An Analysis of NoCs in FPGAs

AN ANALYSIS OF NOCS IN FPGAS

BY

MOHAMMADREZA BINESH MARVASTI, M.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Doctor of Philosophy (2013) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	An Analysis of NoCs in FPGAs
AUTHOR:	Mohammadreza Binesh Marvasti M.Sc., (Electrical and Computer Engineering) University of Tehran, Tehran, Iran
SUPERVISOR:	Dr. Ted H. Szymanski
NUMBER OF PAGES:	xxiii, 208

To my dear parents who devoted their life to their children

Abstract

Accurate analytic models for the area, delay and power of NoC routers realized in FPGA technology are presented. Several router designs are explored, including the demultiplexer-multiplexer design, the broadcast-and-select design, a RAM-based design, and pipelined designs with arbitrary amounts of buffering. The buffers can be realized using embedded memory blocks or using D flip-flops. The analytic models are compared with extensive experimental results, and shown to be very accurate. Using these router models, accurate analytic models for the area, delay and power of graph-based and hypergraph-based NoC topologies realized in FPGAs are presented, including 2D Mesh, Torus, Binary Hypercube (BHC), Generalized Hypercube (GHC), and Hypermesh. Three traffic patterns are considered, (a) Random-Uniform traffic patterns, (b) traffic patterns in Bitonic sorting algorithm, and (c) traffic patterns in FFT parallel algorithm.

The analytic models for NoCs are compared to extensive experimental results and shown to be very accurate, typically within 10%. Using these analytical models, architectural choices such as NoC topology, buffer sizing, crossbar switch design, and degree of pipelining can be explored analytically early in the design-space exploration process. It has been observed that an efficient and accurate early design process results in lower system costs, and in order to come up with feasible designs, early

design-space exploration tools are essential.

Early design-space exploration tools using analytic models are ideal, as they do not require the generation of detailed logic design in a hardware description language such as VHDL or Verilog. However, to date there are no analytic models for NoCs in FPGAs. This thesis addresses this problem.

According to our analytic power models, in an FPGA environment with equal bisection bandwidth the 2D BHC outperforms the 2D Mesh and Torus significantly. For example under equivalent bisection bandwidth, when performing FFT computations in an FPGA environment the 2D BHC consumes $\approx 8\%$ of the power of a 2D Mesh, and $\approx 15\%$ of the power of a 2D Torus.

Hypermeshes are based on the concept of hypergraphs, which consist of a set of nodes and a set of hyperedges, where the hyperedges represent low-latency switches. Under equivalent bisection bandwidth, 2D Hypermesh NoCs outperform the 2D Mesh and Torus significantly. To improve the performance of the Hypermesh, two new hyperedge designs are proposed. We propose the *energy-area product* as a design metric to compare the NoCs. The *energy-area product* reflects both the cost and performance design metrics. Our analysis indicates that the 2D Hypermesh NoCs generally have considerably lower area, energy, and *energy-area product* compared to the 2D Hypercubes. Under equal bisection bandwidth, the area usage of the 2D Hypermesh using the broadcast-and-select designs as the hyperedges uses 30% of the area of the GHC and 42% of the area of the BHC. The *energy-area product* of the 2D Hypermesh under the FFT algorithm is 9% of the GHC, and 29% of the BHC.

Acknowledgements

First of all, I would like to thank God, who has given me many blessings in my life. A number of people have contributed to my success throughout this thesis.

First and foremost, I would like to thank my supervisor Prof. Ted H. Szymanski for a tremendous source of support. I have been extremely fortunate to work under his supervision and I deeply appreciate his time, patience, and support throughout my Ph.D career. His assistance, constructive feedbacks, and technical insights have always been most helpful in addressing my research issues.

I would like to express my deepest gratitude to my family for their support, encouragement, and unfaltering belief in my ability to succeed in my career. Their love and prayers made me the person that I am today.

I would like to thank the members of my committee, Prof. Nicola Nicolici and Prof. Terry Todd for their useful guidance and suggestions. I am particularly grateful to Prof. Nicolici for the technical assistance and useful discussions.

A special thanks to Mrs. Cheryl Gies for the countless conversations and sincere advices that have helped me in last four years. Finally, I appreciate all the staff at the ECE department of McMaster University.

Notation and abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
ACK	Acknowledgement
ALM	Adaptive Logic Module
ASIC	Application-specific integrated circuit
B&S	Broadcast-and-Select
BHC	Binary Hypercube
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
DCS	Distributed Crossbar Switch
DFF	D flip-flop
Demux	Demultiplexer
DM	Demux-Mux
DSP	Digital Signal Processor
EDA	Electronic Design Automation
EM	Embedded Memory Block
FIFO	First In First Out
FLIT	Flow control digit

FPGA	Field Programmable Gate Array
GHC	Generalized Hypercube
HDL	Hardware Description Language
HF	Header Flit
HM	Hypermesh
IC	Integrated Circuit
IP	Intellectual Property
LAB	Logic Array Block
LE	Logic Element
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look-up Table
Mux	Multiplexer
ND	Node-Distance
NoC	Network-on-Chip
PDM	Pipelined Demux-Mux
PE	Processing Element
RU	Random Uniform
SDC	Synopsys Design Constraint
SDM	Space-Division Multiplexing
SoC	System-on-Chip
TDM	Time-Division Multiplexing
TF	Tail Flit
QoS	Quality-of-Service
VC	Virtual Channel

VCD	Value Change Dump
VLSI	Very Large Scale Integration
VHDL	VHSIC hardware description language

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Network on Chip (NoC) is a new paradigm for System on Chip (SoC) design that uses an interconnection network for communication instead of buses. NoCs are high performance, scalable, and power efficient alternative to the bus-based architecture [1][2][4]. Some chip vendors are exploiting NoCs in their chip architectures (such as the AMD Radeon HD 2900 series).

The design of NoCs is a trade off between multiple constraints such as minimizing power, minimizing area, and minimizing packet delay [53]. It is critical to explore early design spaces in order to achieve near-optimal designs due to the trade offs between area, latency, performance, and power consumption.

Early design-space exploration tools can follow analytic or simulation-based methodologies. Most studies in NoC domain have focused on software simulation in order to evaluate design metrics [5][6][7][41]. These software simulation tools such as *Altera's PowerPlay* can be highly accurate however they are complex to develop and they lack a theoretical basis. Furthermore, simulation based tools require a detailed hardware design which must be developed in Hardware Description Languages such as VHDL,

Verilog, System Verilog, etc.

On the other hand analytic models have a theoretical basis and they do not require any detailed logic design in VHDL or Verilog, and they are reasonably accurate [8][56]. An evaluation through an analytical model can be performed much faster compared to the amount of time which simulation would take. An accurate design requires a detailed model however this leads to a much complicated analysis. Therefore, in process of developing analytical models, a trade off exists between accuracy and complexity.

Another important advantage of analytic models is to enable the early design optimization. Usually 90% of system costs are fixed early in design cycle, before detailed design in VHDL or Verilog is performed. NoC designers cannot design, test, and debug multiple different NoCs in VHDL or Verilog to find the best NoC. Early design space exploration tools are needed to find the best designs and also fix the significant system costs before detailed design in VHDL or Verilog can begin.

Field Programmable Gate Arrays (FPGAs) are considered a good alternative to the Application Specific Integrated Circuits (ASICs) to accommodate NoC-based SoCs due to the following advantages: 1) low development cost, 2) low time to market, 3) ease of upgrading, and 4) wide range of memory and functional blocks as well as soft processors.

The FPGA-based NoC is one of the most recent attractive fields of research. It is important to choose a right set of NoC architectures as well as FPGA features that work best together. To date, there are no analytic models for the area, delay and power of NoCs realized in FPGA technologies. The goal of this thesis is to propose analytic models which enable NoC designers to perform an early design exploration

in FPGAs.

1.1 Thesis Contributions

In this thesis, analytic models of NoC routers, and graph and hypergraph-based NoC topologies in FPGAs are proposed. The models are very accurate and general which allow early design-space exploration and evaluation before detailed design are undertaken.

This thesis makes the following specific contributions:

1. *Analytic power, area, and delay models for the basic components of NoCs in FPGAs and several crossbar switch designs*: These switches are *Pipelined and unpipelined Demultiplexer-Multiplexer, Broadcast-and-Select, and Memory-based*. The analytic models for switches with 2 different types of input buffers, Embedded Memory Blocks and DFF Registers, are presented. Different routers can be made by different combinations of switches and input buffers. These combinations should be considered carefully to choose the best candidate for different applications.

2. *Analytic area, delay, and power models for 5 different NoC topologies in FPGAs*: These NoCs are 2D Mesh, Torus, Binary Hypercube (BHC), Generalized Hypercube (GHC), and Hypermesh. These models can be extended to ASICs easily.

3. *Analytic power models for wormhole-switched NoCs in FPGAs*: Analytic power models for crossbar switches, input buffers, and links in wormhole-switched

NoCs are proposed. These power models depend on the topology and characteristics such as expected distance and number of hops traversed by one packet, number of links, number of ports per router, and length of inter-router links. To evaluate an NoC under a specific algorithm, we only need the expected distance and number of hops traversed by a packet in the algorithm. These models can be extended to ASICs easily.

4. *The expected number of hops and distance traversed by a packet for the 5 topologies under Random Uniform traffic pattern, the traffic pattern in the Bitonic sorting algorithm, and the traffic pattern in the Cooley-Tukey FFT*: These traffic patterns are mathematically modelled and are used for the power and energy evaluation of the NoC topologies.

5. *Evaluation of the NoCs*: The NoCs are compared based on their resource usage and power consumption. The NoCs are also evaluated by their energy consumption per algorithm. The energy consumption of the NoCs are considered under 2 parallel algorithms: (1) Bitonic sorting algorithm, and (2) Cooley-Tukey FFT.

6. *New Hyperedge Designs*: Conventional hyperedge designs use the Broadcast-and-Select switch design. To improve the performance of Hypermeshes, two new hyperedge designs are proposed, which are explained in Appendix C.

7. *Energy-Area Product*: The energy-area product is proposed as a proxy for a comparison between topologies, which reflects both the cost and performance metrics in the NoC domain. The NoCs are also compared in terms of the *energy-area product*.

1.1.1 Publications

This thesis includes work that has been published or submitted as follows:

1. M. Binesh Marvasti, and T. H. Szymanski, "The performance of hypermesh NoCs in FPGAs," In Proc. 30th IEEE International Conference on Computer Design (ICCD), 2012.

2. M. Binesh Marvasti, and T. H. Szymanski, "A power-area analysis of NoCs in FPGAs," In Proc. 25th IEEE International SOC Conference (SOCC), 2012.

3. M. Binesh Marvasti and T.H. Szymanski, "An Analysis of Mesh and Hypercube NoCs in FPGAs," Submitted to IEEE Transactions on CAD, 2013.

4. M. Binesh Marvasti and T.H. Szymanski, "An Analysis of Hypermesh NoCs in FPGAs," Submitted to IEEE Transactions on Parallel and Distributed Systems (TPDS), 2013.

1.2 Thesis Organizations

This thesis is organized as follows: Chapter 2 presents background information related to both NoC architectures and FPGAs. Chapter 3 explains two different simulators: 1) a "pseudo random walk" simulator for area, delay, and maximum power analysis, and 2) a wormhole simulator for the power analysis under different traffic patterns. Chapter 4 presents analytic area, delay, and power models of basic components of NoCs in FPGAs and seven different router designs. Accurate area, delay, and power analysis of 5 NoC topologies, 2D Mesh, Torus, BHC, GHC and Hypermesh NoCs, are presented in chapters 5 and 6. In these chapters, analytic models for random uniform

traffic pattern, traffic patterns in Bitonic sorting algorithm and traffic patterns in FFT algorithm for the topologies are presented. Chapter 7 concludes the thesis and presents future directions of this research.

Chapter 2

Network on Chip and FPGAs

In a System on Chip (SoC), the communication between Processing Elements (PEs) (or IP cores) is handled by dedicated wiring or shared bus. However, the use of dedicated wires or shared bus is problematic for several reasons. First, buses are limited in connecting between 3 to 10 PEs [34] and they can not scale to large systems with numerous components. Second, as the number of cores increases on chip, the amount and length of wiring required to connect every component increases. This results in long global latency and higher power consumption.

The Network on Chip is an attractive alternative to dedicated wiring or shared bus in SoCs. It represents a scalable solution for communication between IP cores.

The Networks on Chip can leverage ideas of topology, switching strategy, routing algorithm, and flow control policy from the off-chip networks. However, there are some key differences between NoCs and off-chip networks.

First, NoCs should be designed under very tight area and power budgets. Integrating a large number of cores under area and power constraints is a big challenge for designers. The interconnection networks in the NoCs use a significant amount

of chip power, i.e. 30% of Intel's 80 core network [9] and 36% for the RAW on-chip network (MIT's Microprocessor) [10].

Second, in off-chip networks, communication latency depends on the link transmission latency. However in NoCs where links are relatively short, data can traverse the links with low latency. This quick traversal makes the router latencies dominate in the overall communication latency [11].

Another key difference is bandwidth. Bandwidth in on-chip networks is significantly higher than off-chip networks due to the cheap and plentiful wires in the chip [11]. Therefore, NoC architectures are supposed to offer higher performance while having limited area and power budget.

As Field Programmable Gate Array (FPGA) capacity and capability grow, and they are increasingly used to build a wide range of SoC applications. Therefore, FPGAs can be one of the best candidates for NoC designs.

In this chapter, we give an overview of FPGA architectures and explore the NoC characteristics that have been presented in the literature.

2.1 Topology

Topology relates to the physical layout and connectivity between nodes in the network. Topologies can be classified into four main categories: direct, indirect, graph-based, and hypergraph-based networks.

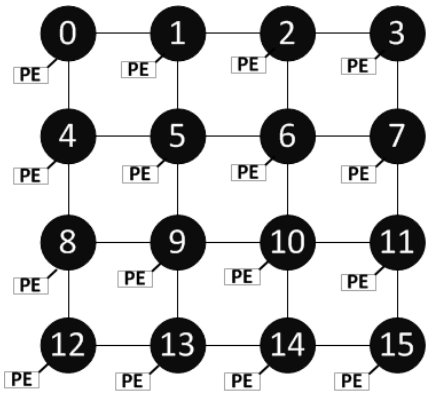


Figure 2.1: 2D Mesh with 16 nodes

2.1.1 Direct Topologies

In a direct network topology, each router is connected to a network node (IP core). In these topologies, as the number of routers in the system increases, the total available communication bandwidth usually increases as well. This offers a desirable scalability to implement massively parallel computers. However, an essential tradeoff between the offered bandwidth and area of routers should be taken into consideration. Higher bandwidth results in lower latency but increases the energy and area costs for the router and link implementations.

In this category, 2D Mesh and Torus (Mesh with wrap-around links) are popular in the NoC domain as they can be easily mapped into the floorplan. Figures 2.1 and 2.2 show an example of 4x4 2D Mesh and Torus.

2.1.2 Indirect Topologies

In indirect network topologies, routers have point-to-point connections to other routers [35], and there are several intermediate routers that are not connected to network nodes (IP core). Examples of popular multi-stage indirect networks are the fat-tree

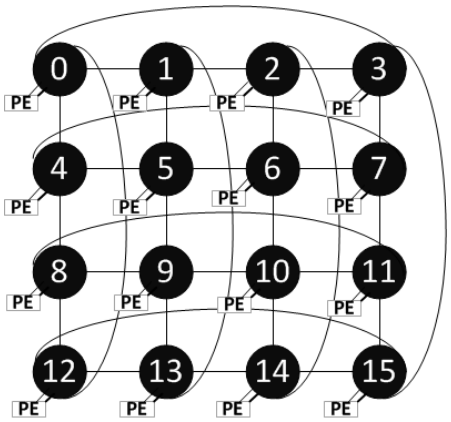


Figure 2.2: 2D Torus with 16 nodes

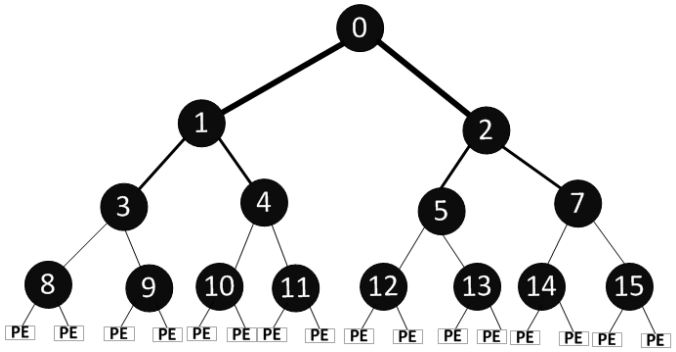


Figure 2.3: Fat-tree with 16 nodes

and butterfly networks [35]. Figure 2.3 shows a fat-tree with 16 nodes (PEs). In the fat-tree topology the root and its neighbors have higher traffic than leaves, therefore the datapath width of the links increase as they get closer to the root. Another example of indirect topologies is a butterfly network. A butterfly topology can be described as k-ary n-fly, which includes k^n nodes (IP Core) and n stages of switches. At each stage there are K^{n-1} switches [35].

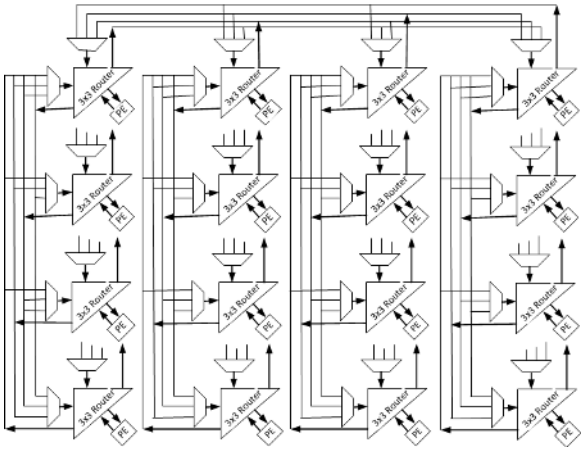


Figure 2.4: Hypermesh with 16 nodes

2.1.3 Graph-based Vs. Hypergraph-based Topologies

In conventional graph-based network such as the Mesh and Torus, each node has a router with direct connections to its nearest neighbors.

Hypergraphs are generalizations of conventional graph in which edges are generalized to yield hyperedges. A hyperedge represents a logical relationship among arbitrary number of vertices rather than just two vertices [12]. An example of hypergraph-based topology is Hypermesh which is shown in figure 2.4.

2.2 Flow Control

In NoCs, flow control is defined as a policy that determines how resources are allocated to messages as they travel through the network [17]. Flow control policy plays a significant role in enhancing the system performance and reliability by managing buffers and links allocated to packets. The flow control is classified into 2 classes: (i) bufferless and (ii) buffered. In NoCs, switching mechanisms are considered in the flow

control domain [13]. The switching mechanism determines how and when messages can traverse the routers in the network. In NoCs, a generated message is broken into multiple packets. A packet can be further broken down into a number of *flits* (flow control digits). Circuit switching and packet switching are two main forms of switching mechanisms and are explained in the following sub-sections.

2.2.1 Circuit Switching

Circuit switching is a form of bufferless flow control which works at the message level. In circuit switching, a physical path between a source and a destination is set up and reserved to transmit an entire message. Circuit switching suffers from poor scalability as the size of the network grows. Moreover it suffers from long initialization time for connection setup. This is due to the fact that several links should be reserved for the entire message transmission process [36].

2.2.2 Packet Switching

Packet switching schemes are a form of buffered flow control. In packet switching, messages are broken down into multiple packets and each packet is handled independently by the network [11]. In contrast to circuit switching, packet switching takes advantage of the per-hop basis to transmit the messages. In this mechanism packets include routing information and data to traverse their path towards their destinations. Packets can also have variable delay depending on the traffic load in the network. Packet switching improves the bandwidth utilization, transmission latency, and communication robustness of the NoCs [2]. There are three well-known buffered packet switching schemes: store-and-forward, virtual-cut-through, and wormhole switching

which are explained below:

Store-and-forward

The store-and-forward is a simple switching protocol in which each node waits to store the complete packet before forwarding any part of the packet to the next node [13]. In this type of switching an entire packet must be buffered, and the buffer size in the router must be at-least equal to the size of the packet. If the buffer in the next router has no sufficient space to buffer the entire packet, the packet has to be stalled. This switching technique requires a large buffer size which makes it less attractive for NoC architectures.

Virtual-cut-through

The virtual-cut-through switching allows a packet to move forward to the next node as soon as the header of the packet is received at the current node and resources (buffer and channel) are acquired [13][16]. In this switching technique, following the header flit, the other flits of a packet are transmitted consecutively. This technique decreases network latency experienced by a packet and increases the network throughput since it doesn't wait for the entire packet to be received. In addition, if the packet stalls, it will stay in the current node and will not block any link. The buffer requirement of this technique is the same as the store and forward switching therefore it is not commonly used in the NoC domain.

Wormhole

The wormhole switching [14] uses the concept of circuit switching mixed with the packet switching technique in order to reduce the packet latency compared to the store-and-forward and the virtual-cut-through switching. In this technique, each router processes the header flit of an incoming packet to determine its next hop and then forwards it. The other flits follow the header flit as they arrive. In this switching technique, a packet can be distributed among a number of routers in its path through the network when there is insufficient buffer space in the next router to store the entire packet. A stalling packet blocks the intermediate links occupied by its flits. The wormhole switching reduces packet latency by allowing a flit to move forward to the next router when a sufficient buffer is available for the flit. Moreover, it requires much smaller buffers than the store-and-forward and the virtual-cut-through switching techniques. As a result wormhole switching is the most attractive and cost-effective switching technique for the NoC architectures.

In wormhole switching the network latency of a packet with H hops length is given by [54]:

$$T = H \times (T_{Routing} + T_{Xbar} + T_{Link}) + (m-1) \times (T_{Xbar} + T_{Link}) + Bl \qquad (2.1)$$

where m is the average number of flits in the packet, and where $T_{Routing}$, T_{Xbar}, and T_{Link} are the delays for the routing, crossbar switch, and link traversal respectively. The term Bl is the average blocking time seen by a header flit [54].

2.3 Virtual Channels

Wormhole switching has been proposed to reduce the buffer requirements and enhance the system throughput. However, each packet may occupy several intermediate switches and links at the same time. This may introduce the problem of deadlocks [2]. To avoid this problem, virtual channels are used. Virtual channel flow control exploits an array of parallel buffers at each input port. In this technique, there are multiple queues per physical channel and as a result if one packet is blocked other packets which belong to other virtual channels can use the idle bandwidth. This improves the throughput of the network and reduces the average packet latency by allowing blocked packets to be bypassed.

In general, virtual channels offer many advantages such as avoiding deadlocks by breaking cycles in the resource dependency graph [20], optimizing wire utilization by letting several logical channels share the physical wires, improving performance by minimizing the stall frequency [19], and providing quality-of-service (QoS) by assigning different priorities to different data flows [21].

2.4 Router Architecture

The router forms the heart of the NoC backbone. It is responsible for transporting packets generated by IP cores. It is comprised of input buffers, crossbar switch, routing logic, and allocation unit as shown in figure 2.5. The router shown in figure 2.5 has N+1 ports, where the additional port is used to connect the router to its IP core (PE).

An arriving flit is stored at the input buffer which is divided into virtual channels

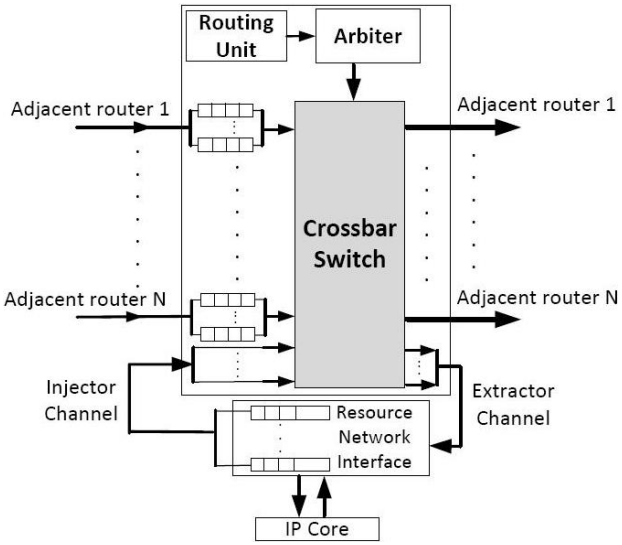


Figure 2.5: An IQ Router

to prevent deadlock and increase throughput. Virtual channels provide multiple input buffers per physical input channel so that packets with different destination output ports can pass blocked packets in a router. The routing unit decodes the routing information in the received header flit in order to find the requested output port. Then the arbitration unit decides if the header flit can proceed toward the output port or not. Once the output port is allocated for the header flit, the header flit traverses the crossbar switch toward the next hop.

2.5 Routing

The routing algorithm determines the path a message takes through the network to reach its destination. Routing algorithm has a direct effect on the performance, power consumption, area, and robustness of NoC-based systems. The routing algorithms can be categorized as unicast or multicast, distributed or source routing, deterministic,

oblivious, or adaptive, minimal or non-minimal.

2.5.1 Unicast and Multicast Routing

A routing algorithm is called unicast when it has a single destination. Multicast (one-to-many) is defined as delivering of a message from a source node to an arbitrary number of destination nodes.

2.5.2 Distributed and Source Routing

Routing can be classified depending on the location of routing information and routing decision logic.

In source routing, the routing path is determined at the source node (before the packet is injected) using pre-computed routing tables which are stored at the associated network interface. Therefore, routing information is inserted to the header of packet and no look-up table or routing logic is needed in the intermediate routers.

In distributed routing, the routing path is determined at each router. Therefore each router must have a routing unit to determine the requested output port of every incoming packet, based on the destination address in the packet.

2.5.3 Deterministic, Oblivious, and Adaptive Routing

According to [13], in terms of how routing algorithms select between the set of possible paths from a source node to a destination node, they are classified into 3 groups.

A deterministic algorithm always produces the same output for a given particular input. Under the assumption of the unicast routing, in deterministic routing algorithms the same path between a source and destination pair is always chosen [13].

