and channel allocation, the Torus structure is reconstructed as shown in figure 3.
3.7 Routing algorithm

The routing process is a comparison process of denotations between the present router and destination router. The router compares its own denotation to the denotation of destination router. First, compare the denotations of the lowest dimension, if they are different, then route in the direction of the lowest dimension, if not, then compare the higher dimension, and repeat the same process as before. The rest of the process may be deduced by analogy until the denotation of present router is the same as the destination router. Our method makes the routing algorithm simpler, with fewer hops than XY route, which is widely used in mesh. Assume that the destination address of packets is $d_3d_2d_1d_0$ and the denotation of present router is $p_3p_2p_1p_0$, our routing algorithm is shown as follows.

\[
\text{for}(i=0, i<n-1, i++)
\]

\{

\text{if } d_i \text{ is not equal to } p_i

\text{then route packet in } i \text{ direction;}

\}

\text{end;}

For example, a data packet will transmit from 0000 to 1011, the hop will be 5 in XY route mesh structure. But, in our algorithm, the hop is only 3.

A routing algorithm is Deadlock-free if the network channels can be enumerated such that the algorithm always routes the packets along channels with strictly increasing (or decreasing) numbers [13].

In our Torus structure, there are four dimensions, which can be considered as four sub networks, $Gs_3$, $Gs_2$, $Gs_1$, $Gs_0$. In one sub network, any two channels are not connected and any packet will route only once in the same sub network in our algorithm. So, the packet must be routed along channels with strictly increasing (or decreasing) numbers in a certain sub network. Our algorithm is dead-lock free in $Gs_0$. Similarly it is also dead-lock free in $Gs_1$, $Gs_2$, $Gs_3$ respectively. At last, when packet enters the network, it will be transmitted in a sub network once, then enters another sub networks and never return back to the same sub network, thus there is no loops beyond sub networks. Thus the
algorithm is dead-lock free in the whole network. In addition, this algorithm routes the packets along the shortest route, so it is also live-lock free.

3.8 Memory

Memory consist of a four bit Input address (a0,a1,a2,a3) and a clock. The Output of the memory is divided into four parts namely the source (sm0-sm7), destination (dm0-sm7) data(y0-y7) and enablerom (enable0-enable7). When a four bit address is given at a0-a3 we get corresponding Outputs at source (sm0-sm7), destination (dm0-sm7) data(y0-y7) and enablerom (enable0-enable7). For example

```
if addmemory ="0010" then
    sm <= "00000011";-----source address
    dm <= "00000000";-----destination address
    y <= "00001111";-------data to be passed between the two instances of Picoblaze
    enablerom<='00000001';----------enable the source router
```
Figure 3.9 Memory

The Source Output (sm0-sm7) is connected to a source selector Inputs via signals sa-sh as ss0-ss7. The source selector has four select Inputs sel0-sel3 and sixteen Outputs sso0-sso15. The select Inputs are connected to signals sa-sd. So if ss0-ss7 is 00000000, select Input will be 0000 and ss0-ss7 will be placed on sso0. Similarly if sel0-sel3 is 0001, Output will be placed on sso1. The Output of the source selector is connected via signals s1, s8, s15, s22, s29, s36, s43, s50, s57, s64, s71, s78, s85, s92, s99, s106 to the Input of sixteen Picoblaze instances as source_Input.
The destination Output (dm0-dm7) is connected to a destination selector Inputs via signals sj-sq as ds0-ds7. The destination selector has four select Inputs sel0-sel3 and sixteen Outputs dso0-dso15. The select Inputs are connected to signals sa-sd. Since these four signals carry the actual four bit source address. So if ss0-ss7 is 00000000, select Input will be 0000 and ds0-ds7 will be placed on dso0. Similarly if sel0-sel3 is 0001, Output will be placed on dso1. The Output of the source selector is connected via signals s2,s9,s16,s23,s30,s37,s44,s51,s58,s65,s72,s79,s86,s93,s100, s107 to the Input of sixteen Picoblaze instances as destination Input.
The data Output (y0-y7) is connected to data selector Input via signals sr-sy as ds0-ds7. The data selector has four Select Inputs sel0-sel3 and sixteen Outputs yso0-yso15. The select Inputs are connected to signals sa-sd. So if ss0-ss7 is 00000000, select Input will be 0000 and yso0-ys7 will be placed on yso0. Similarly if sel0-sel3 is 0001, Output will be placed on yso1. The Output of the data selector is connected via signals s3,s10,s17,s24,s31,s38,s45,s52,s59,s66,s73,s80,s87,s94,s101, s108 to the Input of sixteen Picoblaze instances as data_Input.
The enablerom (enable0-enable7) Output is connected to enable selector Input via signals sz0-sz7 as es0-es7. The enable selector has four select Inputs sel0-sel3 and sixteen Outputs ess0-eso15. The select Inputs are connected to signals sa-sd. The Output of the enable selector is connected via signals s3,s10,s17,s24,s31,s38,s45,s52,s59,s66,s73,s80,s87,s94,s101, s108 to the Input of sixteen routers instances as enable. If enable ="00000001” then corresponding router will be activated.
Figure 3.13 Enable Selector
3.9 Torus Interconnection and placement of Routers