This routing algorithm is suitable for NoCs where network traffic is predictable and relatively simple. In deterministic routing algorithms the current state of the network in terms of traffic load, faulty link, temperature are not considered. An example of this routing algorithm is the XY ordered-dimension routing algorithm.

In oblivious routing algorithms a path is selected from the set of possible paths from a source node to a destination node without considering the state of the network [13]. An example of this algorithm is the Max-Flow routing algorithm. Deterministic routing algorithms are the subset of oblivious routing algorithms.

In adaptive routing algorithms a path is selected among alternative paths based on the factors presented in the current state of the network [13]. This adaptivity usually results in more efficient distribution of traffic in a network but at the cost of additional run-time monitoring and management logic. This routing is suitable for networks in which traffic conditions show irregularity and unpredictability.

2.5.4 Minimal and Non-Minimal Routing

Minimal routing algorithms choose the shortest possible path between the source and the destination nodes, while in non-minimal routing algorithms paths may be longer than the minimum path which might be useful for avoiding congestion and critical (faulty or hotspot) regions. NoCs which are sensitive to communication latency typically choose minimal routing algorithms.

2.5.5 Deadlock and Livelock

A deadlock occurs when a group of packets are unable to move in a network because they are waiting for each other to release resources in a cyclic fashion. Deadlock can be

avoided by analyzing and breaking cyclic dependencies in the dependency graph of the shared network resources by applying routing restrictions or using additional hardware resources such as virtual channels [19]. Livelock occurs when packets continue to move through the network, but they do not make progress toward their destinations [13]. In adaptive non-minimal routing, livelock can occur because of its flexible capability of redirecting packets. One of the ways to avoid livelock in a network is to add a misroute count, which holds the number of times a packet has been misrouted [13]. Once the count reaches a threshold, no more misrouting is allowed.

2.6 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a special kind of semiconductor device that can be programmed after manufacturing. FPGAs usually are constructed by arrays of pre-fabricated logic blocks, routing paths, I/Os, and reconfigurable switches. FPGAs have the ability to be programmed many times to implement any desired digital design.

In an FPGA, logic functions are configured through a series of programmable resources. At the lowest level of the hierarchy, these programmable resources are typically called Logic Elements (LEs), Adaptive Logic Modules (ALMs), or Slices. Each of these programmable resources typically includes one or more Look-Up Tables (LUTs) and D flip-flops (DFFs). These programmable resources are configured to perform complex combinational functions or simple logic gates such as AND and XOR. Array of LEs, ALMs, or slices are typically placed in logic blocks. These logic blocks are usually called Logic Array Blocks (LABs) or Configurable Logic Blocks (CLBs). In most FPGAs, the logic blocks also include memory elements which are

simple D flip-flops (DFFs) or more complete blocks of memory.

The design usually is implemented using a Hardware Description Language (HDL), i.e VHDL, Verilog or Systemverilog, which is programmed to the chip after a compilation procedure. This compilation procedure is done by special software such as Altera Quartus II and Xilinx ISE and it includes synthesizing, placement, routing, and floor planning. Afterwards, the FPGA is programmed by loading data bits into the memory cells. These memory cells control transistor switches to establish non-permanent connections.

An FPGA can support upto a few million logic gates operating at speeds of hundreds of MHz [22]. With growing chip density in terms of the number of transistors and gates in the latest process technology and a host of other features, such as embedded processors, embedded memory blocks, DSP blocks, clocking, and high-speed serial at ever lower price points, FPGAs are a suitable platform for almost any type of designs [22][23].

Figures 2.6 and 2.7 illustrate the general architecture of the LE and ALM in Altera FPGAs. As seen in figure 2.6, an LE includes a 4-input LUT and a DFF register. The LUT is a SRAM memory which contains the truth-table of a combinational logic function after FPGA programming. The LUT with n inputs can implement any n-bit function. Referring to the ALM in figure 2.7, each ALM consists of a 8-input LUT and 2 DFF registers.

2.6.1 FPGA Advantages and Disadvantages

FPGAs and Application Specific Integrated Circuits (ASICs) provide different values to the designers. These values must be carefully evaluated. While FPGAs used to

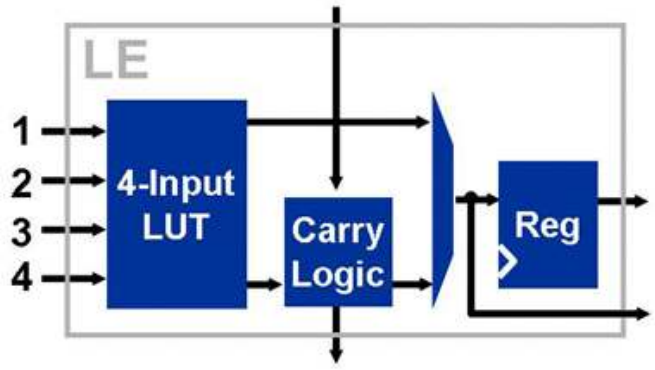


Figure 2.6: Structurer of an LE [23]

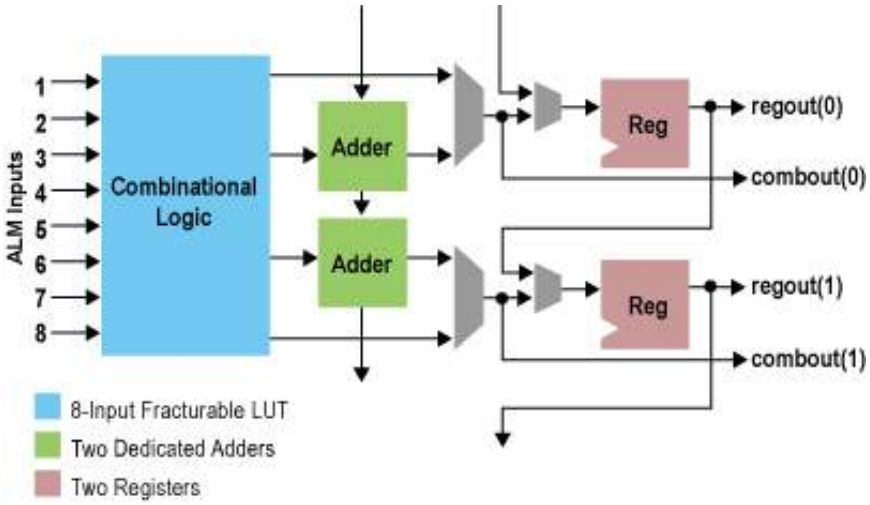


Figure 2.7: Structure of an ALM [23]

be selected for lower speed/complexity/volume designs in the past, todays FPGAs easily push the 500MHz performance barrier [22].

Below is a brief list of advantages and disadvantages of using FPGAs:

1. Advantages:
   (a) Reprogramability: FPGAs can be programmed and re-programmed many times to implement or debug any desired digital circuit
   (b) FPGAs provide flexible and fast prototyping implementation for embedded systems
   (c) Lower development cost and shorter time to market
   (d) FPGAs are using the latest process technologies whereas most ASICs are still using previous process generations
   (e) Suitable for research purposes because of:
      i. Fast design cycle
      ii. Immediate results
      iii. No manufacturing operations involved
2. Disadvantages:
   (a) Generally slower than ASICs
   (b) Limited area
   (c) Need more power
   (d) Not suitable for very high volume designs

2.6.2 Capacity of a Programmable Logic Component in FPGAs

Since FPGAs can be configured in a variety of patterns, and because their fundamental logic structures are different among different FPGA vendors, there is no direct one theoretical mapping to compare the logic unit of one FPGA architecture to another. The comparison can be done after performing full place and route compilations using the appropriate software tools [24]. The efficiency of logic utilization of an FPGA architecture depends on the following factors [24]:

1. The logic capacity of a single logic unit
2. The embedded functions that are present in the FPGA, such as a DSP block or embedded RAM
3. The structure of the design, such as whether the design includes multiplexing, wide functions, or arithmetic functions
4. The effectiveness of the synthesis tool
5. The quality of the place and route software

2.6.3 FPGA-based NoCs

An SoC realized with an FPGA is common today. As FPGA capacity and capability grow, they are increasingly being used to build a wide range of SoC appliances. The increasing logic density with higher operating frequencies as well as functional memory and operating blocks, enable FPGAs to replace ASICs in several high performance applications.

The concept of dynamically reconfiguring FPGAs applies well to micro-network design [2]. The advantage of FPGAs is to efficiently execute a large variety of tasks with distinct requirements using reconfigurable amount of resources [29].

Reference [30] discussed the fixed communication layer of interconnection network on FPGAs to perform dynamic multitasking by tile-based reconfiguration. It was concluded that FPGA-based NoCs are suitable for instances where the reconfigurability is required due to a low hardware overhead. In FPGAs, tasks can be dynamically instantiated in the network by partial reconfiguration, which opens a new way to dynamic multitasking applications.

In FPGAs, there is a rich routing fabric. Reference [31] studied the routability of wiring on FPGAs and explored multiprocessor designs in FPGAs. This was done by considering various NoC topologies and scaling the number of IP cores. It was concluded that it is not required to limit the connectivity to economize the use of resources at the expense of performance.

So far, some different switching techniques for NoCs have been implemented in FPGAs. Exploration of the area, route latency, and route quality between time-multiplexed and packet switched network routers on FPGAs was done in [33]. A virtual-cut-through NoC router designed for FPGAs was proposed in [39]. Reference [32] proposed and demonstrated a lightweight circuit-switched approach for FPGA-based NoCs. It was discussed that their router is flexible enough to be used in general embedded systems and is high-performance enough for many high-throughput data flow applications.

Recently, Altera announced a new family of FPGAs based on Intel 14 nm Tri-Gate process technology, providing designers with upto a 10X increase in programmable

gates [25]. Xilinx announced a new 3D FPGA (Virtex-7 HT) with 6.8 billion transistors, providing designers with access to 2 million logic cells. This is equal to 20 million ASIC gates, which makes this FPGA ideal for system integration, ASIC replacement, and ASIC prototyping and emulation [28].

NoCs are supposed to be a perfect solution for the communication challenges of on-chip interconnections. FPGAs also offer many features such as latest process technology combined with new 3D packaging technology and wide range of memory and functional blocks. Hence, FPGAs can be perfect candidates for NoC designs.

Different designs and features of FPGAs affect the design metrics such as area, power and delay. Therefore it is important to choose a right set of NoC architectures as well as FPGA features that work best together.

2.6.4 NoC Evaluation Metrics

Area and power consumption are the most leading metrics of NoC cost [37]. The NoC designers' goal is to minimize them especially for small and mobile applications, where these resources are limited. More considerations about theses costs are required to accommodate on-chip interconnection network and SoCs into FPGAs.

Since FPGAs have fixed logic units and routing paths, their area and power consumption are relatively correlated. The area and power efficiency of a design usually depends on NoC design parameters and FPGA characteristics. Buffers are the most power and area hungry components among a router's parts. However, buffers are important to reduce the latency and to handle data flow problems. Therefore, buffer type and capacity as well as reduction of area and power of other router's components should be studied in order to acquire cost-efficient design.

There are many metrics to evaluate the speed of routers and topologies. Maximum operating frequency is one of the important factors that influences the speed of message delivery. Other metrics are throughput and latency which are defined respectively as the amount of data transferred over a period of time and the average time taken between sending the data at a source node and receiving it at a destination node. Usually latency is calculated as the average delay of a packet/flit traversing the NoC. The lower bound of average latency is called the best case latency or *zero load latency* which has no packet blocking in the NoC. Other forms of latency include the presence of blocking.

Bisection Bandwidth:

Bisection bandwidth is one of the metrics for a fair comparison between topologies and it is used as a proxy for cost metrics (Area and Power) [11]. By using equal bisection bandwidth, we try to equalize the cost of topologies.

A bisection of a network is defined as a cut that divides the network into two halves of nearly equal size. The bisection bandwidth of a network is defined as the minimum bandwidth over all bisections of the network [13]. For example, in a bus the bisection bandwidth is defined as the bandwidth of a line but for a ring it is bandwidth of two lines.

In this thesis 5 topologies are considered, so we define the bisection bandwidth for each of the topologies in the thesis. Figures 2.8 and 2.9 show the topologies used in this thesis. In these figures each edge denotes two unidirectional links going in opposite directions.

1. For a KxK 2D Mesh (Fig. 2.8a), the bisection bandwidth is proportional to the

bandwidth of 2K lines. There are 2 lines per row, and there are K rows.

2. For a KxK 2D Torus (Fig. 2.8b), the bisection bandwidth is proportional to the bandwidth of 4K lines. There are 4 lines per row, and there are K rows.

3. For a KxK 2D Binary Hypercube which has 2^n nodes (Fig. 2.8c), the bisection bandwidth is proportional to the bandwidth of K^2 lines [15]. There are K lines per row, and there are K rows.

4. For a KxK 2D Generalized Hypercube which has 2^n nodes (Fig. 2.8d), each node is connected to all other nodes in the same dimension, the bisection bandwidth is proportional to the bandwidth of $\frac{K^3}{2}$ lines. There are $\frac{K^2}{2}$ lines per row, and there are K rows.

5. For a KxK 2D Hypermesh which has 2^n nodes (Fig. 2.9), the bisection bandwidth is proportional to the bandwidth of K^2 lines. There are K lines per row, and there are K rows.

To have a fair comparison between the NoC topologies, each topology can be normalized to have an equal bisection bandwidth of O(N) bits/sec. In this thesis the datapath width of the topologies with N nodes are adjusted to have equal bisection bandwidth as follows: $W_{GHC} = \frac{2}{\sqrt{N}}.W_{Hypermesh} = \frac{2}{\sqrt{N}}.W_{BHC} = \frac{8}{N}.W_{Torus} = \frac{4}{N}.W_{Mesh}$.

2.7 Methodology

To design a system using an *Application Specific Integrated Circuit* (ASIC), an ASIC design methodology must be followed. First, the functional requirements and external interfaces of a system should be specified in a Design-Specification. Next, the

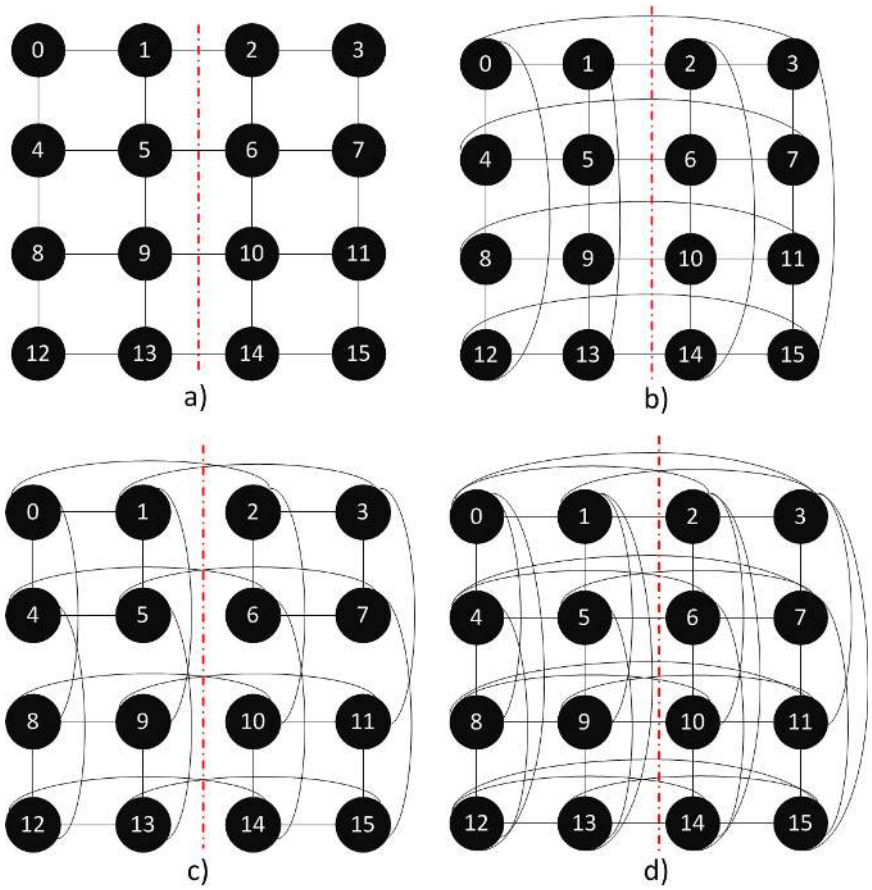


Figure 2.8: The bisection bandwidth in 4 topologies a) 2D Mesh, b) 2D Torus, 3) 2D BHC, 4) 2D GHC (each edge denotes two unidirectional links going in opposite direction)

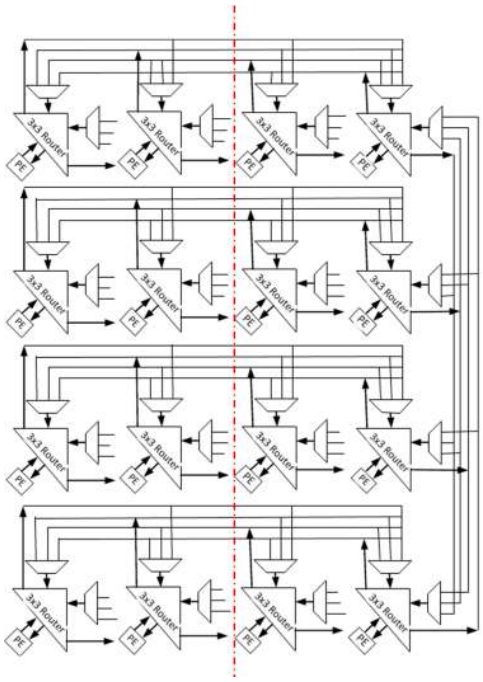


Figure 2.9: The bisection bandwidth in a 2D Hypermesh

system in the Design-Specification should be modelled using a *Hardware Description Language* (HDL). The task of developing a design in an HDL requires expert knowledge, and is very time consuming. Once the HDL description is developed, the functionality corresponding to the design specification should be verified. Once the functional verification is done, the HDL description is converted into an optimized gate level netlist using the synthesis tools. A synthesis tool requires several inputs: (a) the HDL description of the system, (b) a *Standard Cell Library* from an ASIC manufacturer, and (c) timing constraints. The synthesis tool then produces a gate-level netlist as output; equivalently it realizes the system using typically thousands (or millions) of standard cells selected from the Standard Cell Library, and it specifies the cells used and the interconnection of wires between these standard cells.

The synthesis step is a very important step in the ASIC design flow, as it ensures

that the realization of the design is done to achieve optimal or near-optimal results. Synthesis tools often use complex optimization engines, to achieve low area, low power and low delays. Once the synthesis is completed, the gate-level netlist with the timing information is used to perform a pre-layout static timing analysis. When the pre-layout timing model is verified, the synthesized design undergoes a process called *Place-and-Route*. In this process, the standard cells are placed onto a silicon substrate, and the wires are routed to realize the required interconnects. Call the output the *Physical Layout*. After the Place and Route process, a post-layout timing analysis is done on the Physical Layout. This step is similar to the pre-layout timing analysis, but it includes physical layout information as well. It verifies that the Physical Layout meets the Design Specification, after considering realistic wires delays and standard cell delays reported by the ASIC manufacturer in their Standard Cell Library. When the post-layout timing analysis is completed, the next step is to verify the post-layout logic functionality of the physical layout. In this step, the physical layout is simulated to ensure that the design has the correct functionality. When the design has completely passed the simulation, it proceeds to tape-out where it is eventually shipped to the ASIC manufacturer for fabrication.

It is occasionally discovered that a design cannot meet the requirements stated in the Design Specification, very late in the design process. The development of a detailed and functionally correct design in an HDL often takes several months of time, by a large design team. The steps of Place-and-Route, pre-layout static timing analysis, post-layout timing analysis and post-layout logic functionality verification can also take several months of time by a large design team. If it is discovered that a design cannot meet the design specifications at this point, then a significant amount

of time has been lost (several months, potentially up to a year), and significant sums of money has been lost (potentially millions of dollars in salaries). At this point, the company has likely lost the race to be 'first to market' with a design. It may continue to redesign the product, investing more time and money, or it may abandon the design and forfeit the market-share completely. Consider for example if Apple released the first iPhone 6 months after Samsung debuted a similar product. Apple would not the market leader that it currently is, affecting billions of dollars of market capitalization. For this reason, early design space exploration tools are very important. Early design space exploration tools can accurately estimate the area, delay and power used by a design, very early in the design process, before months of time and millions of dollars are invested.

Given a CMOS technology (i.e., a 40 nanometer or a 20 nanometer CMOS process, etc), the ASIC vendors perform extensive experimental results, to generate their Standard Cell Libraries. They fabricate several designs using their standard cells on a silicon ASIC, and they experimentally measure numerous data related to the performance. They typically develop a delay equation to curve-fit the experimentally-observed delay for each type of CMOS logic gate, with N fan-outs for variable N, and they publish this data in the *Standard Cell Library*. The libraries include the VLSI area used for each logic cell, the power used by the cell (at a given clock rate), and the delay of the cell depending on the fan-out load N.

CAD tools can use these proprietary *Standard Cell Libraries* to explore the design space in the early stage of the design process. CAD tools can estimate a logic circuit's delay, by analysing a network description as a collection of interconnected standard cells, and using the data reported in the *Standard Cell Library*. These *Standard*

Cell Libraries are typically available to universities and companies under an *NDA* (Non-Disclosure-Agreement). Unfortunately, university researchers typically cannot publish the data in a Standard Cell Library, or use this data in an analytic model, due to the NDA. In [55], publically available data not covered by an NDA for a 180 nanometer CMOS process was published, however this process is too old to be used today, where 20 nanometer technology is often used.

Unfortunately, there are no similar libraries of area, delay or power for logic cells used in an FPGA. A system designer using FPGAs does not have access to a library of data for various logic functions once realized on an FPGA, for use in early design space exploration. In this thesis, we developed a methodology to collect similar performance data for logic functions when realized in FPGAs, using only a few simple experiments on the FPGAs. We also develop analytic models which can predict the area, delay and power of a complex logic design, using only basic data collected from an FPGA.

In order to estimate delay and power of a wire, the length of the wire in an FPGA should be known. In this thesis, a method first introduced in [60] is used to define length of a wire in an FPGA.

Definition: One *'LE-Length'* represents the square-root of the area of the basic programmable resource in an FPGA, typically called the 'Logic Element' (or LE).

In this thesis, all wire lengths are expressed in multiples of the LE-length. We assume that an LE has unity aspect ratio as shown in figure 4.6, and we define the length of one LE in the x and y dimensions to be 1 LE-length. In the Altera Cyclone IV family of FPGAs, every logic array block (LAB) has 16 LEs. As shown in figure 2.10, we assume that each LAB is square with unity aspect ratio (with an array of 4 by 4 LEs).

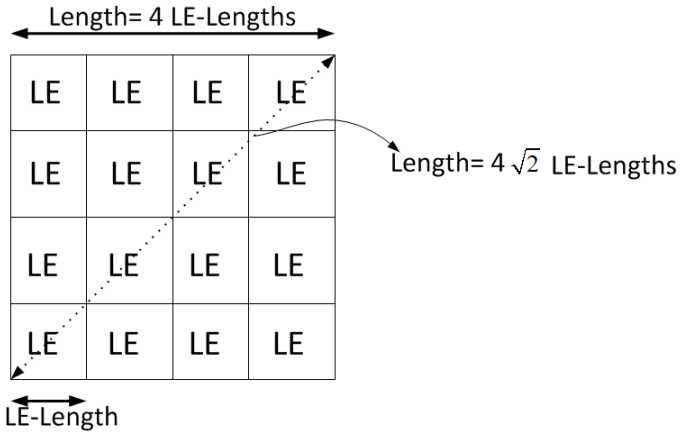


Figure 2.10: The LE-Length and the placement of 16 LEs in a LAB

We also define a new concept for distance in an NoC design using a regular array of identical nodes.

Definition: Given a NoC topology of N identical nodes that has undergone the Place-and-Route process in an FPGA, define the *'Node-Distance'* (ND) as the shortest distance between 2 adjacent nodes in the X or Y dimensions. The ND of an NoC with N nodes is denoted by $\Gamma_{Topology} = \sqrt{A_{Topology}}/\sqrt{N}$, where $A_{Topology}$ is the area of the NoC topology (excessed as a number of LEs used to realize the NoC in the FPGA). Figure 2.11 shows the ND in a 2D Mesh, expressed in terms of the LE-length.

ASICs and FPGAs use the same laws of physics, however there are some differences in their logic structures. For example in ASICs, each N-to-1 multiplexer can be constructed with $\frac{N-1}{m-1}$ smaller m-to-1 multiplexer standard cells arranged in a tree topology. Eq. 2.2 shows the area equation for an N-to-1 multiplexer with datapath width of W bits when realized in an ASIC.

$$A_{Mux}(N,1,W) = \left\lceil \frac{(N-1)}{m-1}.A_{mux(m,1)} \right\rceil .W \tag{2.2}$$

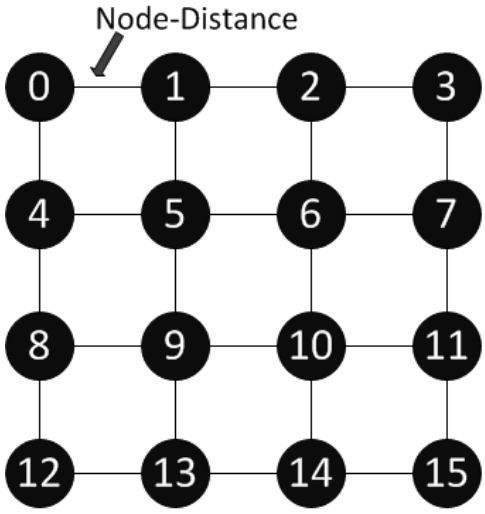


Figure 2.11: Node-Distance in a 2D Mesh

The term $A_{mux(m,1)}$ is the area of a m-to-1 mux standard cell, expressed in square nanometers, available in the ASIC Standard Cell Library.

In an FPGA environment, Eq. 2.2 needs to be modified. The 'area' of a logic design in an FPGA is defined as the number of LEs used to realize the design in the FPGA. The *synthesis efficiency* of a basic logic cell is defined as the average number of LEs used to synthesize the basic logic cell, given a particular FPGA family [60]. For example, Eq. 2.3 shows the area equation for an N-to-1 multiplexer with datapath width W in FPGAs:

$$A_{Mux}(N,1,W) = \left\lceil \frac{(N-1)}{m-1}.S_{Mux(m,1)} \right\rceil .W \tag{2.3}$$

where $S_{Mux(m,1)}$ is the synthesis efficiency for a m-to-1 Mux cell, and the value of m is determined for a FPGA device family. A few simple experiments are needed to find the value of m in a family of FPGAs. To find the synthesis efficiency for a multiplexer, several large multiplexers (i.e., 16-to-1, 32-to-1, 64-to-1) can be synthesized, and the

LEs used for each large multiplexer are recorded. The data is then curve-fitted to the preceding equation, to find the synthesis efficiency and parameter m.

If FPGA vendors would publish a library of useful data for common standard logic cells (i.e., the synthesis efficiency, the delay equation parameters, etc.), then FPGA designers could use this published data for early design-space exploration and optimization, just as ASIC designers can use published *Standard Cell Libraries* of data. Unfortunately, FPGA vendors do not currently publish such synthesis efficiency data, and in any case there have been no published analytic models to use such data. We show that FPGA designers can perform a few simple experiments and derive a small table of data (constants), to be used to model large logic designs in FPGAs, comparable to the *Standard Cell Libraries* published for the ASIC VLSI domain.

Note that each FPGA family (i.e., Altera or Xilinx FPGA families) has its own set of data constants (synthesis efficiencies and delay constants), just as each family of ASIC standard cells has its own Standard Cell Library.

2.7.1 Methodology for Validation of Results

To validate the analytic models for logic functions in FPGAs, many validation experiments are performed for different switch designs, input buffers, NoC topologies, and traffic patterns. Each NoC is designed modularly and implemented using VHDL hardware description language. Figure 2.12 shows the CAD flow for obtaining the validation results. As stated earlier, 2 VHDL simulators were implemented. The simulators run on a Intel Core 2 Duo system with 2GB RAM, running Windows.

The time required to acquire each design metric is different. The experimental area and delay reports can be found using the Quartus and TimeQuest analyzer tools.

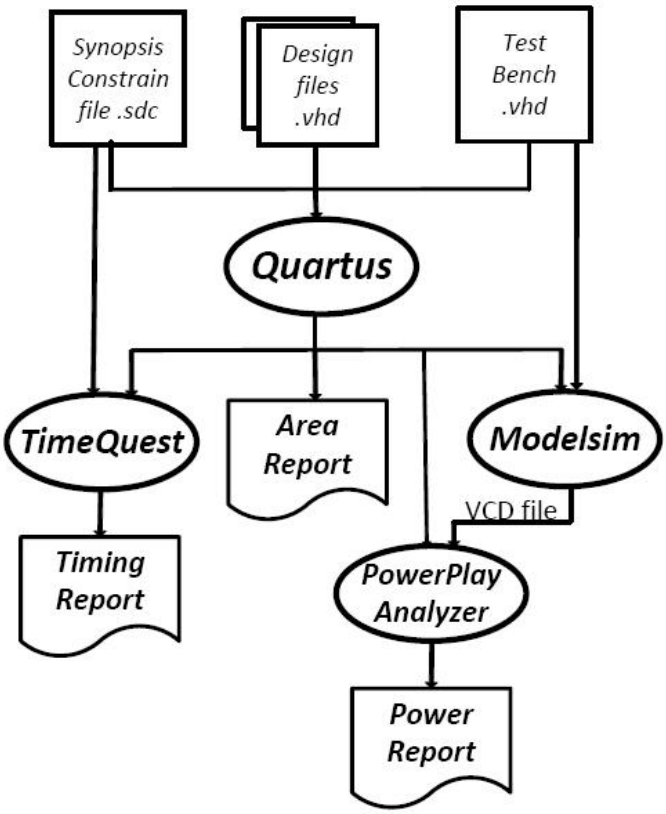


Figure 2.12: Experimental design flow with Altera FPGA CAD tools

The time required to find the experimental area and delay reports for a crossbar switch design is about 2 to 5 minutes of CPU time, depending the datapath width and switch degree. The compilation time to find the experimental area and delay of an NoC is about 15-20 minutes of CPU time. (The time required to find the analytic area or delay using the analytic methods developed in this thesis is virtually instantaneous.)

The most time consuming CAD report is the experimental power report. To determine the power consumption, a testbench associated to the HDL design is required to perform a post-synthesis simulation. The Modelsim tool performs a detailed gate-level simulation, and generates several *signal activity* files called *Value Change Dump* (VCD) files. The Altera *PowerPlay* analyser tool processes these VCD dump files, and

determines the experimental power used. To acquire accurate experimental power results, the simulation should be run for a long period of time (i.e., for several thousand packet transmissions in an NoC) using a suitable testbench. This simulation step can take several hours of CPU time, depending upon the complexity of the circuit. The time require to perform a gate-level simulation varies from 15-20 minutes for a small crossbar switch, or up to 5-10 hours for an 16 node NoC, depending on the router degree, NoC topology, and datapath width. (The time required to find the analytic power using the analytic methods developed in this thesis is virtually instantaneous.)

We performed the validation experiments for NoCs using the Batch-Mean method. Each simulation run is divided to 8 batches. To avoid systematic errors in the power measurement of the NoCs, it is necessary to ignore the warm-up batch since simulators have empty buffers and idle resources initially, before any packets are injected. The remaining batches can be processed to find the average power utilization (and the 95% confidence intervals is desired).

2.8 Summary

In this chapter, we have given an overview of the NoCs and FPGAs. We discussed different aspects of NoC design, such as network topology, switching techniques, routing algorithms, and flow control mechanisms. Moreover, a typical input-queued based router architecture for NoCs was illustrated and discussed. Then, we explained the FPGA's concepts as well as its advantages and disadvantages over ASICs. In addition, we explained why FPGAs are suitable platforms for future NoCs. At the end, we had a brief explanation of the evaluation metrics in NoC domains.

Chapter 3

Simulators

In order to explore the design space in FPGA-based NoCs, I implemented two parametrized VHDL simulators for five NoC topologies (2D Mesh, Torus, BHC, GHC, and Hypermesh). Both simulators use wormhole switching. The first simulator implements a 'Pseudo random walk' model and is designed to acquire experimental area, delay, and maximum power consumption of routers and NoCs. The second simulator implements traditional XY ordered-dimension routing to acquire the power consumption of different traffic patterns.

3.1 Related Works

To date, different NoC simulators have been introduced in the literature. These simulators are implemented with different programming languages such as C, C++, Java, C#, SystemC, Verilog and VHDL. Examples of these simulators are Booksim [47], Wormsim [48], Nirgam [50], Noxim [49], Atlas [51], Xmulator [52], ORION [41], etc.

Although these simulators are useful for research purposes, they do not satisfy the expectations of this thesis. Firstly, the simulator should have the ability to implement different topologies. Secondly, the simulator should be modular in order to be capable of evaluating every component of NoCs. Thirdly, it should be designed for FPGA environments. Fourthly, the expected simulator must use real circuit parameters of FGPAs. Due to these reasons, I was obligated to design and implement 2 NoC simulators such as the pseudo random walk and the wormwhole simulators.

3.2 Pseudo Random Walk Simulator

The pseudo random walk simulator is implemented in VHDL to find the area, maximum power consumption, and maximum operating frequency of NoCs. In this simulator, all the links in an NoC topology are filled by packets in every clock cycle. As a set of N packets leave and new set of N packets enter the NoC, the new packets are forwarded in all routers according to a newly generated pseudo-random switch state. In the pseudo random walk simulator packets are not blocked in the routers and every link is 100% loaded in every clock cycle. In this simulator we have maximum load for NoCs (See Appendix A).

As previously stated, an input-queued router includes input buffers, a crossbar switch, a routing unit, and an arbitration unit. The functionality of each component in the pseudo random walk simulator as well as packets format are described below.

3.2.1 Packet format

In order to forward packets in NoCs, packets are divided into small pieces called *flits*. Therefore the size of a packet can be expressed in terms of flits. In this simulator the packet size is 32 flits and packets have no routing information (i.e. source or destination address). The size of a flit equals to the data path width (which is 16 bits).

3.2.2 Packet Generator

In this simulator, the router injection channel is connected to a PE which generates packets with an injection rate of one flit per node per clock cycle. Each packet generator produces packets containing pseudo-random sequences by using a linear feedback shift register (LFSR) [46]. The initial value of this LFSR (seed) is a repeated stream of '1010'. The initial value is used to start the LFSR sequence at a certain position in the pseudo-random sequence. The implemented LFSR provides 32 different binary numbers and the size of these binary numbers is equal to the physical datapath width. The packet generator sends these binary numbers untill a full packet is sent.

3.2.3 Input Buffers

With every input port, an input buffer is used for temporarily storing the incoming flits. The input buffer works on the basis of the first in first out (FIFO) mechanism. This input buffer can be implemented with 2 methods, (i) with Embedded Memory (EM) blocks, or (ii) with DFF registers. In this simulator, input buffers have a capacity of 4 flits. At each clock cycle one flit is written into an input buffer and is read in the next clock cycle. Since there is no blocking in this simulator, the average

number of flits in an input buffer is 1. Note that in order to read and write in every clock cycle, the read and write request signals should be always active. Therefore the rate of the packet read and write is equal to the packet injection rate.

3.2.4 Arbitration Unit and Routing Unit

In order to move the packets through the switch, an arbitration unit is required. The arbitration unit generates arbitration signals to provide synchronous connections between input ports and output ports of the crossbar switch. In the pseudo random walk simulator, the arbitration signals are chosen from predefined states and these states are identical for all the switches in the network. These states ate explained in Appendix A. In an NxN crossbar switch, the arbitration signals always provide N connections for N input-output pairs. In other words, in an NxN crossbar switch, every input port is connected to a specific output port which results in 100% throughput for the crossbar switch. To see more detail, please see Appendix A.

3.2.5 Crossbar Switches

Different crossbar switches are used in this simulator. Figure 3.1 shows 4 types of crossbar switches. In every crossbar switch the selector signals are derived by the arbitration unit. The arbitration unit makes N connections for N input-output pairs possible therefore the crossbar switch in this simulator works with 100% throughput (To see more detail, please refer to Appendix A).

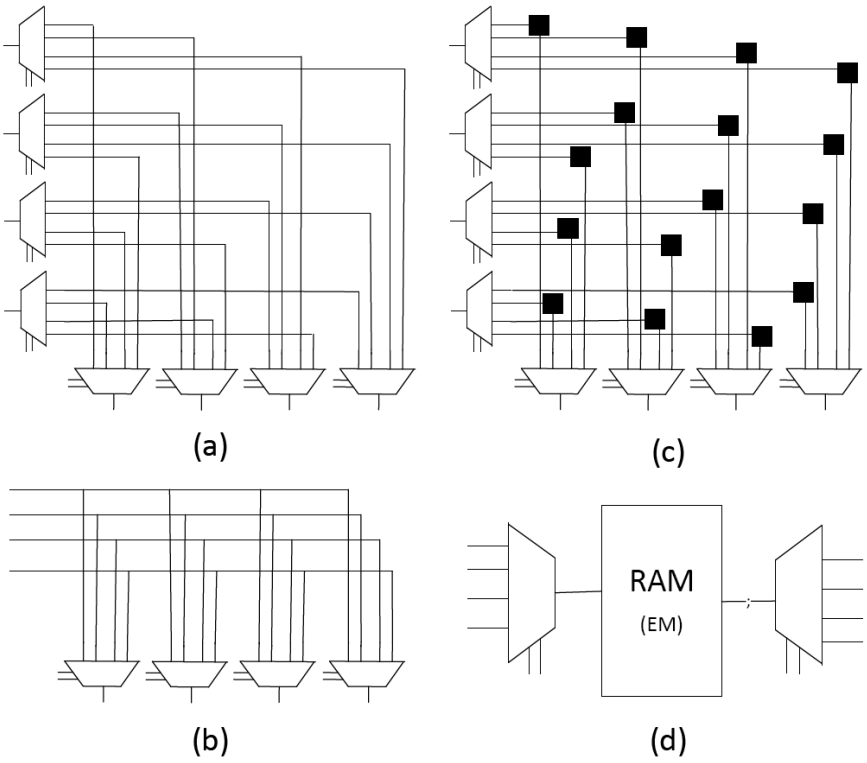


Figure 3.1: Crossbar switches, (a) DM,(b) B&S, (c) PDM, (d) Ram-Based

3.2.6 Simulation

In the pseudo random walk simulator, each packet generator generates a packet with 32 flits at time t=0. Each router initially implements a pseudo-randomly selected permutation of its IO ports for the duration of the packet transfer. As a set of N packets leave and new set of N packets enter the NoC, the new packets are forwarded in all routers according to a newly generated pseudo random switch state. (To see more detail, please refer to Appendix A). Each packet traverses pseudo random paths through the NoC topology, traversing many routers and edges, and exits at a pseudo-randomly selected router. At any one time, there are N packets active in the simulator, and all routers and links are 100% loaded. Figure 3.2 shows a snapshot of inter-router links in a 2D Mesh. As seen in this figure, all the links are 100% loaded.

3.3 Wormhole Simulator

A credit-based wormhole simulator was implemented in VHDL to route packets in the presence of packet blocking under different traffic patterns. In this section, the components of a router and the packet format in this simulator are explained. All the routing information is in the packet therefore we start from the packet format.

3.3.1 Packet Format

The main role of a packet-switched network is to transport packets from the source to the destination. A packet is the smallest logical unit of data that an IP core can inject into the network. As stated earlier, in the wormhole switching, a packet is broken down into multiple flits. In this simulator a flit is the smallest physical unit

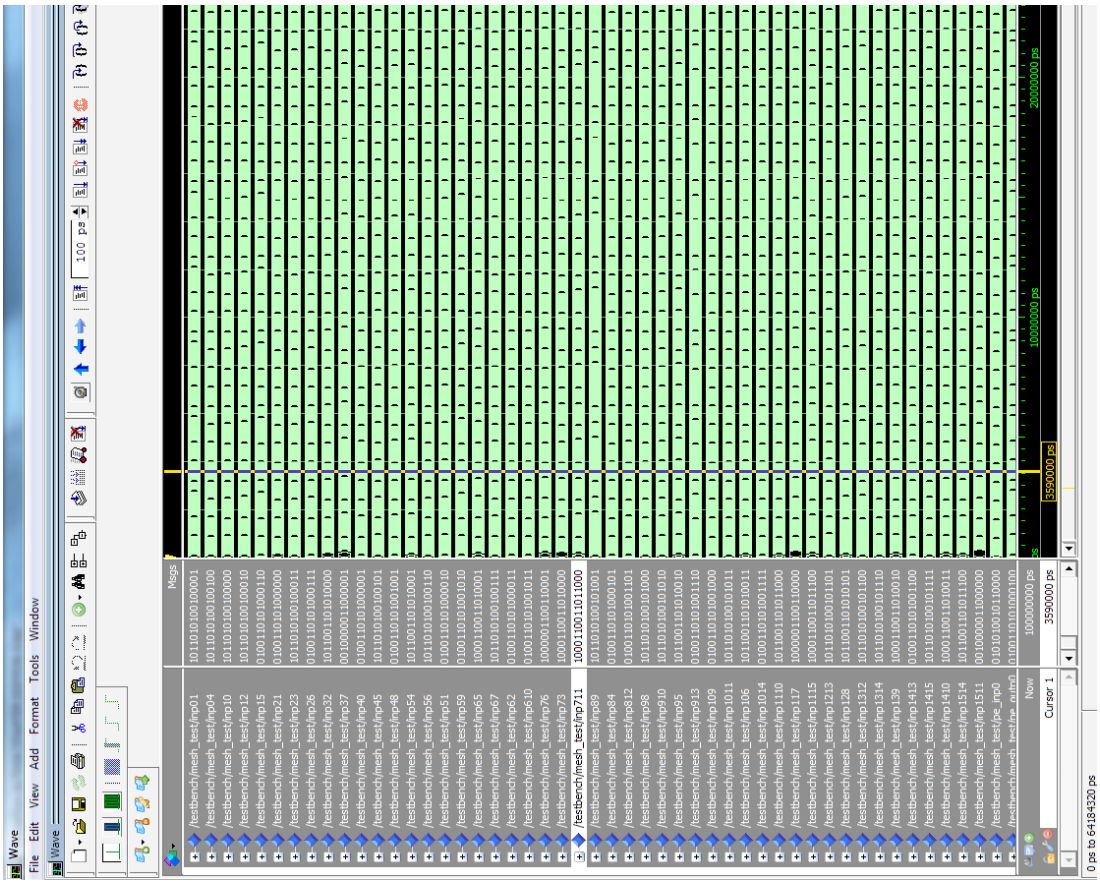


Figure 3.2: Snapshot of the simulation of the inter-router links in a 2D Mesh (all the links are 100% loaded)

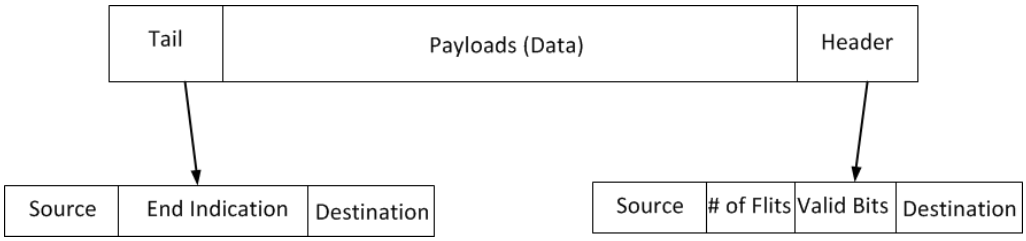


Figure 3.3: Packet format

of data that is routed in the network. The first flit of the packet is called *header flit*. It contains the routing information (i.e. source address, destination address, etc.) of the packet. The last flit of the packet is called *tail flit* which indicates the end of packet transmission. The rest of the flits contain the actual data of the packet which are called *payloads*.

Figure 3.3 shows the structure of a packet in the wormhole simulator.

3.3.2 Packet Generator

In this simulator, the router injector channel (as shown in figure 2.5) is also connected to a packet generator which has the capability of generating packets with different injection rates. Each packet generator uses an LFSR to produce pseudo-random sequences of data. The format of LFSR is same as the LFSR explained in the pseudo random walk simulator.

The packet generator is responsible to form the packets in the required format. It should generate packets for different traffic patterns and put all the routing information in each packet. It is also responsible to generate the packets following a Poisson distribution. We use the LFSR to provide the Poisson process behaviour.

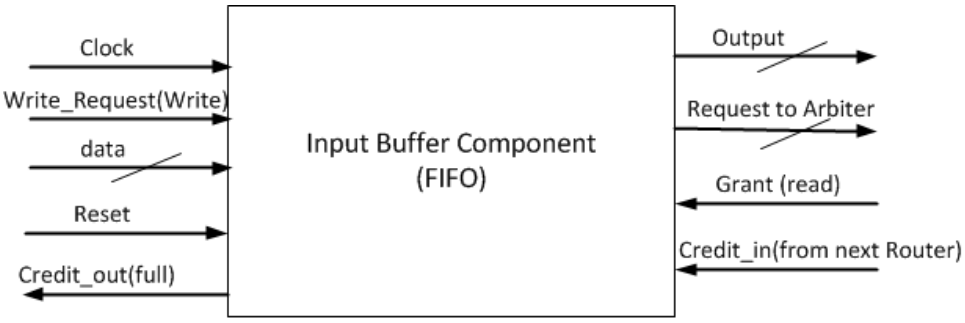


Figure 3.4: Architecture of the Input Buffer Component

3.3.3 Input buffer

As shown in figure 2.5, each input port is connected to an input buffer to store the incoming flits. The input buffer works on the basis of a FIFO mechanism. In this simulator, the input buffer component includes 2 parts. One part handles the control signals (i.e. request, grant, back-pressure, etc) and the other, which is a FIFO buffer, is responsible for storing the incoming packets. The second part can be Embedded memory blocks or DFF registers. Figure 3.4 shows the picture of an input buffer component.

As soon as the header flit arrives at the input port, the 'Write Request' signal is set to 1 in order to store the incoming flit in the FIFO buffer. Once the header flit is written in the first location of the FIFO, the buffer is read and a request signal including the routing information is sent to the router *Arbitration unit*. Once the request gets a 'Grant', the buffer is read. The reading operation stops when the control signal 'Credit in' coming from the *Arbitration unit* becomes invalid. This invalidity of 'Credit in' signal indicates that there is no more space in the adjacent router to store the flits. However the current buffer will continue to fetch new flits until it becomes full. The control signal 'Credit out' is 'Credit in' signal for the

previous router. Both 'Credit in' and 'Credit out' signals are used for the back-pressure technique.

3.3.4 Arbitration unit

In order to move the packets through the switch, an arbitration unit is used in every router. The arbitration unit generates arbitration signals in a round-robin scheme to provide synchronous connections between every pair of the crossbar switch input and output ports. Once it receives a request from an input port, it sends out the routing information to the routing unit and receives the direction information. Then the availability of free buffer locations in the neighbouring destination router is checked by examining the validity of the signal 'Credit in'. If it is available, it generates a 'Grant' signal to the requested input buffer component. It also generates a 'Write request' signal to the input port of next router. Figure 3.5 shows the diagram of the arbitration unit.

3.3.5 Routing Unit

In order to find the destination output port for a requested input buffer, a routing unit is needed (as shown in figure 2.5). This unit is responsible for obtaining the direction that a packet should take based on the address provided by the header flit. The routing unit receives the routing information from the arbitration unit. Once it receives the routing information, it decodes the destination address and finds the associated output port. Finally the routing unit sends the results back to the arbitration unit. In our simulator, we used the ordered-dimension XY routing algorithm.

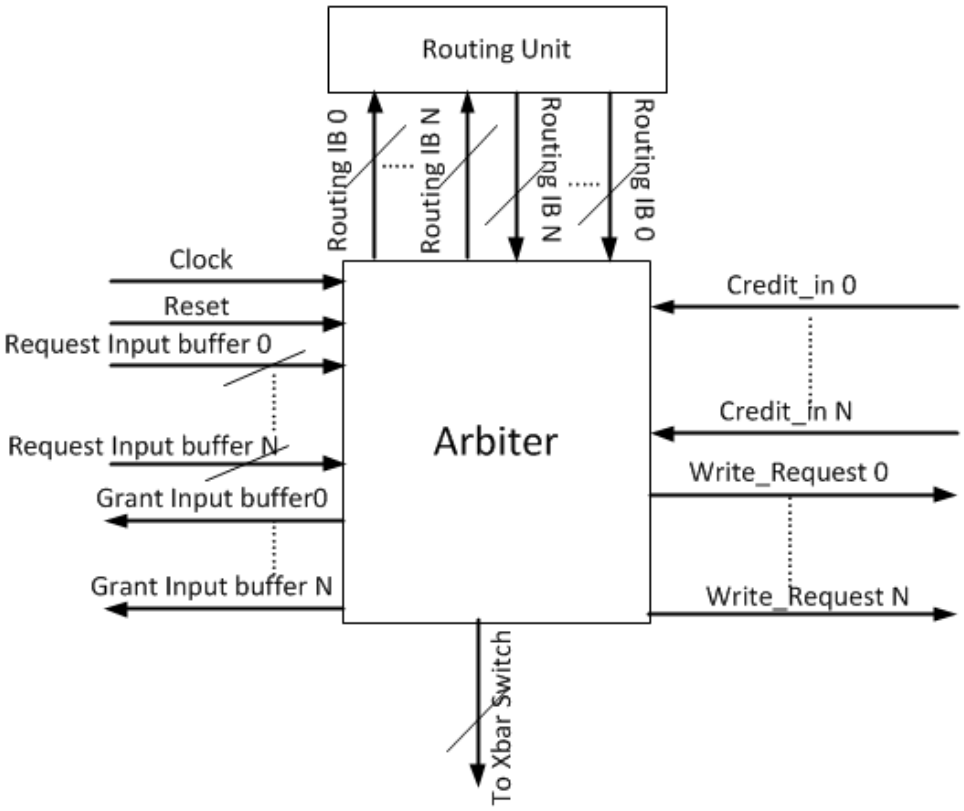


Figure 3.5: Architecture of the Arbitration unit

3.3.6 Crossbar Switch

The crossbar switch is the final stage of the router. It maps the packets coming from the input ports to the assigned output ports. In the wormhole simulator, the Broadcast-and-Select switch design (shown in figure 3.1b) is used.

3.3.7 Communication Between Routers

The most popular scenarios of flow control are handshaking (ACK/NACK) and credit-based protocols. We use a credit-based protocol for the communication between the routers due to its high efficiency. It consumes only one clock cycle to forward a flit from one node to another. Moreover it requires only a one-bit credit signal which is called 'Credit in' or 'Credit out' in our simulator. However, the handshaking protocol requires more clock cycles (at least two) and more wiring (i.e. Request and Acknowledge) for each channel.

3.4 Simulation Software Settings

To implement our simulator, we used the Altera Quartus II V.11 FPGA design tool. The experimental power was determined using the *ModelSim* simulation tool, and the Altera *PowerPlay* tool. Modelsim performs detailed circuit-level simulations, and generates *signal activity* files called *Value Change Dump* (VCD) files. The VCD file is fed into the Altera's PowerPlay tool to determine the power of each part of the router. The power consumption reported by the Altera's PowerPlay tool includes the power dissipated in the each component and the interconnections (switch boxes and tracks) which are driven by the component. The operating conditions for the power

estimation were standard (25° C ambient and board level).

The optimization technique was set to 'balance', which tries to have high performance with minimal logic usage. The fitter settings was also set to the 'standard fit' which maximizes the maximum frequency of the design. *Synopsys Design Constraints* (SDC) and TimeQuest timing analyzer were also used for the static timing analysis. The TimeQuest timing analyzer reports the interconnect and logic gate delays for the critical path. The timing delays (F_{MAX}) reported in this thesis are under the slow 0°C timing model. F_{MAX} is reported regardless of the user-specified clock periods and indicates the maximum allowable frequency of a design. The SDC file is also used in the PowerPlay analyzer to determine accurate power results.