Figure 3.14 Router ID and their placements (Showing various channels and Dimensions)
The 32 channels have been divided into four Dimensions. Dimension n0 consist of channels 1, channel 3, channel 4, channel 6, channel 7, channel 9, channel 10, channel 12. Dimension n1 consist of channel 2, channel 32, channel 3, channel 31, channel 8, channel 30, channel 11, channel 29. Dimension n2 consist of channel 13, channel 14, channel 15, channel 16, channel 21, channel 22, channel 23, channel 24. Dimension n3 consist of channel 17, channel 18, channel 19, channel 20, channel 25, channel 26, channel 27, channel 28. Following the dimension increasing method data is routed as described above.

3.10 Interconnection and Interaction among memory Picoblaze and routers

As mentioned above the signals from memory go to the Input ports of the Picoblaze, source_data goes to Input port 01, after reaching the Input port 01 it is stored in the Scratch pad memory of the Picoblaze. The destination_Input will arrive at Input port 02 and shall be stored in one of the internal registers of the Picoblaze, register s1, similarly data_Input shall arrive at Input port 03 of the Picoblaze and be stored in register s2. After getting stored at the Input registers it shall be Outputted at destination_Outputport, 05 and data_Output_port, 06 respectively. Incoming data is connected at Input port 07, and this is the actual data that was to be passed to the destination Picoblaze, once the data arrives at 07 it is stored in s4 and then Outputted to an Output port 08. A led is connected at these Output ports to verify if the data reached the desired Picoblaze.

Each Picoblaze is connected to a router through two Output signals of the Picoblaze, destination_Outputport and data_Output_port and one Output signal of the memory that is enablerom. The router connected to the Picoblaze will have an enable of ‘00000001’ if the Picoblaze receives data at its Input ports. Once the router is enabled it reads the
destination_Input data and data_Input and stores them in destination_variable and
data_variable registers. Depending on the destination address it will route the data as well
destination (16 bit data) along the channels that connect the routers, all the channels are bidirectional.
As seen above from the outgoing data port of the Picoblaze we can verify if the desired data has reached the desired destination.
CHAPTER 4

EXPERIMENTAL RESULTS

The proposed Architecture is tested by writing a testbench. The Input of memory are a0,a1,a2,a3 which is given as an Input for the testbench. Depending on the value of a0,a1,a2,a3 the memory will produce some source address, destination address, data and activate the enable.

For example lets say a0,a1,a2,a3 is 0000

Therefore source = 00000000

Destination =00001010

Data = 11111111

Enable or eso0=’00000001’

As seen from the above data since source is 00000000 Picoblaze_0000 will be activated. Once it receives data at its source_Inputport it will store in Scratch pad memory. Destination data at the destination_Inputport will be stored in regsiter s1, data at the data_Inputport is stored in Input regsiter s2. On the activation of clock data from these Input regsiters they will be stotred at the Output regsiters of the Picoblaze and eventaually will be seen at destination_Outputport and data Output_port. As seen from the circuit diagram, signal 5 and signal 6 are connected to destination_Outputputport and data_Outputport of Picoblaze_0000 to the Input of router_0000 which are desti_0000 and xdata_0000 respectively.

Since eso0 is at logic 1, router_0000 is enabled. It will store desti_0000 in interanl regsiter desti_register_0000 and xdata_0000 in xdata_0000. Once it is placed in the
interanl regsiters it will read the destination id, which in this case is 00001010. It will read only the first four bits 1010. The router shall now apply the routing logic as below:

Source=0000 and Destination=1010

As seen from above, bit n0 is same for source and destination address hence move on to the next bit n1, since n1 differs move in direction of n1 dimension. Thus channel32_up which is a 16-bit signal will carry desti_0000 and xdata_0000. This will reach router_0010 at its east_in_0010 Input. At router_0010 the destination bits arriving at its Input are compared with its id that is 0010. Bit n2 of 0010 and 1010 are comapred, since they are same, bits n3 are compared, since they differ destination and data bits will move in direction of n3 and reach 1010, which is the destination router. These bits will travel along channel28_down.

Once the destination bits and data reach the desired router, the data bits will be given to Output_reg_1010, which is connected to the Input of a Picoblaze_1010 incomming data port.

In order to verufy if the data reached Picoblaze_1010 which is our destination, the incomming data is Outputted onto outgoingdata_port where an led is connected.