An overview of the experimental flow and the design tools used in this paper is summarized in figure 3.6. The followings also are the settings for the Quartus II CAD tool:

1. FPGA Device: Altera Cyclone IV EP4C75F29C6
2. Analysis and Synthesis Settings:
   (a) Optimization Technique : Balance (Speed/Area)
   (b) Fitter Effort : Standard Fit (Highest effort)
   (c) Simulator : Modelsim-Altera
   (d) Simulation Mode : Gate-level simulation
3. PowerPlay Analyzer Settings:
   (a) Input file : Signal activity file (VCD) generated by Modelsim
   (b) Device power characteristics: Typical

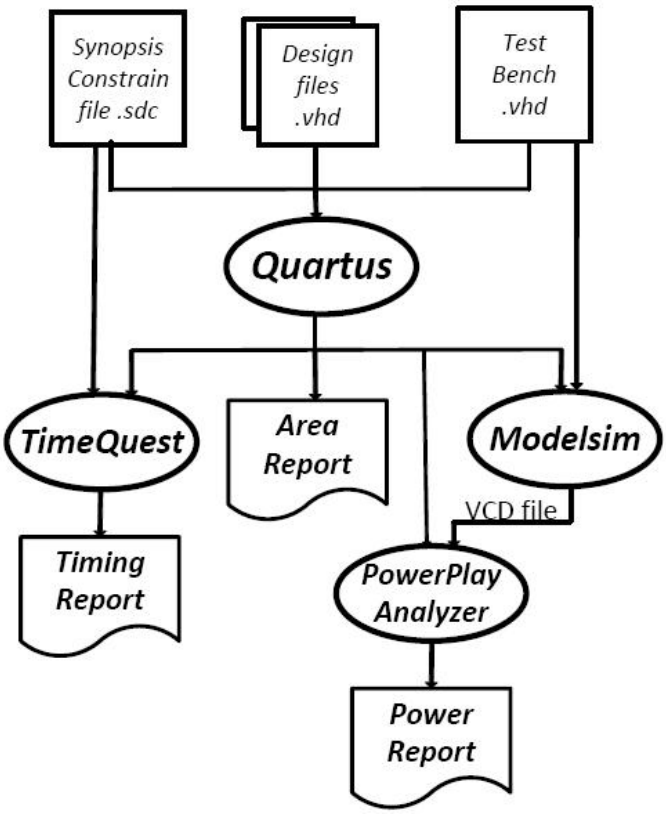


Figure 3.6: Experimental flow with CAD tools

(c) Ambient temperature :25 °C

(d) Junction-to-case:2.2 °/CW

(e) Case-to-heat sink : 0.10 °/CW

(f) Heat sink-to-ambient : 2.80 °/CW

(g) Board thermal model : Typical

(h) Junction-to-board : 3.80 °/CW

(i) Board temperature : 25 °C

(j) Default toggle rate used for input I/O signals: 12.5%

(k) Default toggle rate used for remaining signals: 12.5%

In order to tune our analytic models for Altera Family of FPGAs, many experiments are used in order to determine the equations coefficients in chapter 4. Table 3.1 depicts the parameters and associated values for the Altera Cyclone IV FPGA. These parameters will be defined in later chapters.

3.4.1 PowerPlay Analyzer

PowerPlay analyzer [26] estimates the power consumption of a design by considering the followings:

1. Environmental Conditions
2. Resource Usage
3. Signal Activities

Table 3.1: List of Parameters and associated values for the Altera Cyclone IV FPGA

Parameter	Value
m	2
$S_{Mux(2,1,1)}$	0.666
f_{clock}	50 MHz
$K_{Mux(2,1,1)}$	1.96E-4 (mW/MHz)
$S_{Dmux(1,2,1)}$	1.1
$K_{Dmux(1,2,1)}$	3.36E-05 (mW/MHz)
K_{wire}	4.62E-06 (mW/MHz)
Z_{std}	3.56E-05 (mW/MHz)
K_{DFF}	0.1825E-03 (mW/MHz)
K_{IBEM}	2.7E-02 (mW/MHz)
K_{IBDFF}	0.1825E-03 (mW/MHz)
$F_{Mux}(4,1,16)$	765 MHz
$F_{Dmux}(1,4,16)$	1285 MHz
F_{EM}	405 MHz

To evaluate different designs, they should be tested under the same environmental conditions. Therefore, environmental conditions must be same for all the evaluations.

Signal activities or toggling is another important factor that impacts the power consumption of the design and is used for the power evaluation. Power usually increases linearly with the toggle rate as the capacitive load is charged more frequently for the logic and routing [26]. Therefore in order to have an accurate comparison, different designs should have the same input data.

Different designs use different resources. A design that uses more basic units of the FPGAs (i.e. LEs, ALMs, Slices, multiplier elements, and memory blocks) tends to consume more power than a design with fewer elements.

The Powerplay analyzer reports 2 types of power [26]:

1. Static Power

2. Dynamic Power

Static power is the power consumption of a device regardless of the design activity. Dynamic power is the additional power consumption of a design due to signal activity or toggling. Dynamic power increases linearly with the toggle rate as the capacitive load are more frequently charged for the logic and routing [26]. The dynamic power is the dominant power consumption [38]. As the technology scales down, clock frequencies are increased and therefore, the percentage of dynamic power increases. We always refer to power consumption of a design as the *Dynamic Power*.

In this thesis, we are interested in acquiring the dynamic power consumption of the components in the interconnection networks of NoCs (crossbar switches, buffers, and links). Dynamic power consumption reported by Quartus includes the power consumed by the elements (i.e combinational logic, DFF registers, multiplier elements, and memory blocks) and power consumed by the routings (switch boxes and tracks). The power consumed by local clock distribution within elements (i.e combinational logics, DFF registers, multiplier elements, and memory blocks) are included in the power consumption of the elements [27].

The PowerPlay analyzer also reports the power dissipation of the clock control blocks and I/O buffers at the pins of FPGAs. A clock control block is a clock buffer that dynamically enables or disables the clock networks [27]. The power consumed by the I/O buffers at the pins of FPGAs includes both the static and dynamic powers. Since the clock control blocks and I/O buffers at the pins of FPGAs are not the components of NoCs, we ignore them in the analysis.

Figures 3.7 and 3.8 illustrate 2 screen snapshots of the reported power by the PowerPlay analyzer for a 2D Mesh with 16 nodes. In figure 3.7 embedded memory blocks are used as input buffers while in figure 3.8 DFF registers are used.

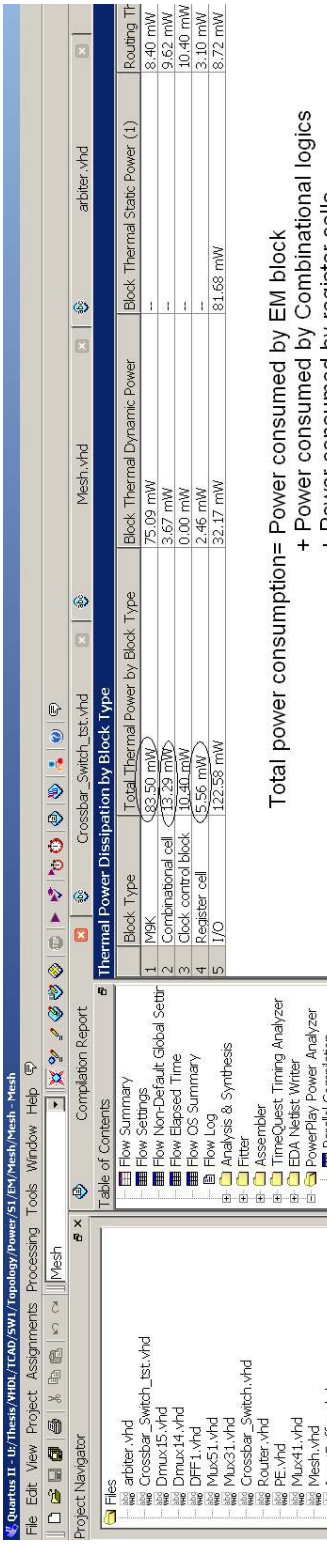


Figure 3.7: Screen Snapshot of 2D Mesh with 16 nodes using EM blocks as input buffers

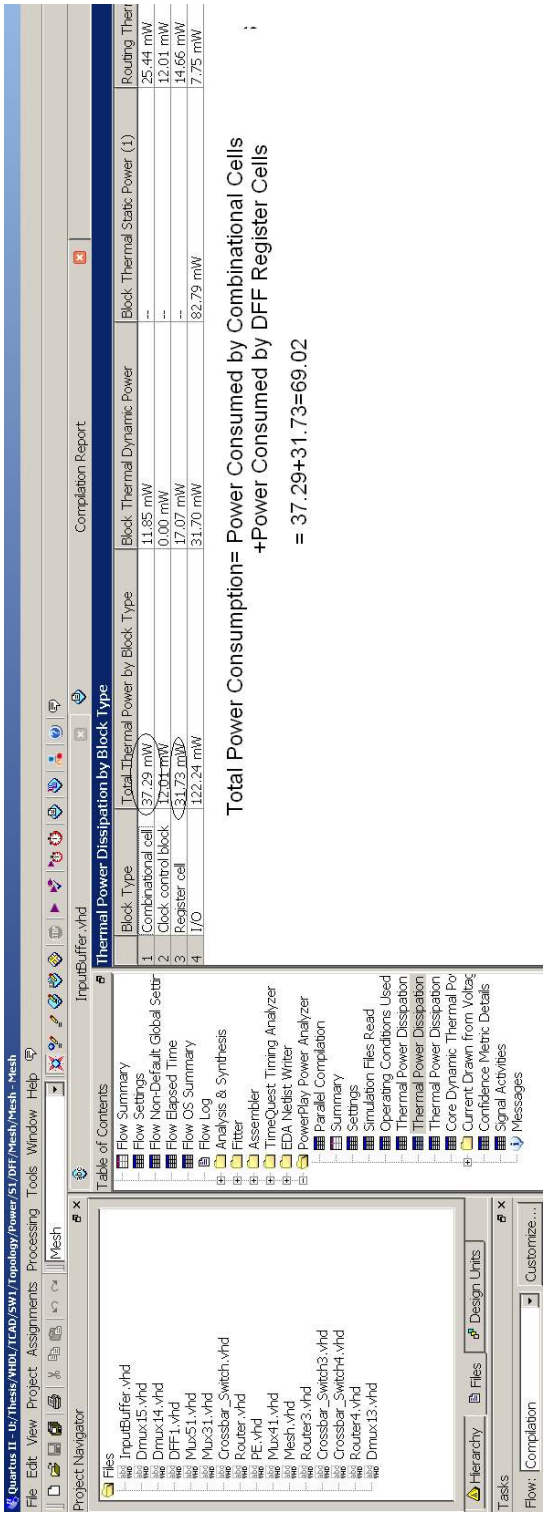


Figure 3.8: Screen Snapshot of 2D Mesh with 16 nodes using DFF registers as input buffers

When an NoC uses DFFs as input buffers, the total dynamic power is equal to the power consumed by the combinational cells plus the power consumed by the register cells. When the design uses embedded memory blocks, the total dynamic power equals to the summation of power consumed by combinational cells, register cells and embedded memory blocks.

3.5 Summary

In this chapter, two different simulators called the Pseudo random walk and the Wormhole simulators, were presented. The experimental flow, different tools, and settings for the simulation software were explained. Moreover, an overview of the Altera PowerPlay analyzer tool and how to estimate power using this tool was presented.

Chapter 4

Analytic Model of Basic Components of NoC Routers

As stated in chapter 2, in an FPGA, logic functions are configured through a series of programmable resources. These programmable resources are called *Logic Elements (LEs)*, *Adaptive Logic Modules (ALMs)*, or *Slices*, and are typically the basic unit of FPGAs. Each of these basic units consists of one or more look-up tables (LUT) and DFF registers.

The basic components of an NoC router in FPGAs are Multiplexers (Mux), Demultiplexers (Demux), wires, broadcast buses, and input buffers (as shown in figures 2.5 and 3.1). In this chapter, (1) the analytic area, delay, and power models for the basic components of NoCs, (2) analytic models for 7 types of crossbar switches, (3) and a comparison between analytic and experimental results are presented. At the end, a comparison of area per throughput and power per throughput of the switches is presented.

4.1 Related Works

Architectural-level estimation of area, delay, and power are suitable for design-space exploration because it is much faster than low-level simulation, and still gives us reasonable accuracy.

In [55], an analytic methodology was proposed for the area and power of a IQ-switch using a Broadcast-and-Select architecture, using 180 nm CMOS VLSI. The authors also proposed an approach to minimize the power dissipation in the Broadcast-and-Select switches. The power analysis of Input-Queued and CIXQ crossbar switches implemented in an FPGA technology was proposed in [60]. In [60], Broadcast-and-Select, pipelined and unpipelined Demux-Mux crossbar switch designs were considered.

An analytical power model to explore different switch configurations was proposed in [40]. Their model estimates the average and maximum power of the Alpha 21364 router and the IBM Infini-Band switch. According to their experimental results, input buffers are the largest power hungry component in routers, followed by the crossbar switch. At the end, the arbiter consumes negligible power.

Reference [45] analyzed the power consumption of the switch fabric in network routers and proposed the bit-energy model to estimate the power consumption. This model is tightly dependent on circuit implementations.

Reference [44] proposed a high-level power model based on the number of flits passing through a router, and used parametric regression to derive the model.

There are some power and area libraries, such as ORION 2.0 [41], Xpipes [42], etc., which are based on certain architectures and circuit implementations. However, using these libraries cannot guarantee an acceptable accuracy for evaluation of different NoC

designs and there might be upto 40% error [43]. This much of inaccuracy can lead to erroneous NoC design choices. In this chapter, we propose an analytic power, area and delay model for the FPGA-based NoCs' components which is accurate enough to predict and evaluate the design metrics of the components.

4.2 Basic Components

In this section analytic area, power and delay models of the basic components of NoC routers such as Mux, Demux, wires, broadcast buses, and input buffers are presented. The power model presented in this chapter reflects the maximum power consumption of each component when it is 100% loaded.

4.2.1 Mux and Demux

In ASICs, each N-to-1 multiplexer tree can be constructed with $\frac{N-1}{m-1}$ smaller m-to-1 multiplexer cells arranged in a tree topology [55]. However in FPGAs, the area of an N-to-1 Mux tree with a datapath width of W bits (expressed as the number of LEs, ALMs, or Slices) is given by Eq. 4.1 [60] (we changed the equation slightly to be more accurate), where the value of m is determined by the synthesizer compiler and FPGA device family.

$$A_{Mux}(N,1,W) = \left\lceil \frac{(N-1)}{m-1}.S_{Mux(m,1,1)} \right\rceil .W \tag{4.1}$$

In Eq. 4.1, the parameter $S_{Mux(m,1,1)}$ is defined as the *Synthesis Efficiency* of a basic m-to-1 multiplexer cell, i.e., it equals the average number of LEs, ALMs, or Slices used for the synthesis of a basic m-to-1 logic cell, given a particular FPGA

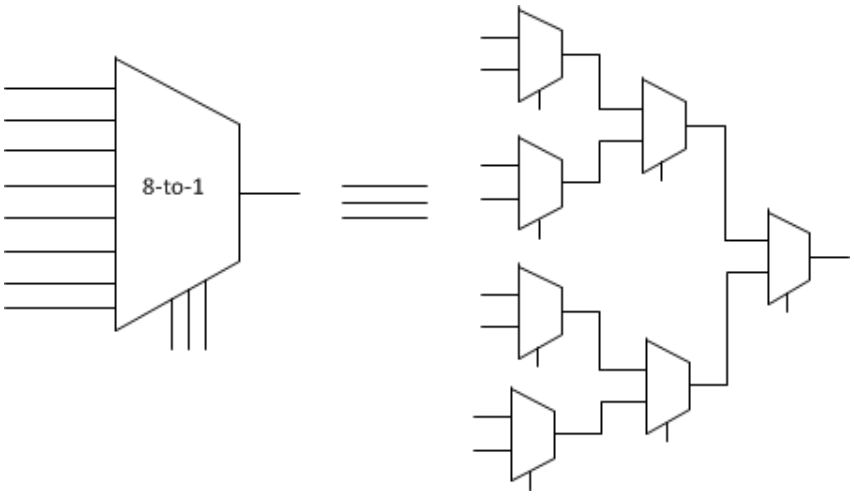


Figure 4.1: Area Model for an 8-to-1 Mux

family [60]. The area model for a 8-to-1 Mux tree is shown in figure 4.1.

In the Altera Cyclone family of FPGAs (which use 4-input LUTs), experimental results indicate that *m=2*, while in the Altera Stratix family of FPGAs (which use Adaptive-LUTs), experimental results indicate that *m=4*.

The delay of a Mux(N,1,W) tree is given by the delay of a basic m-to-1 Mux cell times the depth of the large Mux-tree, and is given by Eq. 4.2:

$$D_{Mux}(N, 1, W) = \log_m N . D_{Mux(m,1,W)} \tag{4.2}$$

where $D_{Mux}(m, 1, W)$ is defined as the delay of a m-to-1 Mux cell. The delay model of a 8-to-1 Mux is shown in figure 4.2.

In general, the power of typical standard cell is given by Eq. 4.3, where f_{clk} is the clock rate, f_{duty} is the probability of a signal transition at the output port, C_{int} is the intrinsic capacitance of the cell switched during an output transition, and C_{load} is the

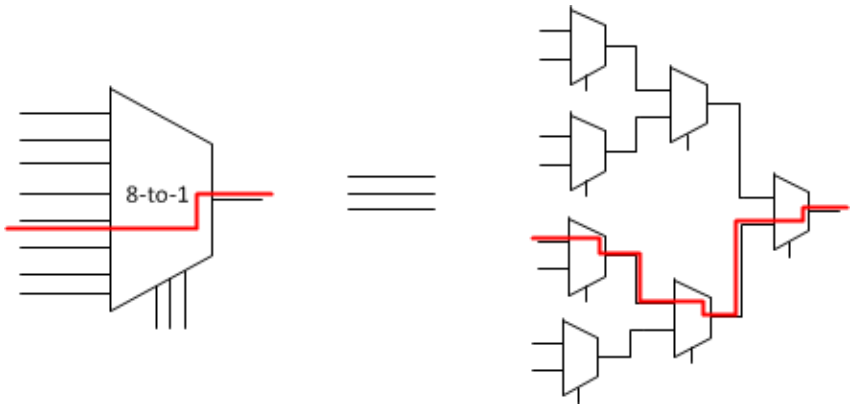


Figure 4.2: Delay Model in an Mux 8-to-1

capacitive load being driven (including wire and input capacitances).

$$P(stdcell) = \frac{1}{2}.f_{clk}.f_{duty}.(C_{int} + C_{load}).(V_{dd})^2 \tag{4.3}$$

The power consumption of a N-to-1 Mux tree, with a datapath width of W bits, is derived by Eq. 4.3 and evaluated by 2 models: a) full power model and b) low power model [55]. In the full power model all smaller Mux cells are active at the same time while in low power model, only $\log_m N$ smaller m-to-1 Mux cells need to be active and unnecessary Mux cells are switched off. The power consumption of a large N-to-1 Mux tree is proportional to the number of active cells. Therefore the low power model considerably reduces the power [60]. However, a circuit must be designed so that the data signals at each inactive input port can be held constant [60].

The power consumption of a N-to-1 Mux tree in the full power model, with a datapath width of W bits, is derived by Eq. 4.3 and can be expressed by Eq. 4.4 [60]:

$$P_{Mux}(N, 1, W) = \lceil \frac{(N-1)}{m-1}.S_{Mux(m,1,1)} \rceil .K_{Mux(m,1)}.f_{clk}.f_{duty}.W \tag{4.4}$$

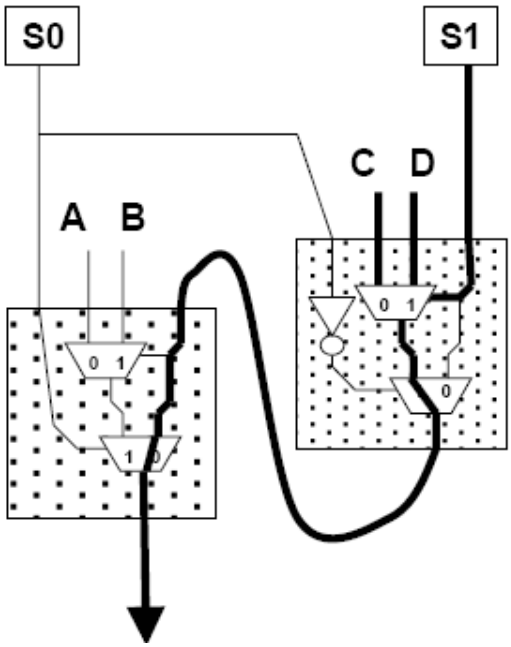


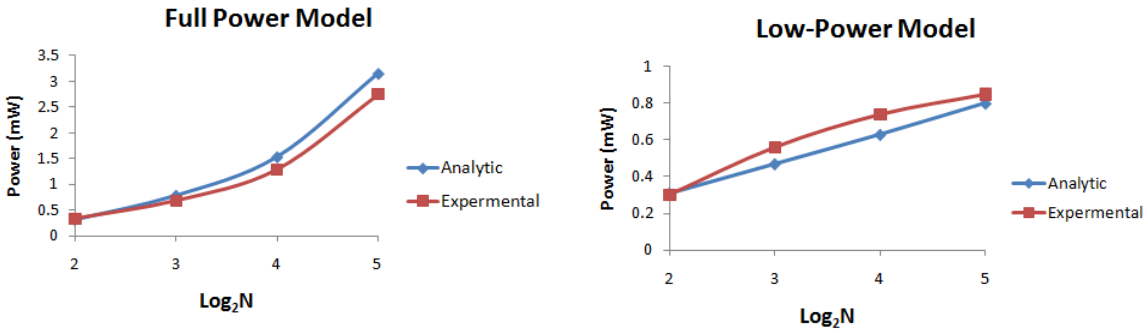
Figure 4.3: Implementation of a 4-to-1 Mux in Cyclone family of Altera FPGAs [87]

where $K_{Mux(m,1)}$ reflects the average intrinsic and load capacitances switched for a m-to-1 Mux logic cell in the Eq. 4.3. In this model, the $(1/2)V_{dd}^2$ term has been absorbed into the parameter $K_{Mux(m,1)}$.

In the low power model [55], a large N-to-1 Mux tree has only $\log_m N$ active cells and the power model is explained in [60] (we changed the equation slightly to be more theoretically justifiable), and is given by:

$$P_{Mux}(N, 1, W) = \lceil \log_m N \rceil . K_{Mux(m,1)} . f_{clk} . f_{duty} . W \tag{4.5}$$

In the Altera Cyclone family of FPGAs where *m=2*, a 4-to-1 Mux cell is constructed by two 4-input LUTs [87] (shown in figure 4.3). This technique of implementation of 4-to-1 Mux cell reduces the number of Lookup Tables (LUTs) needed to implement multiplexers [87]. In other words, instead of using three 4-input LUTs, two



(a) (b)

Figure 4.4: Analytic and experimental power consumption of N-to-1 Muxes in (a) Full Power Model, (b) Low Power Model (W=16).

4-input LUTs are used. Therefore, the synthesis efficiency of a 2-to-1 Mux ($S_{Mux(2,1)}$) in the Altera Cyclone family of FPGAs is $\frac{2}{3} = 0.666$.

As seen in figure 4.3, if any input port gets data, both LUTs are switched on. For large N the low power model significantly reduces the power consumption of N-to-1 Mux tree.

In the Altera Cyclone family of FPGAs where *m=2*, the $K_{Mux(2,1)}$ is experimentally determined as the average of $K_{Mux(2,1)}$ in the full power model and the low power model. However using 2 different $K_{Mux(2,1)}$ leads to the better results.

Figures 4.4a and 4.4b depict the analytic and experimental power consumptions of N-to-1 Muxes in the full power model and the low power model. In these 2 figures, the datapath width is 16 bits and f_{clk}=50 MHz. By comparing these 2 figures, the low power model of Muxes reduces the power consumption of large N-to-1 Muxes considerably.

Using the same methodology, it is possible to construct a large 1-to-N Demux tree using a binary tree of smaller 1-to-m Demux cells. Hence, the area, delay, and power

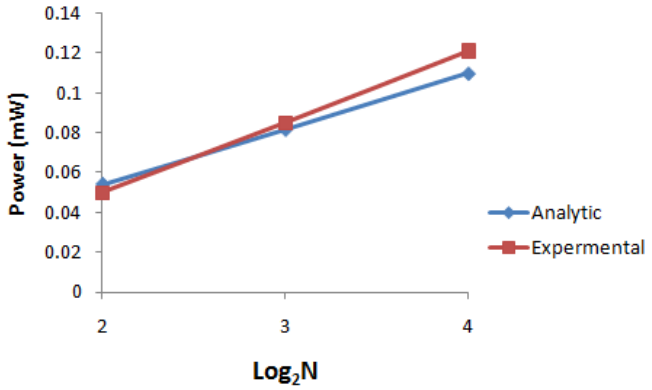


Figure 4.5: Analytic and experimental power consumption in 1-to-N Demuxes (W=16)

of a 1-to-N Demux tree (with datapath width = W) can be found by Eq. 4.6 [60], Eq. 4.7, and Eq. 4.8 respectively.

$$A_{Dmux}(1,N,W) = \left\lceil \frac{(N-1)}{m-1}.S_{Dmux(1,m,1)} \right\rceil .W \tag{4.6}$$

$$D_{Dmux}(1,N,W) = \log_m N.D_{Dmux(1,m,W)} \tag{4.7}$$

$$P_{Dmux}(1,N,W) = \lceil \log_m N \rceil .K_{Dmux(1,m)}.f_{clk}.f_{duty}.W \tag{4.8}$$

where $K_{Dmux(1,m)}$ reflects the average intrinsic and load capacitances switched for a 1-to-m Demux logic cell in the Eq. 4.3.

Figure 4.5 shows the analytic and experimental power consumption in 1-to-N Demuxes in the Altera Cyclone IV family of FPGAs. In this figure, the datapath width is 16 bits and f_{clk}=50 MHz.