4.1 Simulation

The Complete step by step result of Simulation is as shown:
As seen in the above figure 4.1 for Picoblaze_0000 destination Input and data Input value is Output at destination Output and data Output port resoectively. This transfer from Input to Output is done via in_test.psm file, which is the assembly language file of the Picoblaze. The fact that the data is seen at the Output proves the fact that the Picoblaze is working as desired.
As explained above destination and data will travel along channel 32_up and reach Input of router_0010 at east_in_0010. This value is placed on channel28_down through north_out_0010 as per the rule mentioned above.

Figure 4.2 Output of router_0010

The data placed on channel28_down will go to router_1010’s south_in_1010.
Figure 4.4 Router_1010
As seen from Figure 4.5 the data 1111111 has reached the desired destination Picoblaze.

A similar logic is followed for any data transfer between any two Microprocessors.
4.2 Synthesis

After simulation the next step is to perform Synthesis. But before doing that we need to select a board that could hold such a large design. We started with sparten 3e but the synthesis results showed that sparten 3e was too small to hold this design. Hence Virtex-4 is implemented for Synthesis.

Figure 4.6 Synthesis with Sparten 3E
As seen in Figure 4.7, the Number of Slices available in Virtex 4 is huge and it can easily accommodate the proposed design. Hence the device selected for synthesis is Virtex 4 XC4VFX12 FF668.
The complete synthesis summary is as shown below:

As seen from the Device Utilization Summary Report the total number of slice registers used by the proposed design is 2940 out of 10,944, hence the percentage utilization is only 26%. After successfully generating a synthesis report the next step is to use the Xilinx Clocking Wizard.
Architecture Wizard is used to configure a DCM component to Output a clock at 55 MHz.
Place and Route

Using the Virtex-4 user manual we assign pin locations as above.
Figure 4.11 Device Schematic
4.6 Virtex 4 Schematic

As seen in the schematic below we are using the 8 leds to verify the Output.
The 8 LEDs on the board will light when the desired data will reach the desired destination. These LEDs will be driven by the outgoing data port of the Picoblaze (expected data). Thus helping us to verify that FPGA is working as intended.
CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

In this thesis we have designed a network structure with as many as sixteen instances of Picoblaze. The first biggest challenge has been in deciding the topology that would work the best in terms of passing data back and forth, we implemented the architecture in torus. The next step was to decide on how these routers would be placed and a routing logic.

In this thesis, we reconstruct Torus structure for NOC application. In this structure, the routes are denoted as four dimensions and are relocated at new positions. All the 32 channels are divided into 4 dimensions, each corresponding to every bit of route node denotation. Based on our structure, we propose a dimension increasing routing algorithm, in which every route node only needs to compare its own denotation with the denotation of destination and route the data packet to the dimension in which the two denotation bits are different. This algorithm requires fewer hops than traditional XY route widely used in mesh topology for NOC applications. The simplicity of the routing algorithm leads to a convenient implementation of NOC router.

We build a Torus structure with Picoblaze instances and implement the routing algorithm using Xilinx ISE 11.1. According to the simulation results, our Torus network and its associate routing algorithm, the sixteen instances of Picoblaze and the memory module are working as per planned.

Any Picoblaze can pass data to any of the other fifteen instances efficiently and smoothly as the Dimension Increasing Routing Algorithm. We can verify if the data has reached the destination Picoblaze by checking its Output data port.
This thesis presents a design that enables multiple instances of an 8-bit Microprocessor to transmit data choosing the best possible path. The Dimension Increasing method which has been adapted for our routing logic ensures that the routing is Deadlock free. It even eliminates the use of virtual channels which lead to more complex algorithms and more demands of NOC resource.

5.1 Recommendations

The existing design can be improved by enabling simultaneous data transmission among the processors. The present design enables data routing between two processors, one at a time. Simultaneous data routing can be achieved by modifying the existing hard coded memory such that it generates multiple source, destination and data bits at a time and passes to multiple processors. This enhancement shall significantly improve the performance of the design as well as making the architecture more efficient.


[28] Xilinx “Picoblaze 8-bit Microprocessor for CPLD Devices” XAPP387 (V1.1), January 9 2003


VITA

Graduate College
University of Nevada, Las Vegas

Arpita H Kadakia

Degrees:
Bachelor of Electronic Engineering, 2007
Vidhyalankar Institute of Technology, India

Diploma in Electronics and Communication Engineering, 2004
Bombay Institute of Technology, India

Thesis Title:
Data Routing in Multicore Processors Using Dimension Increment Method

Thesis Examination Committee:
Chairperson, Dr. Henry Selvaraj, Ph.D
Committee Member, Dr. Shahram Latifi, Ph.D
Committee Member, Dr. Emma E. Regentova
Committee Member, Dr. Lawrence L. Larmore