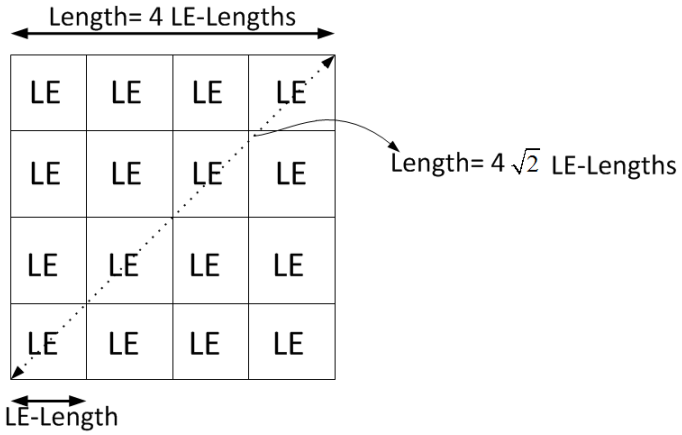


Figure 4.6: The LE-Length and the placement of 16 LEs in a LAB

From now on, we refer to LEs as the basic unit of programmable logic for the Cyclone family of Altera FPGAs and *m=2*.

4.2.2 Broadcast buses and links

The programmable interconnections in FPGAs use wire segments with different lengths. Long distance wires are constructed by aggregating multiple short wire segments [64], and they use repeaters to increase the signal strength and lower delays. The delay of long wire in an FPGA is linearly related to the length of the wire [64] [65].

Definition: One *'LE-Length'* represents the square-root of the area of the basic programmable resource, the 'Logic Element' (or LE).

In this thesis, all wire lengths are expressed in multiples of the LE-length. We assume that an LE has unity aspect ratio as shown in figure 4.6, and we define the length of one LE in the x and y dimensions to be 1 LE-length. In the Altera Cyclone IV FPGAs, every logic array block (LAB) has 16 LEs. As shown in figure 4.6, we assume that each LAB is square with unity aspect ratio (4 by 4).

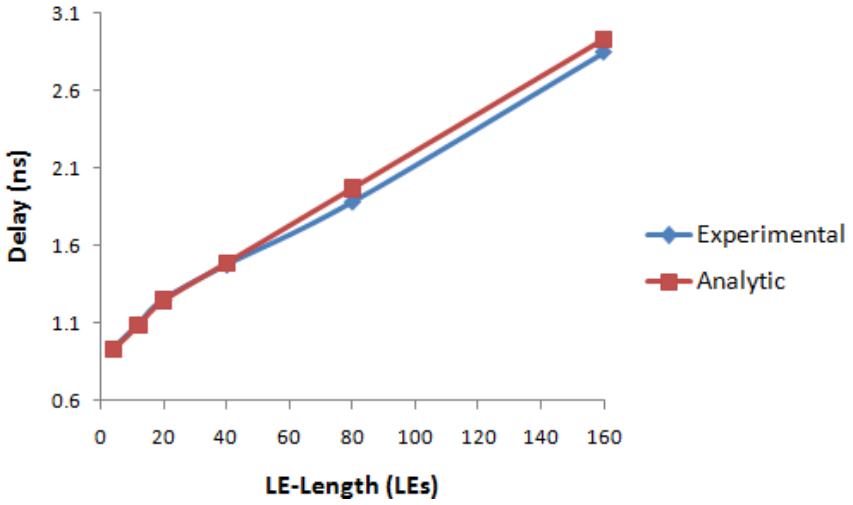


Figure 4.7: Experimental and analytic delay of a wire with different LE-lengths.

According to our experimental tests, the delay of a short bus is $O(logL)$, and the delay of a long bus is $O(L)$. Let $D_{bus}(L)$ be the delay of a bus of length L connecting two LEs in an FPGA. Referring to figure 4.7, at X=20 LE-Length, the slope of delay line changes. To fix this issue, we use an accurate 2 piece-wise linear approximation for the bus delay. The delay of a bus with W parallel wires and length L is modelled as (1) delay of a short wire when $4 \leq L \leq 20$ LE-Length, and (2) delay of a long wire when $20 \leq L$. The lower bound of 4 is due to the minimum distance between 2 logic array blocks (LAB) which is the minimum length of wire that is possible to measure.

The delay of a bus of length L is given by (in nanoseconds):

$$D_{bus}(L) = a.(L) + b \tag{4.9}$$

where for $4 \leq L \leq 20$ the parameters a=1.98E-2, b=0.85, and for $20 \leq L$ the parameters a=1.2E-2, b=1.01.

In a bus composed of W parallel wires which drives several gates, the power is

given by Eq. 4.10:

$$P_{Bus}(L, W, N) = L_N.K_{wire}.f_{clk}.f_{duty}.W + N.Z_{std}.f_{clk}.f_{duty}.W \tag{4.10}$$

where K_{wire} represents the capacitance switched by the wire during a signal transition for one LE-length, where Z_{std} represents the input capacitance switched by one wire per signal transition for every standard-load, and where L_N is the average length of each wire. In this model, the $(1/2)V_{dd}^2$ term has been absorbed into the constant K_{wire} and Z_{std}.

To determine K_{wire}, different lengths of wires with one fan-out were realized on an FPGA and the power was measured. The source and destination were DFF arrays with width W=16 bits, and these were manually positioned and fixed using the Altera tools. K_{wire} is defined as the weighted average of the power dissipated in a wire of unit length, measured over several different lengths. In practice, K_{wire} has different values in the x and y dimensions, since the LEs do not have unity aspect ratios. However, to yield a tractable analysis we assume a single value of K_{wire} by taking the average of the experimental results. Figure 4.8 illustrates the test circuit for the power consumption of a wire. Referring to figure 4.8 the power consumed in a wire connecting DFF1_1 and DFF2_1 is determined by the total power consumption of the DFF1_1 (which includes the power consumption of DFF and the power consumption of the wire) minus the power consumption of DFF.

Figure 4.9 shows the analytic and experimental power consumption in a wire with different LE-lengths.

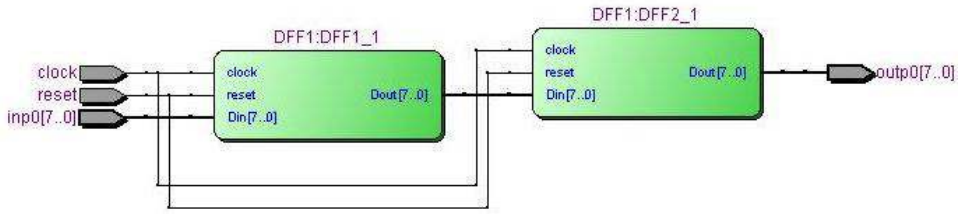


Figure 4.8: Test circuit for power consumption in a wire

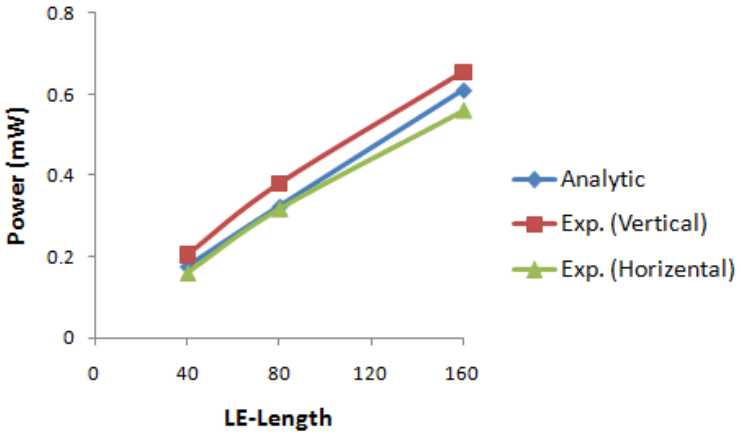


Figure 4.9: Analytic and experimental power consumption of a bus with one load (W=16)

L_N in Broadcast-and-Select Crossbar switches:

To determine the length and power consumption of a bus of length of L_N in an NxN Broadcast-and-Select (B&S) crossbar switch, we need to assume the following:

1. All multiplexers and the crossbar switch have unity aspect ratios (ie a square shape)
2. The wires are vertical or horizontal (No diagonal wires)
3. The placements of all components are manually fixed
4. DFFs are manually placed very close to the crossbar switch, around the perimeter
5. Each LAB is square with unity aspect ratio (i.e. size (x by x)).

To estimate the length of a broadcast bus in an NxN B&S crossbar switch, we can consider the minimum-cost spanning tree that connects one input or source (root) to N Multiplexers (children), where the cost is the sum of the edge lengths. A minimum-cost spanning tree is a Steiner tree, which in general is very difficult to find. However, for this regular switch, the Steiner trees are straight-forward to find.

An NxN crossbar switch consists of N Multiplexers. Let each multiplexer be constructed with M LEs. The dimensions of one multiplexer are $\sqrt{M}$ by $\sqrt{M}$. The switch with N multiplexers each composed of M LEs and has dimensions $\sqrt{NM}$ by $\sqrt{NM}$. The length of a minimum-cost spanning tree connecting the source to all N destinations is given by:

$$L_N = \sqrt{M}.N \tag{4.11}$$

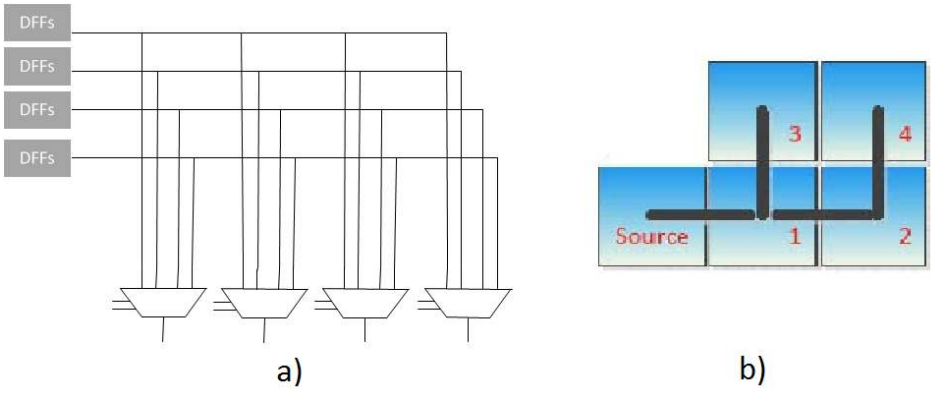


Figure 4.10: a) Test case for power estimation of a broadcast bus b) a Steiner tree for acquiring L_N

Proof. Consider a large N*N crossbar switch. Each axis consists of $\sqrt{N}$ multiplexers. Consider a bus entering the lower left corner, and crossing the switch horizontally. This bus will cross horizontally over $\sqrt{N}$ multiplexers, and have length $\sqrt{N}.\sqrt{M} = \sqrt{N.M}$ units. In each column, the bus must traverse an extra $\sqrt{N}-1$ multiplexers vertically. Therefore, each column will have a vertical bus of length $\sqrt{M}(\sqrt{N}-1)$. Therefore, the total length of the broadcast bus is $\sqrt{NM}+\sqrt{N}.\sqrt{M}.(\sqrt{N}-1)$ which is equal to $\sqrt{M}.N$. □

Figure 4.10 (a) shows the test case for power estimation, and (b) shows the model to find L_N for a broadcast bus in a 4x4 crossbar switch.

Referring to Eq. 4.10, the power of a bus equals the power used to drive the wires plus the power to drive the loads. By modelling a wire which drives several loads in Eq. 4.10, it is possible to infer the standard load Z_{std}. Figure 4.11 depicts a broadcast bus which drives 4 Muxes. Figure 4.12 shows the analytic and experimental power consumption of a broadcast bus in an NxN Brodcast-and-Select crossbar switches.

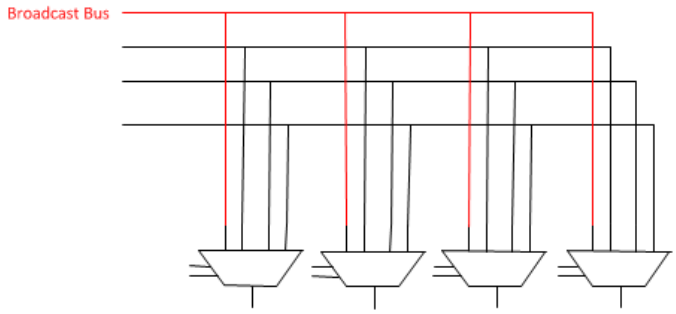


Figure 4.11: A Broadcast bus driving 4 loads

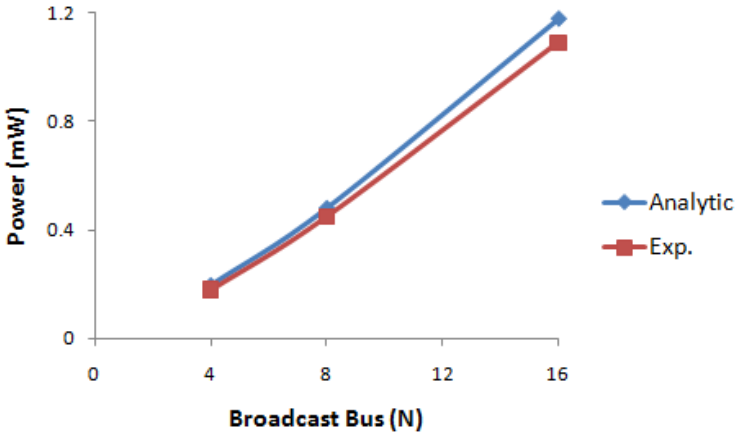


Figure 4.12: Analytic and expermental power consumed by a broadcast bus in an NxN Brodcast-and-Select crossbar switches(W=16)

4.2.3 Input Buffer

Generally, input buffers can be implemented with 2 methods, (i) with *Embedded Memory* (EM) Blocks, or (ii) with *DFF registers*. Hence, we will use both methods (EM blocks and registers) to realize the input buffers. In Altera FPGAs, thousands of EM blocks named MXK are available, each with X Kbits of SRAM. In the Altera Cyclone IV FPGA, hundreds of EM blocks named M9K are available, where each has 9 Kbits of SRAM. Each EM can be programmed as a W-bit wide memory, for $1 \leq$ W ≤ 32 bits. The use of EM requires some control logic and short wires for interfacing. The power consumption of the control logic and the short wires is very low and can be ignored.

When using EM blocks as a FIFO queue, two pointers are required (for reading and writing), and the resource usage of these pointers will depends upon the capacity of the input buffer. The number of the LEs used for these pointers is given by Eq. 4.12:

$$A_{IBEM}(Depth) = 2 \times \log_2 Depth \tag{4.12}$$

where $Depth$ is the depth of an input buffer.

In FPGAs, the power of a FIFO queue realized in an EM block is relatively independent of the FIFO capacity or the data path width [60], when FIFO queue fits completely within one EM block. The power consumed in an input buffer (i.e., a FIFO queue) residing in one EM block, for W ≤ 32 bits, is expressed by Eq. 4.13 [60]:

$$P_{IBEM} = K_{IBEM} . f_{clk} . f_{duty} \tag{4.13}$$

Referring to Eq. 4.13, K_{IBEM} represents the intrinsic and load capacitance switched by each output bit of the memory block during a signal transition, and f_{duty} is the duty cycle. In this model, the $(1/2)V_{dd}^2$ term has been absorbed into the constant K_{IBEM}.

In the second method to realize input buffers, the input buffers are constructed using DFFs available in the LEs in the FPGA. The resource usage depends on the buffer depth and width. An output buffer is typically used for latching the result of a read operation when depth of the input buffer is more than 1. The area and power consumption of the input buffers constructed by DFF registers are given by Eq. 4.14 and Eq. 4.15:

$$A_{IBDFF}(Depth, W) = (Depth + 1).W \tag{4.14}$$

$$P_{IBDFF}(Depth, W) = (Depth + 1).K_{DFF}.f_{clk}.f_{duty}.W \tag{4.15}$$

where K_{DFF} is the intrinsic and load capacitance switched by each output bit of a single DFF register during a signal transition. In this model, the $(1/2)V_{dd}^2$ in Eq. 4.3 has been absorbed into the constant K_{DFF}.

4.3 Crossbar Switches

In this section, analytic models for seven switch designs in 2 groups, shown in figure 4.13, are presented. In the first group 4 switch designs called the Demux-Mux,

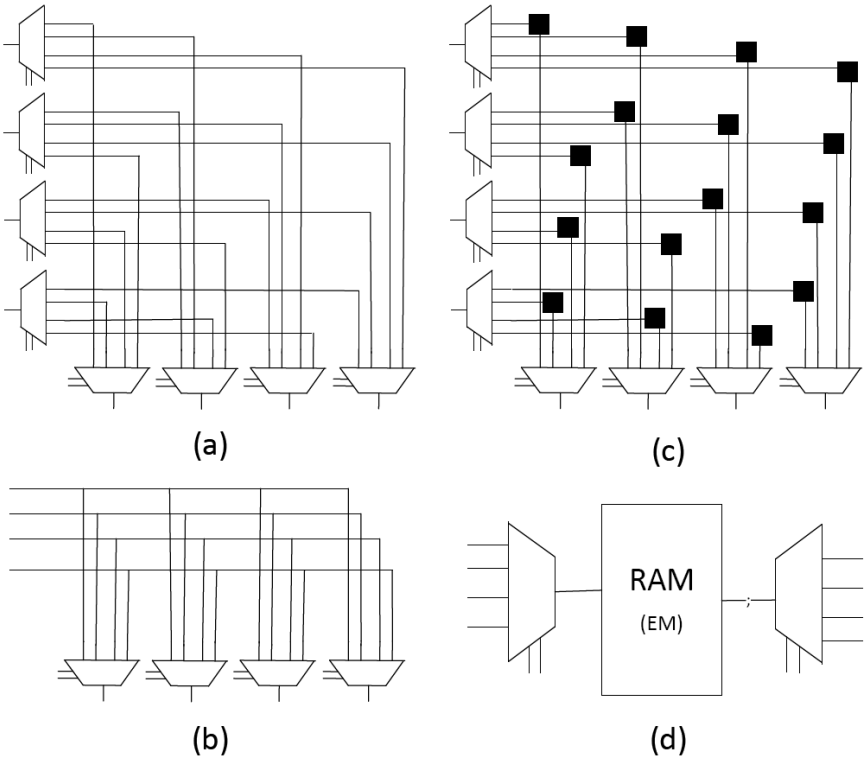


Figure 4.13: 4 Crossbar switches, (a) DM,(b) B&S, (c) PDM, (d) Ram-Based

Broadcast-and-Select, Pipelined Demux-Mux, and Ram-based crossbar switch are explained. The second group includes the crossbar switches in first group (except the ram-based switch), however, the Muxes and Demuxes are pipelined (Figure 4.14). From now on we refer to Demux as a Demultiplexer, and Mux as a Multiplexer.

4.3.1 The Demux-Mux (DM1) Design (S1)

Referring to figure 4.13a the DM1 crossbar design consists of Demuxes that connect to the input ports, and Muxes that connect to the output ports. In FPGAs, for small size switches (3x3 and less), the Demux are often replaced by broadcast buses by the synthesizer. The total resource usage of a DM1 switch (W bits wide) uses the area

model in Eq. 4.1 and Eq. 4.6, and is given by Eq. 4.16 [60]:

$$A_{S1}(N,W) = N.A_{Dmux}(1,N,W) + N.A_{Mux}(N,1,W) \tag{4.16}$$

The total power consumed by the DM1 switch uses the low power model of Mux (Eq. 4.5) and Eq. 4.8 and is given by Eq. 4.17:

$$P_{S1}(N,W) = N.P_{Dmux}(1,N,W) + N.P_{Mux}(N,1,W) + N.P_{bus}(\bar{L},W,Z) \tag{4.17}$$

where $\bar{L}$ is the average wire length in a bus between a Demux and Mux tree $\approx \sqrt{A_{S1}(N,W)}$ and $Z = 1$. Referring to Eq. 4.17, in every clock cycle at-most N buses are active. For small switches (4x4 or less), the power in the wires can be ignored.

The delay of $DM1$ switch of size NxN is equal to the delay of a large Demux tree (Demux(1,N,W)) plus delay of a large Mux tree (Mux(N,1,W)) and is given by Eq. 4.18:

$$D_{S1}(N,W) = D_{Demux}(1,N,W) + D_{Mux}(N,1,W) \tag{4.18}$$

4.3.2 The Broadcast-and-Select (B&S1) Design (S2)

Figure 4.13b illustrates an NxN *Broadcast-and-Select (B&S)* crossbar switch. Each input buffer broadcasts over a dedicated broadcast bus to all output ports. Each output port has one (N-to-1) mux-tree, which selects the broadcast from the appropriate input buffer.

The total area of this router is given by Eq. 4.19 [60]:

$$A_{S2}(N, W) = N.A_{Mux}(N, 1, W) \tag{4.19}$$

The total power consumed in this switch uses the models Eq. 4.4 and Eq. 4.10 and is given by Eq. 4.20 [60]:

$$P_{S2}(N, W) = N.P_{bus}(L, W, N) + N.P_{Mux}(N, 1, W) \tag{4.20}$$

The length of a broadcast bus L is equal to $N * \sqrt{X}$, where X is the average number of LEs in the Mux N-to-1 and can be found by Eq. 4.11.

The delay of $B\&S1$ switch NxN is equal to the delay of a large Mux tree ($D_{Mux}(N, 1, W)$) plus the delay of a bus with average length of $\sqrt{A_{S2}(N, W)}$, and is given by Eq. 4.21:

$$D_{S2}(N, W) = D_{Mux}(N, 1, W) + D_{bus}(\sqrt{A_{S2}(N, W)}) \tag{4.21}$$

4.3.3 The Pipelined (PDM1) Design (S3)

Figure 4.13c illustrates a pipelined-DM1. N^2 pipeline DFFs (with W DFFs each) are inserted between the Demuxes and Muxes of switch design S1 (DFFs are shown by black square in figure 4.13c). In our experiments, the DFFs in the existing LEs can be used, and no new resources (LEs) are needed. The area and power consumption are given by Eq. 4.22 and Eq. 4.23 respectively [60].

$$A_{S3}(N, W) = A_{S1}(N, W) \tag{4.22}$$

$$P_{S3}(N,W) = P_{S1}(N,W) + W.N^2.P_{DFF} \tag{4.23}$$

The power of a DFF P_{DFF} is given by $P_{DFF} = K_{DFF}.f_{clk}.f_{duty}$, where K_{DFF} represents the capacitance switched by one DFF per transition. The $(1/2)V_{dd}^2$ term in Eq. 4.3 has been absorbed into the constant K_{DFF}. Referring to Eq. 4.23, the f_{duty} of P_{DFF} is equal to (1/N) [60] because in every clock cycle at-most N words of data move through the switch and N DFFs get new data.

The delay of pipelined design switch of size NxN is equal to the delay of a large Mux tree (Mux (N,1,W)) (since it is larger than the delay of a large Demux tree), and is given by Eq. 4.24:

$$D_{S3}(N,W) = D_{Mux}(N,1,W) \tag{4.24}$$

4.3.4 A Ram-Based Design (S4)

Figure 4.13d illustrates a Ram-based switch. In a Ram-based switch, the crossbar switch is replaced by one EM block. A N-to-1 Mux connects to the N input ports to the EM, and a 1-to-N Demux connects the EM to the output ports. The area of the crossbar (in terms of LEs) equals the area of Mux and Demux and is given by Eq. 4.25:

$$A_{S4}(N,W) = A_{Mux}(N,1,W) + A_{Dmux}(1,N,W) \tag{4.25}$$

The total power consumed by Ram-based crossbar switch with datapath width W is given by Eq. 4.26:

$$P_{S4}(N,W) = P_{Mux}(N,1,W) + P_{EM}(W) + P_{Dmux}(1,N,W) \tag{4.26}$$

where P_{EM} is the power consumed by an EM and is given by Eq. 4.13.

The delay of Ram-based switch NxN is equal to delay of the EM block (D_{EM}) and the large Mux tree which has a larger delay than the large Demux tree, and is given by 4.27:

$$D_{S4}(N,W) = D_{EM} + D_{Mux}(N,W) \tag{4.27}$$

where in Cyclone family of FPGAs $D_{EM} = 2.53ns$.

Although this type of switch uses fewer resources, it suffers from low throughput, since the switching is done in the time domain.

4.3.5 The Demux-Mux 2 (DM2) Design (S5)

Figure 4.13a illustrates a Demux-Mux 2 switch design. The Demuxes and Muxes are constructed in tree topologies. Pipeline latches ($2^{\lfloor(\log_2 N)/2\rfloor}$ DFFs) are inserted in the middle level ($\lfloor\frac{\log_2 N}{2}\rfloor$) of the Demux and Mux trees, creating many smaller Mux and Demux trees of $\sqrt{N}$-to-1 and 1-to-$\sqrt{N}$ respectively. Figure 4.14 shows the 1-to-4 pipelined Demux and 4-to-1 pipelined Mux. These pipeline latches use DFFs within the existing LEs (no new LEs are required). By pipelining the trees, the trees are broken into $(1+\sqrt{N})$ sub-trees of size $\sqrt{N}$. The total resource usage of a Demux-Mux switch (DM2) uses the area model in Eq. 4.1 and Eq. 4.6 and is given by Eq. 4.28:

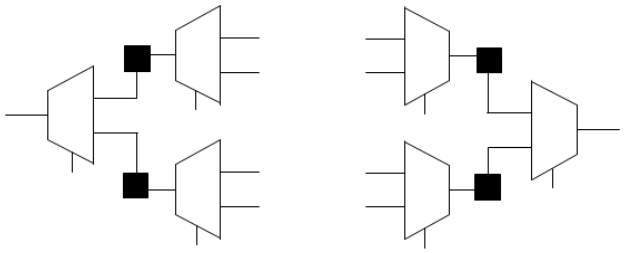


Figure 4.14: Pipelined Mux and Demux (Black squares denote DFFs)

$$A_{S5}(N,W) = N.(1+\sqrt{N}).\Big(A_{Dmux}(1,\sqrt{N},W) + A_{Mux}(\sqrt{N},1,W)\Big) \qquad (4.28)$$

As stated previously, for small sizes of Demux-Mux switches (3x3 and less), the Demuxes are often replaced by broadcast buses by the synthesizer. For example in the 1-to-4 Demux of 4x4 DM2 crossbar switch, only one of the three 1-to-2 Demuxes is remaining and others are replaced by a broadcast buses.

As mentioned earlier, by pipelining the trees, the trees are broken into $(1+\sqrt{N})$ sub-trees of size $\sqrt{N}$. These $(1+\sqrt{N})$ sub-trees are arranged in 2 stages (as shown in figure 4.14). In the first stage only one sub-tree exists, and in the second stage there are $\sqrt{N}$ sub-trees. In the DM2 switch, the low power model of Muxes is used since only one input port of a pipelined Mux switches. Therefore, in every clock cycle in the pipelined Muxes of the *DM2* switch only 2 sub-trees of size $\sqrt{N} - to - 1$ are active. In the pipelined Demuxes there are 2 sub-trees of size $1 - to - \sqrt{N}$ active in every clock cycle. The total power consumed by the switch design DM2 uses the low power model of Mux in Eq. 4.5 and power model of a Demux in Eq. 4.8, and is given by Eq. 4.29:

$$\begin{aligned} P_{S5}(N,W) &= N.2.\Big(P_{Dmux}(1,\sqrt{N},W) + P_{Mux}(\sqrt{N},1,W)\Big) \\ &+ N.P_{bus}(\bar{L},W,Z) + N.2.W.P_{DFF} \end{aligned} \qquad (4.29)$$

where $\bar{L}$ is the average wire length in a bus between a Demux and Mux tree $\approx \sqrt{A_{S5}(N,W)}$ and $Z = 1$. Referring to Eq. 4.29, in every clock cycle at-most N buses are active. The last term in Eq. 4.29 is the power used by the DFFs located in the Demuxes and Muxes. Since in the pipleined Mux of the *DM2* switch design only one input port switches, only one DFF is active in every clock cycle. In the pipelined Demuxes, only one DFF is active in every clock cycle (as shown in figure 4.14).

The delay of a NxN $DM2$ switch is equal to the delay of a Demux(1,$\sqrt{N}$,W) plus Mux($\sqrt{N}$,1,W), and is given by Eq. 4.30:

$$D_{S5}(N,W) = D_{Demux}(1,\sqrt{N},W) + D_{Mux}(\sqrt{N},1,W) \tag{4.30}$$

However, in an 4x4 DM2 some of the Demuxes are replaced by broadcast buses, and therefore the delay of $DM2$ switch NxN is equal to the delay of a Mux($\sqrt{N}$,1,W) plus delay of a bus. The length of a bus is approximated by the square root of area of all the Muxes in DM2.

4.3.6 The Broadcast-and-Select (B&S2) Design (S6)

Figure 4.13b illustrates a B&S2 switch design. In this switch, pipeline DFFs ($2^{\lfloor(\log_2 N)/2\rfloor}$ DFFs) are inserted in level $\lfloor\frac{\log_2 N}{2}\rfloor$ of each Mux tree, creating $(1+\sqrt{N})$ smaller Mux trees of size $\sqrt{N}$-to-1. These DFFs are shown by black squares in figure 4.14. These latches use the DFFs already available in the LEs, and do not incur an additional cost. The total area of NxN B&S2 switch is equal to the area of N N-to-1 Muxes which N-to-1 Mux is constructed by $(1+\sqrt{N})$ $\sqrt{N}$-to-1 Muxes, and is given by Eq.

4.31:

$$A_{S6}(N,W) = N.(1+\sqrt{N}).A_{Mux}(\sqrt{N},1,W) \tag{4.31}$$

The total power consumed uses Eq. 4.4 and Eq. 4.10 and is given by Eq. 4.32:

$$\begin{aligned} P_{S6}(N,W) &= N.(1+\sqrt{N}).P_{Mux}(\sqrt{N},1,W) \\ &+N.2^{\lfloor(\log_2 N)/2\rfloor}.W.P_{DFF} + N.P_{Bus}(L,W,N) \end{aligned} \tag{4.32}$$

where the first term is the power consumed in N Muxes, where the second term is the power consumed in the $2^{\lfloor(\log_2 N)/2\rfloor}$ pipeline DFFs, and where the last term is the power dissipated in the N broadcast buses. The length L is equal to $N * \sqrt{X}$, where X is the average number of LEs in the pipelined Mux N-to-1 and is found by Eq. 4.11.

The delay of $B\&S2$ switch NxN is equal to the delay of a Mux ($\sqrt{N}$,1,W) plus the delay of a bus $\sqrt{A_{S6}(N,W)}$ expressed in multiples of LE-Length, and is given by Eq. 4.33:

$$D_{S6}(N,W) = D_{Mux}(\sqrt{N},1,W) + D_{bus}(\sqrt{A_{S6}(N,W)}) \tag{4.33}$$

4.3.7 The Pipelined Demux-Mux 2 (PDM2) Design (S7)

The PDM2 switch design is illustrated in figure 4.13c, but DFFs are inserted in the Demux and Mux tree as shown in figure 4.14. These DFFs use the DFFs already available in the LEs, and do not incur an additional cost. The area and power

consumption are given by Eq. 4.34 and Eq. 4.35 respectively:

$$A_{S7}(N,W) = N.(1+\sqrt{N}).\Big(A_{Dmux}(1,\sqrt{N},W) + A_{Mux}(\sqrt{N},1,W)\Big) \quad (4.34)$$

$$P_{S7}(N,W) = P_{S5}(N,W) + W.N^2.P_{DFF} \quad (4.35)$$

where the f_{duty} of P_{DFF} is equal to (1/N) [60] because in every clock cycle at-most N words of data move through the switch and N DFFs get new data.

In the delay model of NxN PDM2, to determine the wire length we assume that the Muxes are placed in a square with unity aspect ratio and Demuxes are placed around the square. There are also N^2 DFFs between the Demuxes and Muxes. Therefore, the delay of a NxN PDM2 switch design is equal to the delay of a Mux ($\sqrt{N}$,1,W) (since it is larger than delay of a large Demux tree) and delay of a bus with length of $\approx \sqrt{N.(1+\sqrt{N}).A_{Mux}(\sqrt{N},1,W)}$. We approximate the length of the bus connecting the pipelined DFF (the DFF in figure 4.13c) to the Mux as the square-root of area of Muxes in the PDM2. Therefore the delay model of PDM2 is given by Eq. 4.36:

$$D_{S7}(N,W) = D_{Mux}(\sqrt{N},1,W) + D_{bus}(\sqrt{N.(1+\sqrt{N}).A_{Mux}(\sqrt{N},1,W)}) \quad (4.36)$$

where $N.(1+\sqrt{N}).A_{Mux}(\sqrt{N},1,W)$ is the area of Muxes in the PDM2 switch.

4.3.8 Experimental Results of the Crossbar switches

Table 4.1 compares the analytic and experimental results in different crossbar switches of sizes 4x4, 8x8, and 16x16. As seen in this table, the 'Ram-based' crossbar switch

(S4-Ram) dissipates the lowest power and area for large N. However, this switch operates in a slow Time division Multiplexing (TDM) mode and has low throughput. The switch design S1 (DM1) has the next lowest power consumption. Using the pipelined Mux or Demux increases the maximum frequency of the crossbar switches. However it affects their area and power consumption. For a small N, switch S5-DM2 has the highest maximum frequency. But for large N, switch S7-PDM2 has the highest frequency.

The analytic and experimental results in Table 4.1 show a good agreement (typically within 5-10%) which indicates that the analytic models for the crossbar switch designs are very accurate.

Several metrics can be used to choose the best crossbar switch for NoC routers. Since integrating a large number of crossbar switch under area and power constraints is a big challenge for designers, switches are compared by power per throughput (mW/Mbps) and area per throughput (LEs/Mbps). Figures 4.15 and 4.16 compare different crossbar switches in terms of power per throughput (mW/Mbps) and area per throughput (LEs/Mbps). The throughput of each switch is defined as the maximum possible delivered Mbps, expressed as (N ports) * (width W) * (clock rate $fmax$), assuming the switches are fully loaded. The results in figures 4.15 and 4.16 are based on experimental results.

In terms of power per throughput, the switch design S1 (DM1) has the lowest power per throughput for 4x4 and 8x8 crossbar switches. However for the 16x16 switch, the design S3 (PDM1) is the lowest power per throughput among the switches due to the relatively higher frequency. The B&S2 switch (S6) has the best area per throughput since it uses only pipelined Muxes. The B&S1 switch (S2) is the second

Table 4.1: Analytic and Experimental results for different crossbar switches (W= 16)

Crossbar Switch	Size (NxN)	Anlyt. Area (LE)	Exp. Area (LE)	Area Error (%)	Anlyt. Power (mW)	Exp. Power (mW)	Power Error (%)	Anlyt. Fmax (MHz)	Exp. Fmax (MHz)	Fmax Error (%)
S1-DM	4x4	384	384	0%	1.47	1.44	2%	479	457	5%
	8x8	1664	1664	0%	5.84	5.98	2%	319	364	12%
	16x16	6912	7168	4%	17.12	16.74	2%	240	280	14%
S2-B&S	4x4	128	128	0%	2.04	1.95	5%	421	480	12%
	8x8	640	640	0%	10.21	9.58	6%	305	385	20%
	16x16	2560	2560	0%	44.34	44.39	1%	237	314	24%
S3-PDM	4x4	384	384	0%	2.05	2.25	9%	765	589	29%
	8x8	1664	1664	0%	7.00	8.59	18%	510	505	1%
	16x16	6912	7168	4%	19.46	22.16	12%	382	422	9%
S4-Ram	4x4	96	96	0%	1.72	1.92	10%	264	271	2%
	8x8	208	208	0%	2.21	2.21	0%	225	250	10%
	16x16	432	432	0%	3.02	2.68	12%	197	232	15%
S5-DM2	4x4	320	328	3%	3.01	3.04	1%	624	774	19%
	8x8	1536	1456	5%	9.59	10.73	10%	479	476	1%
	16x16	7680	7680	0%	22.60	24.63	8%	479	390	22%
S6-B&S2	4x4	192	192	0%	3.91	3.61	8%	624	756	17%
	8x8	768	768	0%	14.00	12.60	11%	432	454	4%
	16x16	2560	2560	0%	53.69	48.24	11%	398	418	5%
S7-PDM2	4x4	576	576	0%	3.69	4.12	10%	624	702	11%
	8x8	1920	1920	0%	10.90	12.42	12%	432	532	18%
	16x16	7680	7680	0%	24.40	26.83	9%	398	460	13%

lowest area per throughput. The Ram-based design ($S4$) has the highest power and area per throughput, since at each time slot at-most one output is activated, resulting in low throughput.

4.4 Summary

In this chapter, analytic models for the area, power, and delay of the basic components of a router realized in FPGAs were studied. In addition the analytic models of 7

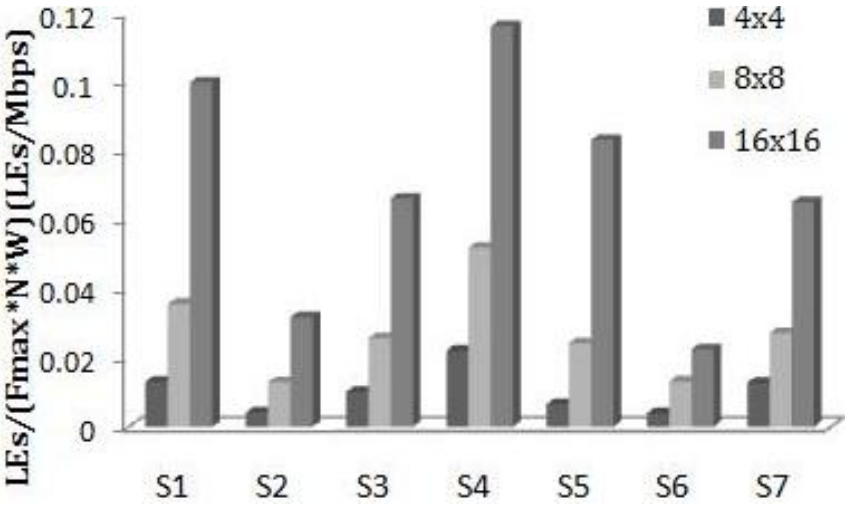


Figure 4.15: Area per throughput (Expermental Results)

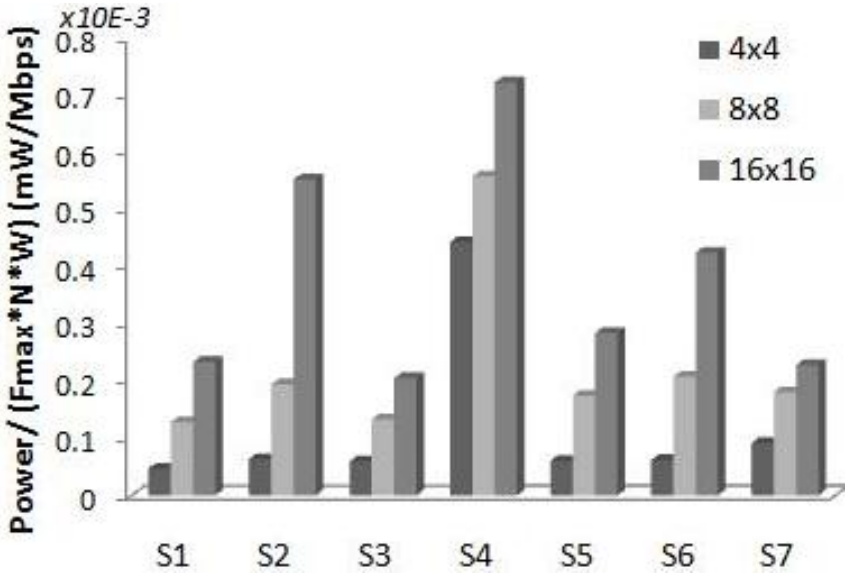


Figure 4.16: Power per throughput (Expermental Results)

crossbar switches were studied. Crossbar switches are compared in terms of power per throughput and area per throughput. If the crossbar switch is power-constrained, then the Demux-Mux crossbar switch (S1) is optimal, as it consistently has the lowest power. If the crossbar switch is the delay-constrained, then the Pipelined Demux-Mux switches (S3 and S7) are optimal, as they consistently have low delays. If the crossbar switch is area-constrained, then the B&S switch designs (S2 and S6) are optimal, as they consistently have low area.

Chapter 5

Analytic models for 2D Mesh, Torus and BHC

To date, there are no analytic models for the area, delay and power of NoCs realized in FPGA technologies. In this chapter, analytic area, power, and delay models for 3 different topologies are presented. All these topologies are made by the basic components which were described in chapter 4. The power model includes (a) maximum power consumption and (b) power consumption under 3 traffic patterns in wormhole switching. The 3 traffic patterns are (1) the Random Uniform traffic pattern, (2) the traffic pattern in Cooley-Tucky FFT algorithm, and (3) the traffic pattern in Bitonic sorting algorithm. The analytic models are compared to the experimental results, which result in good agreement. These models can be used for the early design-space exploration and evaluation of different NoC topologies for FPGAs.

5.1 Related Works:

Analytic models have been widely studied in ASIC NoC design. Recently, some analytic power and performance models using wormhole switching have been proposed in the NoC domain. Wormhole switching is more attractive in NoCs since it needs small buffers. In NoCs, most power is dissipated through buffers. In [54], two analytical models for predicting the average message latency and network power consumption using arbitrary buffer allocation schemes are presented. These models are based on queuing theory to predict the average packet latency in wormhole NoCs.

Reference [57] and [58] present a general performance model for latency and throughput of different NoCs using queuing theory. These models use the M/G/1/K queue and complex inter-dependent equations to estimate end-to-end delay of a packet. In these models, they assume that each PE independently injects a packet into the local buffer of its router following a Poisson model. This is a key assumption to simplify the analytic model by using queueing models that are already solved in queuing theory. However, studies have shown that higher packet injection rates decreases the percentage of exponentially distributed inter-arrival times [79].

An analytic model for energy consumption of different NoCs realized in ASICs is proposed in [66]. An analytic delay model at the transaction level (signal level) was proposed in [62] and then used for analysis of NoC topologies. Although these models are suitable for comparison, they are not accurate enough for the actual evaluation.

Some papers have used standard cell library parameters to analyze NoC cost metrics. In [56], analytic energy and area models of NoC topologies for tiled chip multiprocessors using predicted 65nm VLSI technology were presented. They also compared various topologies including a Mesh, concentrated Mesh, Torus and fat tree

Table 5.1: Previous Analytic models for NoCs

Model	Network	Delay	Power	Area
H. S. Wang et al. [59]	-		✓	
H. S. Wang et al. [66]	Mesh & Torus		✓	
U. Y. Ogras et al. [57], [58]	Arbitrary	✓		
J. Balfour and W. Dally. [56]	Arbitrary		✓	✓
M. Arjomand and H. Sarbazi-Azad [54]	Arbitrary	✓	✓	
M. Binesh Marvasti and T. H Szymanski [72]	Torus & GHC		✓	✓
S. E. Lee and N. Bagherzadeh [69]	-		✓	
T. H Szymanski [77] [83]	Mesh&BHC & Hypermesh	✓		

in terms of cost (power and area) and performance. In [55], an analytic methodology was proposed for the area and power of an IQ-switch using a Broadcast-and-Select architecture, which uses 180 nm CMOS VLSI.

Table 5.1 illustrates the analytic models proposed for NoCs based on a general router model. The Analytic models for FPGAs are based upon the same laws of physics as the analytic models for an ASIC, however FPGAs offer some unique features and constraints such as EM blocks at fixed locations on the die or limited type of primitive gates. These must be incorporated into the analysis.

5.2 Area and Delay Analysis of NoC Topologies

An NoC can be represented as a graph $G(V, E)$ where V is the set of vertices (or nodes) and E is the set of directed edges. Each vertex $v(x, y) \in V$ can be identified by its x and y coordinates on a plane. Assume each processor core uses C LEs of

the FPGA. In a *Concentrated* network, each NoC node has M processor cores with an area of M*C LEs. The area usage in each router is denoted by $A_{R,TDM}(D, W)$, where D is the router degree and W is the number of bits in each directed edge. Three different topologies are studied in this section. Assume each topology has N nodes, one router per node, and a concentration of M processors per node. There are 2 main design options for interconnecting processors to the routers: (i) each of the M processors in a node has its own dedicated high-speed IO port on the local router, or (ii) all of the M processors in a node share Y high-speed IO ports on the local router, where Y $\leq M$. Either option can be analytically modelled; we will assume router option (ii). In option (ii), each node has Y M-to-1 Mux trees and 1-to-M Demux trees, to connect the communicating processors to the Y ports on the router. The area of the interfacing logic for concentration in one router is given by:

$$\begin{aligned} A_{Conc}(Y, M, W) = Y \times A_{Dmux}(1, M, W) \\ + Y \times A_{Mux}(M, 1, W) \end{aligned} \tag{5.1}$$

There are 2 design options to multiplex multiple *Virtual Channels (VCs)* onto each directed edge, *Space-Division-Multiplexing (SDM)* or *Time-Division-Multiplexing (TDM)*. The TDM design option is less costly and is assumed here. When K VCs are multiplexed onto each edge, each router input port also has a 1-to-K Demux, a K-to-1 Mux, and K VC latches (each W bits wide), to implement the K VCs. Each TDM router output port has also one latch (W bits wide). In analytic and experimental results the latch in output port is ignored.

Therefore, the total area of the router with router degree = D, data path width

= W bits, and K VCs multiplexed onto each edge is given by:

$$\begin{aligned} A_{R,TDM}(D,W,K) &= A_{Si}(D,W) + D \times \Big(K.A_{VC} + W\Big) \\ &+ D \times A_{Dmux}(1,K,W) \\ &+ D \times A_{Mux}(K,1,W) \end{aligned} \tag{5.2}$$

where $A_{Si}(D,W)$ is the total resource usage of the crossbar switch design i, ($i = 1..7$), as described in chapter 4. Note that in direct networks, which every router can generate and absorb traffic, an NxN router uses Demuxes and Muxes with *N-1* ports. The slightly smaller Mux and Demux sizes are due to the fact that a packet arriving on a given port doesn't depart on the same port, since self-directed traffic is not injected into the network. The area for memory a VC (A_{VC}) is given by Eq. 4.12 and Eq. 4.14.

The total area of any NoC topology with N nodes (with router degree = D, Datapath width = W, with K VCs per edge, with area of the processor core = C, with Concentration = M, and with number of injector and extractor channels = Y) can be expressed by following parametric equation:

$$\begin{aligned} A_{NOC}(D,K,M,Y,W) = N \times M \times C \\ + N \times A_{R,TDM}(D,W,K) \\ + N \times A_{Conc}(Y,M,W) \end{aligned} \tag{5.3}$$

5.2.1 2D Mesh

A 2D Mesh with 16 nodes and without wrap-around edges is shown in figure 5.1a. This physical placement allows for fast communications over short-distance paths

between nearest neighbors. The 2D Mesh network with N nodes and contraction M has one router per node. Therefore each local router, except routers in corners and edges, has degree 4+Y per node, with 4 directed edges to nearest neighbors (North, East, South, West- N,E,S,W) and with Y local shared high-speed IO ports. The total area of Mesh NoC with N nodes can be obtained by Eq. 5.3, where the degree D = 4+Y except the routers on the edges.

5.2.2 2D Torus

A 2D Torus with 16 nodes and with wrap-around edges is shown in figure 5.1b. The layout in figure 5.1b results in a long wrap-around edge, which increases the critical path delay and decreases the maximum clock frequency. The optimization engine in the Altera compiler will typically use the 'Folded Torus' layout shown in figure 5.1d, which reduces the maximum wire length and which allows for reasonably fast communications. The area 2D Torus NoC with N nodes can be obtained by Eq. 5.3, where the degree D=4+Y.

5.2.3 2D BHC

A *Binary Hypercube (BHC)* with 16 nodes is shown in figure 5.1c. In a BHC with N nodes, each node has an integer label composed of $log_2 N$ bits. Each node has $log_2 N$ connections to nearest neighbors, whose integer labels differ from its own in one bit. The BHC with N nodes and contraction M has one router and M processors per node. Using the second router option, the router degree is $(log_2 N + Y)$. The total area of the BHC is given by Eq. 5.3, where its router degree is D=$log_2 N + Y$.

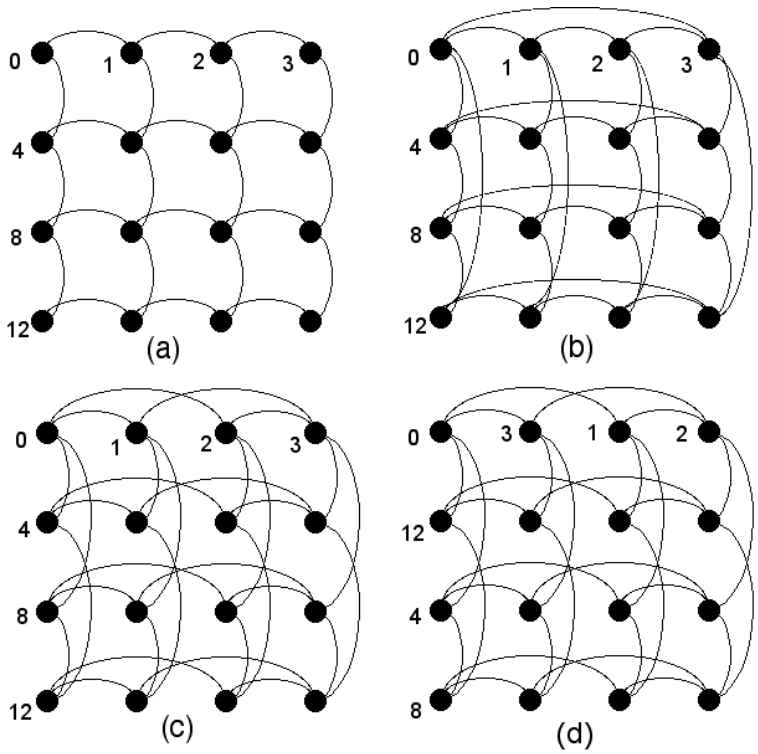


Figure 5.1: 3 NoC Topologies: (a) 2D Mesh, (b) 2D Torus, (c) Binary Hypercube (BHC), (d) 2D 'Folded' Torus layout.

5.2.4 Critical Path Delay

The maximum working frequency of each topology is mathematically modelled as the inverse of the critical path delay in the topology. The critical path delay is defined as the maximum delay of combinational logic between a source and destination register, including the setup and hold times for the registers. In each NoC topology the critical path depends on the router delay, the type of input buffers (EM or DFFs), and the wire delay between routers. The delay models for wires and routers were presented in chapter 4.

Definition: Define the *'Node-Distance'* (ND) of an NoC topology with N nodes as the shortest distance between 2 adjacent nodes in the X or Y dimensions. The ND is equal to the length of the shortest edges in figure 5.1, expressed in terms of the LE-length as defined in chapter 4. The ND of an NoC is denoted by $\Gamma_{Topology} = \sqrt{A_{Topology}}/\sqrt{N}$, where $A_{Topology}$ is the area of the NoC topology.

In FPGAs, the position of EM blocks are fixed on the die floorplan during the fabrication process. When EM blocks are selected as input buffers, the distance between 2 EM blocks in one row may affect the delay of topology. However, our experiments indicate that the delay of the short wires in an FPGA is small relative to the delay of combinational logic.

The delay of an NoC topology using EM blocks as Input Buffers given by:

$$D_{NoC-EM}(D,W) = D_{bus}(L) + D_{Si}(D,W) + D_{EM} \tag{5.4}$$

where L is the length of the longest link between the input buffers in 2 neighboring routers in one row or column of the FPGA, and D_{EM} is delay of an EM block. In

the Altera Cyclone IV family of FPGA, the delay of a link connecting 2 EM blocks in a row is 2.2 ns. Note that in Eq. 5.4 if the pipelined switch designs are used, the pipeline DFFs break the critical path of the NoC topology into 2 shorter paths. The higher delay path yields the critical path.

The delay of an NoC topology using DFFs as input buffers is typically smaller, since the DFFs are faster than EM blocks. For the larger networks ($N \geq 16$) the delay of a NoC topology using DFFs will depend on the delay of the router and the delay of the longest wire between neighboring routers, and is given by:

$$D_{NoC-DFF}(D, W) = D_{bus}(Z \times \Gamma_{NOC}) + D_{Si}(D, W) \tag{5.5}$$

where Z is the length of the largest wire connecting 2 neighboring nodes (expressed in multiples of the ND). In the BHC, let $Z = \frac{\sqrt{N}}{2}$. In a 2D Mesh, let $Z = 1$. In a 2D folded Torus using the layout in figure 5.1d, let $Z = 2$.

5.3 Analytic Power Models

The power of an NoC operating in steady-state is modelled by 3 components, the power consumed in the routers, buffers and wires, as shown in Eq. 5.6.

$$E[P_{NOC}] = E[P_{switches}] + E[P_{wires}] + E[P_{buffers}] \tag{5.6}$$

In a given time interval, $P_{switches}$ denotes the average power consumed in the switches, P_{wires} denotes the average power consumed in the wires, and $P_{buffers}$ denotes the

average power consumed in the buffers. When the NoC operates under a steady-state condition, these powers can experimentally be measured. Figure 5.3 illustrates the typical fractions of power used by crossbar switches, wires and input buffers in an FPGA in the 2D BHC NoC with 16 nodes using Demux-Mux 1 (S1). The power reported in this figure is the maximum achievable power consumption in the components.

Let N_{router} denote the number of routers in the NoC. Let N_{Port} denote the average router degree, i.e., the average number of input buffers per router. Let $\hat{L}_{wires}$ denote the total length of wires (for all NoC edges), expressed as a multiple of the Node-Distance. Assuming all components (switches, input buffers and wires) are 100% loaded, the maximum power used by an NoC is given by:

$$P_{router} = P_{Switch} + N_{Port}.P_{IB} \tag{5.7}$$

$$max(P_{NOC}) = N_{router}.P_{router} + \hat{L}_{Wires}.P_{Wire} \tag{5.8}$$

where P_{router} and P_{IB} are the average power consumed by each component assuming 100% load, and are given in chapter 4. P_{Wire} equals the power used by one wire segment of length L = 1 Node-Distance (with datapath width = W).

For all power analyses in this section, the following assumptions are made:

1. The NoC uses a deterministic ordered-dimension XY routing algorithm, and packets are delivered along minimum hops paths.

2. The load evenly distributed over the nodes.

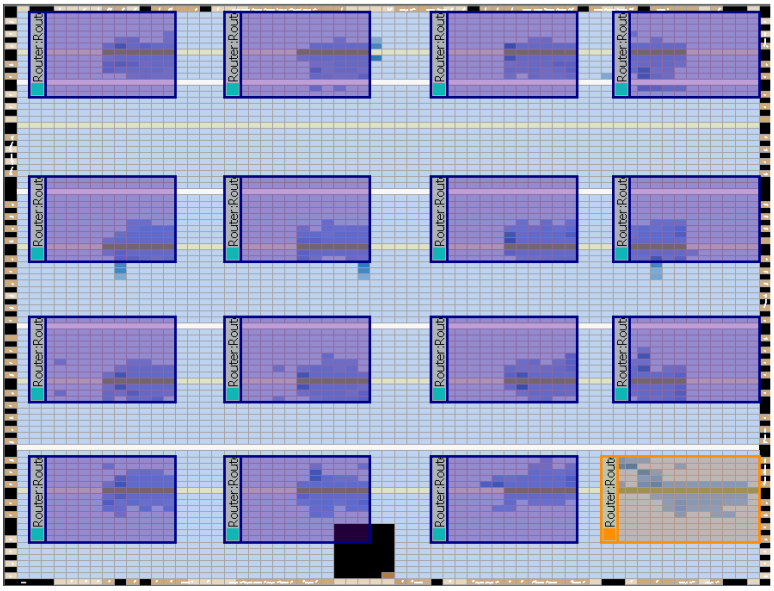


Figure 5.2: An NoC after Synthesis and Placement in an Altera FPGA

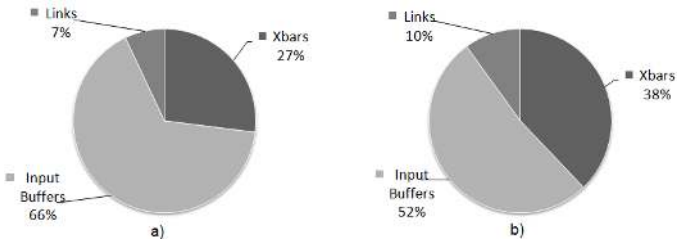


Figure 5.3: NoC Power Distribution, Saturated Mode, a) Buffer EMs b) Buffer DFFs. (2D BHC, N=16, W=16, Switch=DM, Buffer=4 flits, 50 MHz)

We also assume the layout of the NoC on the FPGA is relatively square after layout, synthesis and optimization, as shown in figure 5.2.

5.3.1 Power Model in Wormhole Switching

Let each idle PE generates a packet following a Poisson distribution with average arrival rate of $\lambda_P < 1$ packets per node per clock cycle. Therefore, the average injection rate of flits per idle node is $\lambda_F = \lambda_P \times m$ flits per node per clock cycle,

where m is the average number of flits per packet. In an NoC with N independent nodes, the maximum flit arrival rate before blocking is considered is $\approx N\lambda_F$ flits per clock cycle. Assuming an average blocking probability per header flit of P_B, the average effective rate of flits arrivals to the network per node per clock cycle is given by [71]:

$$\lambda'_F = \lambda_F(1 - P_B) \tag{5.9}$$

The blocking probability P_B can be defined as the expected waiting time that a header flit waits in the blocking queues over the packet latency, when their requested output links are busy (See Appendix B).

A 'hop' is defined as an edge traversal regardless of edge length. Let H be the expected number of hops traversed by a packet from the source to the destination. If the destinations are uniformly distributed, the average arrival rate of flits to one link can be estimated as the effective arrival rate of flits to the network ($N \times \lambda'_F$), weighted by the number of links traversed by a packet on average divided by the number of links in the network (N_{Link}) [78]:

$$\lambda_{Link} = N \times \lambda'_F \times \frac{H}{N_{Link}} \tag{5.10}$$

Similarly, the average arrival rate of flits to one IO port of a crossbar switch can be estimated as follows: Due to the wormhole routing, each packet generated at one node will appear at several components, which is reflected in the following equations:

$$\lambda_{Xbar} = \lambda'_F \times \frac{(H+1)}{N_{Port}} \tag{5.11}$$

$$\lambda_{IB} = \lambda'_F \times \frac{(H+1)}{N_{Port}} \tag{5.12}$$

where N_{Link} is the total number of links in the topology, and N_{Port} is the average number of ports per router in the topology.

Given the Random-Uniform or Butterfly traffic patterns, the analytic power model for any NoC network can be approximated by modifying Eq. 5.8, to reflect the reduced load of each component:

$$\begin{aligned} P_{NOC} &= \lambda_{Xbar}.N.P_{Switch} + \lambda_{IB}.N.N_{Port}.P_{IB} \\ &+ \lambda_{Link}.N_{Link}.\frac{L_{X+Y}}{H}.P_{Wire} \end{aligned} \tag{5.13}$$

where L_{X+Y} is the expected length of wires traversed by a packet under the given traffic pattern. In Eq. 5.13, the average wire length per edge hop traversed by a packet under the given traffic pattern is L_{X+Y}/H. In Eq. 5.13, λ_{link}, λ_{Xbar} and λ_{IB} are the effective arrival rates to each component, which have been lowered to reflect the blocking probability P_B due to wormhole switching. However, for light loads the P_B is very low and often can be ignored ($P_B \approx 0$) [54], and we assume it in this thesis. For moderate loads, P_B can be computed using analytic models presented in [54] and [91].

5.3.2 Wire Lengths and Hop Count

To determine the power consumption of NoC topologies under a given traffic pattern, we need to determine (1) the expected distance traversed by a packet (L_{X+Y}) in the given traffic pattern, and (2) the expected number of hops H per packet. Three

Table 5.2: List of Symbols Used Throughout This Thesis

Symbols	Description	Units
N	Network Size	
n	$\log_2 N$	
K	$\sqrt{N}$	
Γ_T	Node Distance (ND) of Topology T	$\sqrt{LEs}$
L_X	The X distance traversed by one packet in a row	ND
L_Y	The Y distance traversed by one packet in a column	ND
H	The expected number of hops traversed by one packet in a given topology	Integer
E^j	Butterfly Permutation, $j = 0, ..., \log_2(N) - 1$	
L_{BR}	The distance traversed by a packet in bit reversal permutation	ND

traffic patterns are explored; (1) Random Uniform traffic, (2) the traffic pattern in the Cooley-Tukey FFT Algorithm, and (3) the traffic pattern in the Bitonic sorting algorithm. Table 5.2 lists the symbols used throughout this chapter.

5.3.3 Random Uniform

For the Random Uniform (RU) traffic pattern, each node is equally likely to send a packet to every node. If the source and destination of a packet are the same router, assume that the packet does not traverse the router or the NoC.

2D Mesh

Consider a 2D Mesh with N nodes, as shown in figure 5.4a. Each row or column has K=$\sqrt{N}$ nodes. The 2D Mesh is a non-symmetric network; the edge nodes differ from the interior nodes, and symmetry cannot be exploited. The expected horizontal distance traversed by one packet depends upon the position of the source node i in

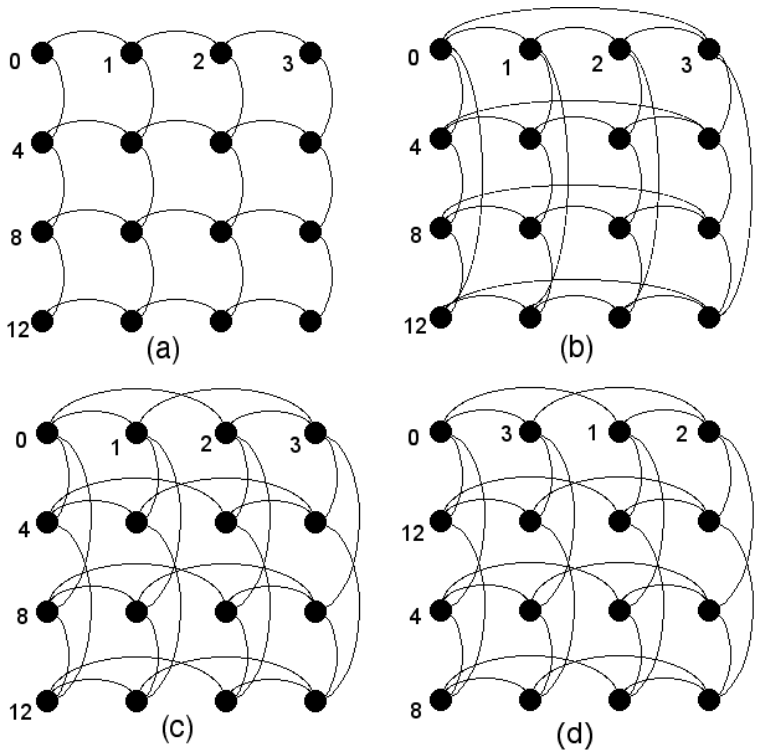


Figure 5.4: 3 NoC Topologies: (a) 2D Mesh, (b) 2D Torus, (c) Binary Hypercube (BHC), (d) 2D 'Folded' Torus layout.

each row, where $i = 0, .., K-1$.

Theorem 1: In a 2D Mesh, the expected distance traversed by a packet in the X dimension under the RU traffic pattern, expressed as multiples of the Node-Distance Γ_{Mesh}, is given by:

$$E[L_X] = \frac{2}{K^2} \times \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \Big(K - (2i-1) \Big)^2 \tag{5.14}$$

Proof. (Theorem 1). The proof is established by enumeration. Consider any node p in a row in the X dimension for $0 \leq p \leq K-1$. The traffic leaving node p is uniformly distributed over the K nodes in the row. Let i and j represent the number of edge traversals in the left and right directions respectively. The expected distance traversed in the X dimension by traffic leaving node p is given by:

$$\frac{1}{K} \Big(\sum_{i=0}^{p} i + \sum_{j=0}^{K-p-1} j \Big) \tag{5.15}$$

By taking the expectation for nodes $0 \leq p < K$ and mathematically rearranging the terms, the result in Eq. 5.14 is obtained. □

By symmetry, the expected distance traversed by a packet in vertical (Y) dimension is given by:

$$E[L_Y] = \frac{2}{K^2} \times \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \Big(K - (2i-1) \Big)^2 \tag{5.16}$$

Based on Theorem 1, the total expected distance traversed by a packet in a 2D Mesh in both dimensions under the RU traffic pattern, expressed as multiples of the

Node-Distance Γ_{Mesh}, is given by:

$$\begin{aligned} E[L_{X+Y}] &= (E[L_X] + E[L_Y]) \\ &= \frac{4}{K^2} \times \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \Big(K - (2i-1) \Big)^2 \end{aligned} \tag{5.17}$$

This result Eq. 5.17 is used in Eq. 5.13 to compute the 2D Mesh power. The expected number of hops (H) traversed by a packet in a 2D Mesh under the RU traffic pattern is also given by Eq. 5.17, since every edge has a length L = 1 ND.

Note that if the source and destination of a packet belong to the same router, the packet is routed by the resource network interface (RNI) and does not traverse the router or the NoC.

2D Torus

A 2D Torus is shown in figure 5.4b and figure 5.4d. The expected distance traversed by a packet under the RU traffic pattern in the 2D torus is easily computed by exploiting the symmetry of the network. The expected distance depends upon the layout of the Torus. The Altera Quartus CAD tools will lay out the 2D Torus using the folded Torus layout in figure 5.4d. The average distance traversed by a packet using the 2D folded Torus layout in figure 5.4d is given by Property 1.

Property 1: In a 2D Torus using the 'folded Torus' layout shown in figure 5.4d, where packets are constrained to follow minimum hop and minimum distance paths, the expected distance traversed by a packet in the X and Y dimensions under the RU traffic pattern, expressed as multiples of the Node-Distance Γ_{Torus}, is given by:

$$E[L_{X+Y}] = 2(K/2) \tag{5.18}$$

where $N = K^2$. The proof follows by enumeration. Let each edge in figure 5.4d have a length of 2 NDs (the length of the two short edges can be increased to 2 ND). By examining all possible source nodes, the expected distance traversed per packet in the X or Y dimension is K/2. The expected number of hops (H) traversed per packet in both the X and Y dimensions is obtained by dividing the expected distance in Eq. 5.18 by 2, as the length of each edge = 2 ND. Therefore the expected number of hops (H) traversed by one packet in both the X and Y dimensions $K/2$.

2D BHC

A binary hypercube (BHC) with $N = 2^n$ nodes is shown in figure 5.4c. The BHC can be viewed an an n-dimensional Torus, with 2 nodes aligned along each dimension. The average distance traversed by a packet in the BHC is given by Theorem 2.

Theorem 2: In a Binary Hypercube with the physical layout shown in figure 5.4c, where packets are constrained to follow minimum hop and minimum distance paths, the expected distance traversed by a packet in the X dimension under the RU traffic pattern, expressed in terms of the node distance Γ_{BHC}, is given by:

$$E[L_X] = \frac{K-1}{2} \tag{5.19}$$

Proof. (Theorem 2). The proof is established by induction. Suppose the theorem is true for $N = K^2$. Therefore the expected distance traversed by a packet leaving any node in a row is given by Eq. 5.19, which is repeated below:

$$E[L_X] = \frac{1}{K}\Big(\sum_{i=0}^{K-1} i\Big) = \frac{K-1}{2} \tag{5.20}$$

The theorem is clearly true for $K = 2$ (basis step), where the expected distance is given by:

$$\frac{0+1}{2} = \frac{K-1}{2} \tag{5.21}$$

Consider the Binary Hypercube with $N = (2K)^2$ nodes, using the 2D physical layout shown in figure 5.4c. According to the induction step, we assume that the Eq. 5.19 is true for $N = K^2$, then we need to prove that the Eq. 5.19 is true for $N = (2K)^2$ nodes. For $N = (2K)^2$, the expected distance traversed by a packet leaving any node which is RU distributed over all other nodes is given by:

$$\frac{1}{2K}\Big(\sum_{i=0}^{K-1} i + \sum_{j=0}^{K-1}(K+j)\Big) = \frac{2K-1}{2} \tag{5.22}$$

where the second summation includes traversal over the long edge of length K to the other half of the row. By rearranging, the expected distance traversed by traffic leaving any node which is RU distributed over all other nodes in the same row is given by Eq. 5.19 in Theorem 2. The result of Eq. 5.19 for $N = (2K)^2$ is equal to Eq. 5.22. Therefore, if the theorem is true for $N = K^2$ then it is also true for $N = (2K)^2$, and by induction, the theorem is true for all N. □

By symmetry, the distance in the Y dimension is equal to Eq. 5.19. Based on Theorem 2, the total expected distance traversed by a packet in a 2D BHC in both dimensions under the RU traffic pattern, expressed as multiples of the Node-Distance Γ_{BHC}, is given by:

$$E[L_{X+Y}] \;=\; K-1 \tag{5.23}$$

The 2D BHC can be viewed as an n dimensional torus, with 2 nodes aligned along each dimension. The average number of hops *(H)* per packet given the RU traffic pattern is $n/2$, as each dimension is traversed with probability 1/2. These results are used in Eq. 5.13 to compute the BHC power.

5.3.4 Cooley-Tukey FFT Algorithm

The Cooley-Tukey FFT algorithm is shown in figure 5.5. In a large FFT, neighboring nodes may transfer of several Kbytes to Mbytes of data. The bold lines in figure 5.5 show the complete paths for packets in the FFT. For example, a packet generated by node 0 visits nodes [8,12,14,15,15]. The FFT data-flow graph consists of log_2N Butterfly permutations (denoted E^j for $0 \leq j < log_2N$) followed by a Bit-Reversal (BR) permutation. In the next sub-sections, we compute the expected number of hops and the expected distance traversed by a packet in the FFT graph, to be used in Eq. 5.13 to find the NoC power.

2D Mesh

Assume the 2D Mesh layout as shown in figure 5.4a, and let $n = log_2N$. By examining each Butterfly permutation in figure 5.4a, the following properties are observed to be true:

Property 2: Each Butterfly permutation E^j for $0 \leq j \leq n/2 - 1$ requires each packet to traverse 2^j Node-Distances.

Property 3: Each Butterfly permutation E^j for $n/2 \leq j \leq n - 1$ requires each packet to traverse $2^{(j-n/2)}$ Node-Distances.

Property 4: In a 2D Mesh with the layout in figure 5.4a, where packets are

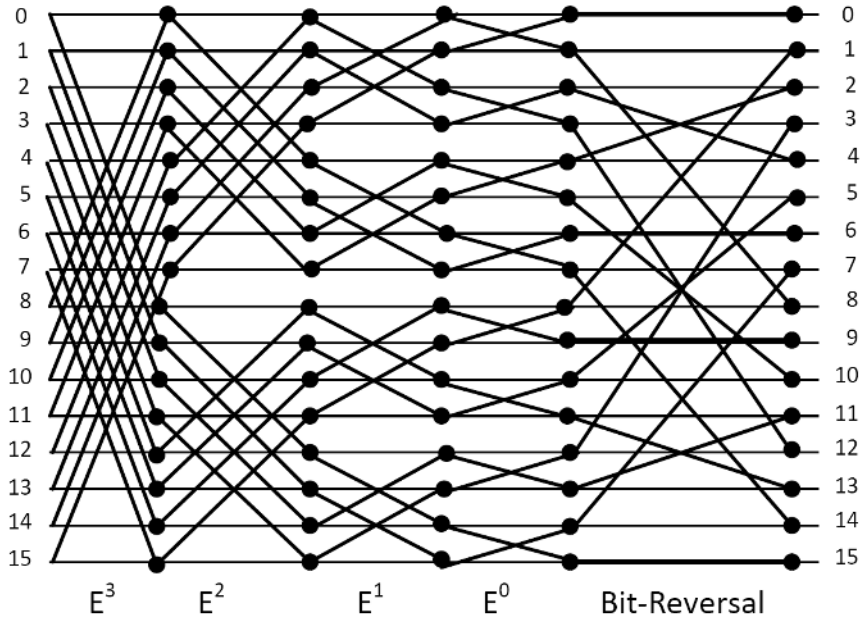


Figure 5.5: Cooley-Tukey FFT algorithm with 16 nodes

constrained to follow minimum hop and minimum distance paths, the expected distance traversed by one packet in the X and Y dimensions under the FFT algorithm, expressed as multiples of the Node-Distance Γ_{Mesh}, is given by:

$$\begin{aligned} E[L_{X+Y}] &= \sum_{j=0}^{n/2-1} 2^j + \sum_{j=n/2}^{n-1} 2^{j-n/2} + E[L_{BR}] \\ &= 2.(2^{\frac{n}{2}} - 1) + E[L_{BR}] \end{aligned} \tag{5.24}$$

Table 5.3: Average number of hops and distances under BR permutation

		Network Size			
Topology	Results	16	64	256	1024
Mesh (Fig. 5.4a)	Hops $E[H_{BR}]$	2.5	5.25	10.62	21.312
	Distance $E[L_{BR}]$	2.5	5.25	10.62	21.312
Torus (Fig. 5.4d)	Hops $E[H_{BR}]$	2	4	8	16
	Distance $E[L_{BR}]$	4	8	16	32
BHC (Fig. 5.4c)	Hops $E[H_{BR}]$	2	3	4	5
	Distance $E[L_{BR}]$	3	7	15	31

where $E[L_{BR}]$ is the expected distance per packet in the BR permutation. $E[L_{BR}]$ is determined experimentally in a Matlab program. According to Table 5.3, in a 2D Mesh $E[L_{BR}]$ is exactly equal to the expected distance per packet in the RU traffic pattern, given in Eq. 5.17. Similarly, according to Table 5.3, in a 2D Mesh the expected number of hops per packet in the BR permutation ($E[H_{BR}]$) is exactly equal to the expected hops per packet in the RU traffic pattern, also given in Eq. 5.17. According to Matlab simulations, the same equalities holds for the 2D Torus, BHC, GHC, and Hypermesh networks.

To find the power of the NoC using the FFT algorithm in Eq. 5.13, an expression for the average distance traversed by each packet transmission (L) is required. The distance reported in Eq. 5.24 is based on $(n+1)$ packet transmissions, as shown in figure 5.5. The expected distance per packet transmission is given by:

$$E[L_{X+Y}] = \frac{1}{(n+1)}.(2.(2^{(n/2)}-1) + E[L_{BR}]) \tag{5.25}$$

This result is used in Eq. 5.13 to determine the NoC power given the FFT traffic pattern. The average number of hops *(H)* traversed by a packet in the 2D Mesh under the FFT algorithm is also given by Eq. 5.25.

2D Torus

Properties 2 and 3 can be applied to a 2D folded Torus layout as shown in figure 5.4d (with slight adjustments due to the repositioning of the nodes in figure 5.4d relative to figure 5.4b.) Each butterfly permutation E^j for $0 < j < n/2$ requires a packet to traverse $2^{(j+1)}$ Node-Distances. Similarly, each butterfly permutation E^j for $n/2 \leq j < n$ requires a packet to traverse $2^{(j+1-(n/2))}$ Node-Distances. In a 2D

folded Torus with the layout in figure 5.4d, where packets are constrained to follow minimum hop and minimum distance paths, the expected distance traversed by each packet in the X and Y dimensions under the FFT algorithm, expressed as multiples of the Node-Distance Γ_{Torus}, is given by:

$$E[L_{X+Y}] = \frac{1}{(n+1)} * (4.(2^{(n/2)} - 1) + E[L_{BR}]) \tag{5.26}$$

The expected number of hops (H) traversed per packet in both the X and Y dimensions is again obtained by dividing the expected distance by 2. The expected number of hops traversed by one packet under FFT traffic pattern is:

$$E[H_{Torus}] = \frac{1}{(n+1)} * (2.(2^{(n/2)} - 1) + E[H_{BR}]) \tag{5.27}$$

where $E[H_{BR}]$ is the expected number of hops traversed by a packet in 2D Torus under the BR permutation.

2D BHC

Referring to the BHC in figure 5.4c, each packet in a butterfly permutation E^j for $0 \leq j < n/2$ traverses one edge with distance 2^j, as stated in Property 2. Each packet in a butterfly permutation E^j for $n/2 \leq j < n$ traverses one edge with distance $2^{(j-n/2)}$, as stated in Property 3. Therefore, the expected distance traversed by one packet in the FFT graph in the BHC and 2D Mesh topologies are equal. The expected number of hops traversed by one packet under the FFT traffic pattern is:

$$E[H_{BHC}] = \frac{1}{(n+1)} * (n + E[H_{BR}]) \tag{5.28}$$

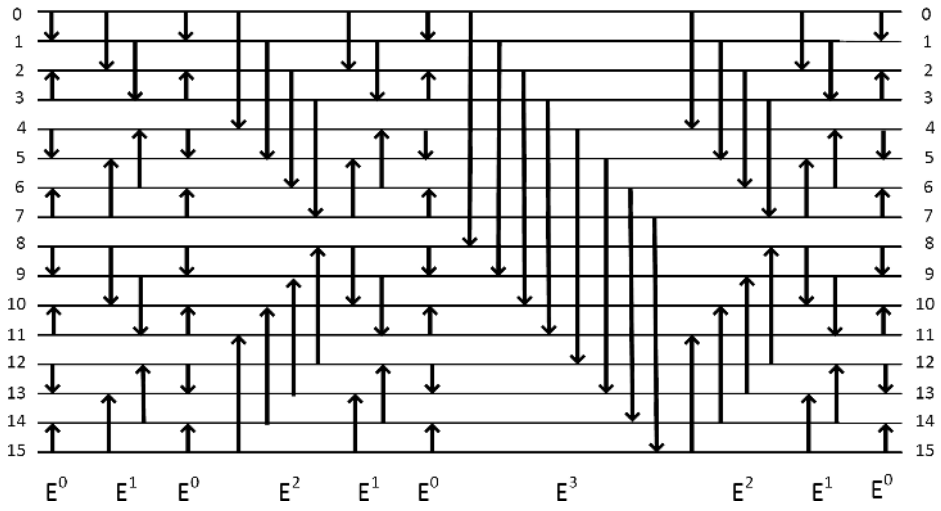


Figure 5.6: Bitonic sorting algorithm with 16 nodes [83]

where $E[H_{BR}]$ is the expected number of hops traversed by a packet in the 2D BHC under the BR permutation.

5.3.5 Bitonic sorting algorithm

The Bitonic sorting algorithm is shown in figure 5.6. The Bitonic sorting algorithm consists of $\frac{\log_2 N.(\log_2 N+1)}{2}$ butterfly permutations (denoted E^j for $0 \leq j < \log_2 N$). In the following, we compute the expected distance and average number of hops traversed by a packet in the Bitonic sorting algorithm.

2D Mesh

Consider the 2D Mesh layout with N nodes as shown in figure 5.4a, and let $n = \log_2 N$. In the 2D Mesh, the distance traversed by each packet in a butterfly permutation E^j for $0 \leq j < n$ is given by Property 2 and 3. The expected distance traversed by each packet in the Bitonic sorting algorithm, in a 2D Mesh, is given by Property 5.

Property 5: In a 2D Mesh with the layout in figure 5.4a, where packets are constrained to follow minimum hop and minimum distance paths, the total distance traversed by each packet in the X and Y dimensions under the Bitonic sorting algorithm, expressed as multiples of the Node-Distance Γ_{Mesh}, is given by:

$$L_{X+Y} = \sum_{j=0}^{n/2-1} (n-j) \times 2^j + \sum_{j=n/2}^{n-1} (n-j) \times 2^{j-n/2} \tag{5.29}$$

where the $(n-j)$ denotes the number of an E^j permutation per node in Bitonic sorting algorithm, as shown in figure in figure 5.6. For example in figure 5.6, there are 4 E^0, 3 E^1, 2 E^2, and 1 E^3 permutations.

The distance reported in Eq. 5.29 is based on $\frac{n.(n+1)}{2}$ packet transmissions, as shown in figure 5.6. The average distance per packet transmission is given by:

$$E[L_{X+Y}] = \frac{2}{n.(n+1)}.(\sum_{j=0}^{n/2-1} (n-j) \times 2^j + \sum_{j=n/2}^{n-1} (n-j) \times 2^{j-n/2}) \tag{5.30}$$

This result is used in Eq.5.13 to determine the NoC power given the Bitonic sorting algorithm. The average number of hops *(H)* traversed by a packet in the 2D Mesh under the Bitonic sorting algorithm is also given by Eq. 5.30.

2D Torus

Properties 2 and 3 can be applied to a 2D folded Torus layout as shown in figure 5.4d (with slight adjustments due to the repositioning of the nodes in figure 5.4d relative to figure 5.4b.) Each butterfly permutation E^j for $0 < j < n/2$ requires a packet to traverse $2^{(j+1)}$ Node-Distances. Similarly, each butterfly permutation E^j for $n/2 \leq j < n$ requires a packet to traverse $2^{(j+1-(n/2))}$ Node-Distances. In a 2D

folded Torus with the layout in figure 5.4d, where packets are constrained to follow minimum hop and minimum distance paths, the expected distance traversed by each packet in the X and Y dimensions under the Bitonic sorting algorithm, expressed as multiples of the Node-Distance Γ_{Torus}, is given by:

$$E[L_{X+Y}] = \frac{2}{n.(n+1)}.(\sum_{j=0}^{n/2-1}(n-j)\times 2^{(j+1)} + \sum_{j=n/2}^{n-1}(n-j)\times 2^{(j+1-n/2)}) \quad (5.31)$$

where the $(n-j)$ denotes the number of an E^j permutation per node in Bitonic sorting algorithm, as shown in figure in figure 5.6.

The expected number of hops (H) traversed per packet in both the X and Y dimensions is again obtained by dividing the expected distance by 2, i.e., the length of each edge. The expected number of hops H traversed by one packet under Bitonic sorting algorithm is:

$$E[H_{Torus}] = \frac{2}{n.(n+1)}.(\sum_{j=0}^{n/2-1}(n-j)\times 2^{j} + \sum_{j=n/2}^{n-1}(n-j)\times 2^{j-n/2}) \quad (5.32)$$

2D BHC

Referring to the BHC in figure 5.4c, each packet in a butterfly permutation E^j for $0 \leq j < n/2$ traverses one edge with distance 2^j, as stated in Property 2. Each packet in a butterfly permutation E^j for $n/2 \leq j < n$ traverses one edge with distance $2^{(j-n/2)}$, as stated in Property 3. Therefore, the expected distance traversed by one packet in the Bitonic sorting algorithm in the BHC and 2D Mesh topologies are equal.

In the 2D BHC, each packet in a butterfly permutation E^j for $0 \leq j < n$ is realized by one edge traversal. Therefore, the expected number of hops traversed by

one packet under Bitonic sorting algorithm is given by:

$$E[H_{BHC}] = 1 \tag{5.33}$$

5.4 Results

In this section, analytic and experimental results of different topologies are studied. The resource usage reported for the NoCs includes the resource usage of switches and input buffers in terms of LEs. The additional resource usage by the processors is not considered. As stated in Eq. 5.13, the power of an NoC is given by the power consumed in the switches, wires and input buffers. In our experiments, each NoC is clocked at the same clock frequency $f_{clk} = 50$ MHz. For the analytic results, we assume that the clock frequency of 50 MHz is feasible for all the topologies. The additional power consumed by the processors is not modelled, to focus on the power used in the NoC topology.

Given the RU, Bitonic or FFT traffic patterns, the crossbar switches and links in an NoC may be partially loaded. The power consumed by a partially loaded crossbar switch utilizing j ports out of D ports is simply equal to the maximum power of the crossbar switch, pro-rated by the fraction j/D, as shown in Eq. 5.13.

Table 5.4 shows the expected number of hops traversed by a packet under various traffic patterns, for NoCs with different sizes. As seen in this table, the BHC has the lowest number of hops, especially for large N.

To validate the proposed analytic models for the NoCs using different router designs, we compare the analytic results to extensive experimental results. The simulators are explained in chapter 3, and are used for getting experimental results.

Table 5.4: Average number of hops traversed by a packet for NoC topologies

		Network Size				
Topology	Traffic Pattern	16	32	64	128	256
Mesh	Uniform	2.5	3.64	5.25	7.48	10.625
	Bitonic	1.4	2.04	2.05	3.09	3.11
	FFT	1.7	2.16	2.75	3.513	4.513
Torus	Uniform	2	2.82	4	5.65	8
	Bitonic	1.4	2.04	2.05	3.09	3.11
	FFT	1.6	2.02	2.57	3.28	4.22
BHC	Uniform	2		3		4
	Bitonic	1		1		1
	FFT	1.2		1.28		1.33

Tables 5.5 and 5.6 compare the analytic and experimental results for the 2D Mesh, Torus, and BHC NoCs, with different switch designs. All networks use a datapath width W = 16 bits. In Table 5.5, EM blocks are used as input buffers, while in Table 5.6, DFF registers are used. The experimental power reported in these 2 tables is the peak power, obtained under the pseudo random walk model. The results show excellent agreement, typically within 8%.

As illustrated in Tables 5.5 and 5.6, realizing the Input-Buffers using EM blocks will reduce the cost (in LEs) and will require fewer memory bits, but it will decrease the maximum frequency. The lower frequency occurs since (i) the EM blocks are built by SRAM cells which are slower than DFFs, and (ii) the placement of EM blocks are fixed within the FPGA die, which will slightly increase the delay of the critical path.

Figures 5.7, 5.8, and 5.9 compare the analytic and experimental power results of topologies shown in figures 5.4a, 5.4c, and 5.4d using wormhole switching under RU traffic pattern, FFT, and Bitonic algorithms (all datapath widths W = 16 bits). The power consumed by arbitration and routing is excluded, to focus on the datapath. The

Table 5.5: Analytic and Experimental results for different NoC topologies with 16 nodes (Input buffer= EM blocks, Pseudo Random Walk model, W= 16, Freq.=50 MHz)

NoC	Switch Design	Anlyt. Area (LE)	Exp. Area (LE)	Area Error (%)	Anlyt. Power (mW)	Exp. Power (mW)	Power Error (%)	Anlyt. Fmax (MHz)	Exp. Fmax (MHz)	Fmax Error (%)
Mesh	S1	3392	3407	1%	112.89	102.35	1%	147	163	10%
	S2	2112	2112	0%	118.07	113.27	5%	141	157	10%
	S3	5312	5312	0%	119.20	117.32	2%	264	242	9%
	S4	1856	1860	1%	117.77	104.35	3%	170	151	12%
Torus	S1	8000	8000	0%	148.05	127.42	16%	147	146	1%
	S2	2880	2880	0%	158.05	149.89	6%	141	143	1%
	S3	8000	8000	0%	159.73	159.82	1%	264	237	11%
	S4	2368	2317	3%	145.17	126.19	15%	170	150	13%
BHC	S1	8000	8000	0%	148.05	127.42	16%	147	146	1%
	S2	2880	2880	0%	158.05	149.89	6%	141	143	1%
	S3	8000	8000	0%	159.73	159.82	1%	264	237	11%
	S4	2368	2317	3%	145.17	126.19	15%	170	150	13%

experimental results were obtained using Batch-mean method, were each simulation is divided into 8 batches. The first batch is ignored since it is used as a warm-up period. Each batch consists of 6000 clock cycles. Each packet includes 32 flits and and packet arrival rate is 0.0078 packets per node per clock cycle. Each input buffer is 4-flits deep. The analytic and experimental results show very good agreement, typically within 12%.

To have a fair comparison between the NoC topologies, each NoC is normalized to have an equal bisection bandwidth of O(N) bits/sec. We fix the datapath width of BHC (W_{BHC}) to 16 bits, and adjust the widths of the Mesh and Torus as described in chapter 2. For example, For N=64, the datapath widths for the Mesh, Torus and BHC are 64, 32 and 16 bits respectively. It is also assumed that each input buffer has capacity of 4 flits.

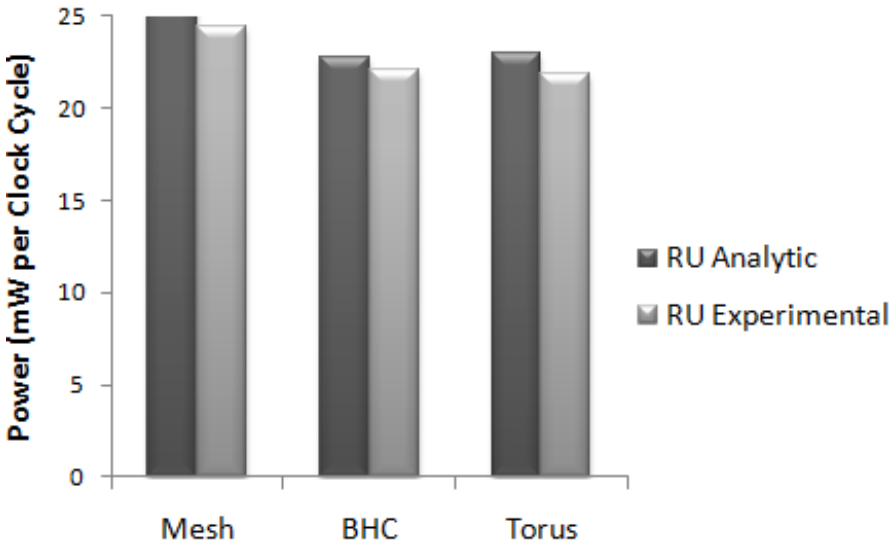


Figure 5.7: Analytic vs. experimental results of power under RU traffic (W=16, Buffer=EM block, Switch=S2)

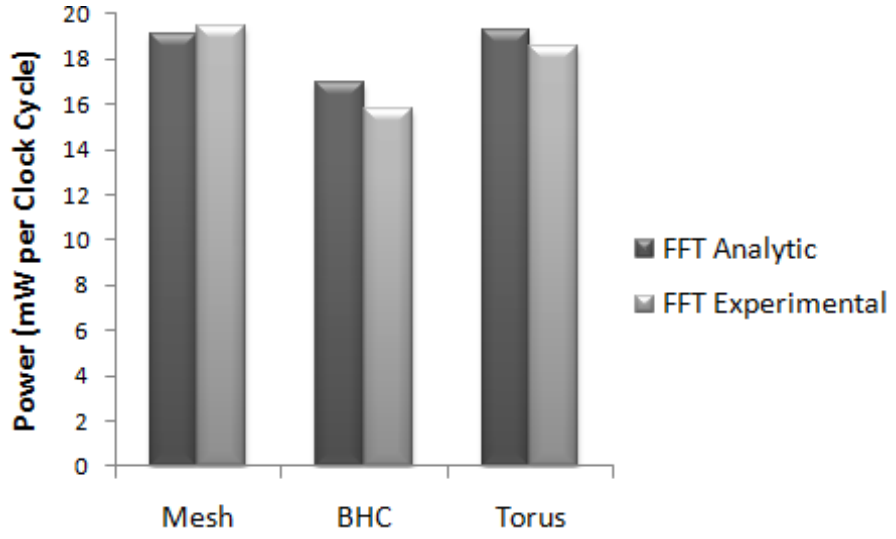


Figure 5.8: Analytic vs. experimental results of power under traffic pattern in FFT algorithm (W=16, Buffer=EM block, Switch=S2)

Table 5.6: Analytic and Experimental results for different topologies (Input buffer= DFF registers, Pseudo Random Walk model, W= 16)

NoC	Switch Design	Anlyt. Area (LE)	Exp. Area (LE)	Area Error (%)	Anlyt. Power (mW)	Exp. Power (mW)	Power Error (%)	Anlyt. Fmax (MHz)	Exp. Fmax (MHz)	Fmax Error (%)
Mesh	S1	8256	7990	3%	74.66	69.02	8%	296	249	18%
	S2	6976	6788	3%	80.06	86.61	8%	273	260	5%
	S3	10176	10173	1%	80.76	90.85	11%	326	270	20%
	S4	6720	7255	7%	79.81	83.17	4%	204	193	6%
Torus	S1	14080	12945	9%	101.04	101.2	1%	262	215	21%
	S2	8960	8826	2%	112.09	112.01	1%	251	246	2%
	S3	14080	13938	1%	112.72	117.43	4%	329	279	18%
	S4	8448	8395	1%	99.41	103.29	4%	192	173	11%
BHC	S1	14080	12945	9%	101.04	101.2	1%	262	215	21%
	S2	8960	8826	2%	112.09	112.01	1%	251	246	2%
	S3	14080	13938	1%	112.72	117.43	4%	329	279	18%
	S4	8448	8395	1%	99.41	103.29	4%	192	173	11%

Figure 5.10 shows the expected power consumption in different NoC topologies under the RU traffic pattern, when each router has one processing node and uses the 'Demux-Mux' (S1) crossbar switch, given equivalent bisection bandwidth. Each input buffer is constructed by DFFs and has capacity of 4 flits. The packet arrival rate is 0.0078 packet per node per clock cycle. Note that the y-axis is logarithmic.

As seen in figure 5.10, the BHC uses lowest power compared to other topologies, by a significant margin. For example, for N=256 nodes with DFFs as input buffers, the power of the (BHC, 2D Mesh, 2D Torus) are (0.678 Watts, 7.822 Watts, 3.542 watts) respectively (given equal Bisection BW). The BHC consumes only 9% of the power of the 2D Mesh, and the BHC consumes only 19% of the power of the Torus, for RU traffic. (It is well-known that for N=16, the 2D torus and BHC are topologically equivalent. This equivalence is observed in figure 5.10, where the power used by the BHC and Torus for N=16 are equivalent.)

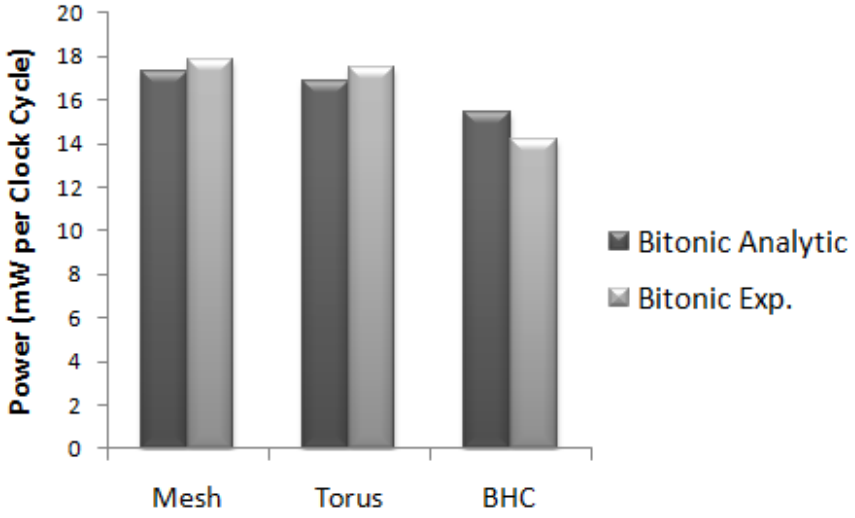


Figure 5.9: Analytic vs. experimental results of power under Bitonic sorting algorithm (W=16, Buffer=EM block, Switch=S2)

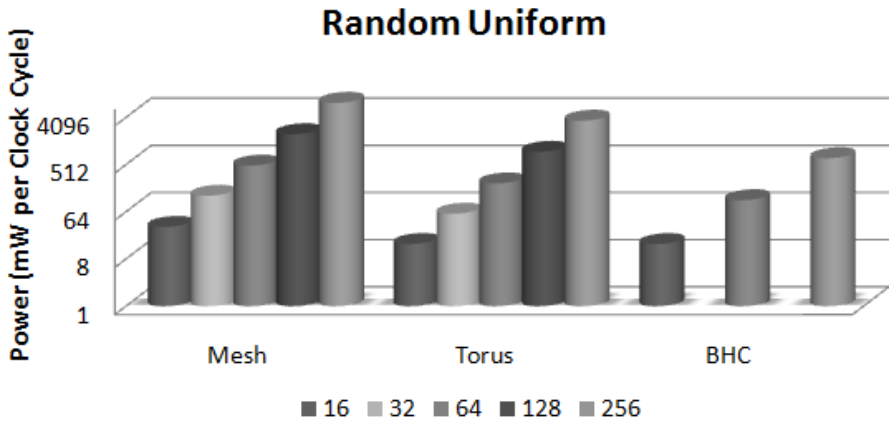


Figure 5.10: Analytic power results for different topologies and sizes under RU traffic pattern (with Equal Bisection Bandwidth, Switch=S1, Buffer= DFF Registers)

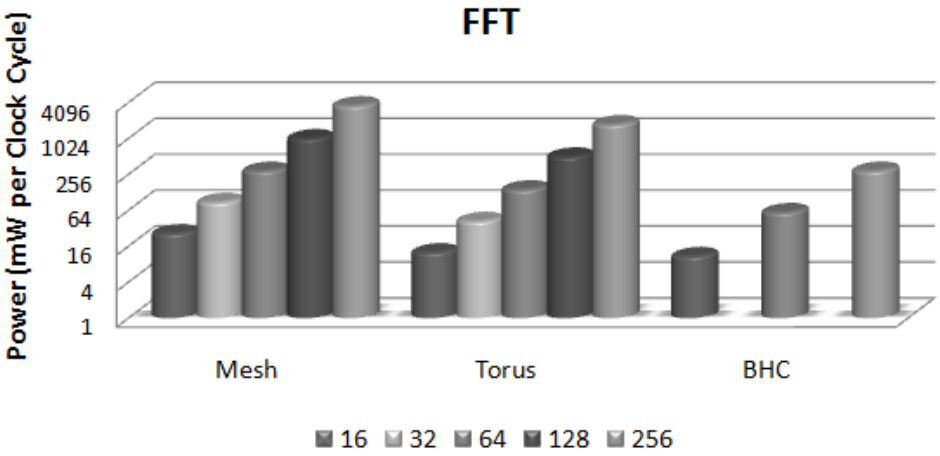


Figure 5.11: Analytic power results for different topologies and sizes under the Cooley-Tukey FFT Algorithm (with equal Bisection Bandwidth, Switch=S1, Buffer= DFF Registers)

Figure 5.11 shows the expected power consumption for different NoC topologies when the Cooley-Tukey FFT algorithm is used (given equal Bisection BW). Again, each router has one node and uses the 'Demux-Mux' crossbar switch. As seen in figure 5.11, the BHC uses lowest power compared to other topologies, by a significant margin. For example, for N=256 nodes with DFF registers as input buffers, the power consumption of the (BHC, 2D Mesh, 2D Torus) are (0.289 Watt, 3.627 Watt, 2.002 Watt) respectively. The BHC uses only 8% of the power of the 2D Mesh, and the BHC uses only 15% of the power of the 2D Torus, under the FFT traffic pattern with equivalent bisection bandwidth.

Figure 5.12 shows the analytic power consumption in the BHC with the 4 different switch designs, when RU traffic pattern is used. This figure can be used to select the best switch design, and to decide whether EM blocks or DFFs are used to realize the switch input buffers. (Note that the y-axis is logarithmic.)

Choosing a suitable crossbar switch and input buffer design is a trade-off among

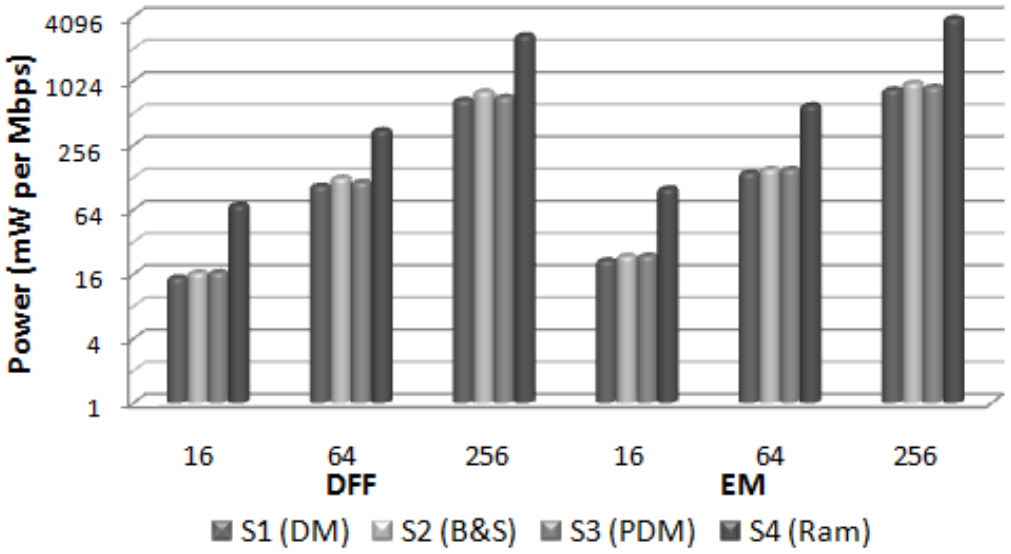


Figure 5.12: Power analysis of different switch designs in BHC with different network diameter, under RU traffic (W=16)

many competing design constraints. For small and medium sizes, if the applications are power-constrained then the Demux-Mux crossbar switches would be the best choice. If a design is delay-constrained, the best choice of switch would be the Pipelined-Demux-Mux design. For the area-constrained designs, the Broadcast-and-Select switch design is the best choice.

As stated in previous chapter, two types of input buffers are available on the FPGAs, the EM blocks and DFF registers. Using EM blocks (rather than DFFs) for the input buffers will reduce the NoC area, since this choice uses embedded memory bits rather than DFFs. However, the use of EM will decrease the maximum allowable clock frequency. In a design with multiple VCs or large input buffers, realizing input buffers using EM is power efficient and area efficient since all the buffer memory for multiple virtual channels can fit into one EM block.

5.5 Conclusions

Analytic models for the area, power, and delay of 3 NoC topologies have been proposed, including 2D Mesh, Torus, and BHC. Three traffic patterns are considered, Random Uniform, the traffic pattern in the FFT algorithm, and the traffic pattern in the Bitonic sorting algorithm. The NoC analytic models are compared to extensive simulations, and shown to be very accurate, typically within 10%, comparable to the ORION CAD modelling tool [41]. The experimental results shows that the proposed analytic models can be efficiently used for the early design-space exploration of various NoC configurations and topologies within an acceptable accuracy level. Our analysis indicates that given equivalent bisection bandwidth, Binary-Hypercubes consume much less power than the 2D Torus and Mesh, primarily due to the reduction in area and power consumed by the routers. In BHC, packets traverse fewer routers on average, which significantly lowers the required power. In an FPGA environment, when performing FFT computations the BHC consumes $\approx 8\%$ of the power of a 2D Mesh, and $\approx 15\%$ of the power of a 2D Torus.

In addition, when using multiple VCs on an edge, the use of Embedded Memory blocks for input buffers rather than DFFs will significantly decrease both the area (measured in LEs) and power used in the routers and NoCs.

Chapter 6

Analytic models for 2D Hypermesh and GHC

In chapter 5, we studied analytic models for the graph-based NoC topologies such as 2D Mesh, Torus, and BHC. In this chapter, 2 other topologies, named the Generalize Hypercube (GHC) and Hypermesh, are studied. The 2D Mesh, Torus, BHC, GHC are graph-based NoC topologies, while the Hypermesh is based on the concept of hypergraphs, which are generalizations of the conventional graph in which individual edges are able to join an arbitrary number of nodes [83]. In a 2D Hypermesh, the nodes in each row or column are members of a hyperedge, where packets can traverse a hyperedge with minimal queuing delays. The Cray TITAN supercomputer located at the Oak Ridge National Laboratory in the USA uses an optical Hypermesh type of interconnection network. Several other supercomputers also use Hypermesh Networks.

In this chapter, accurate analytic models for the area, delay and power of the 2D Hypermesh and GHC NoC topologies realized in FPGAs are presented. These 2

topologies are constructed by the basic components which were explained in chapter 4. The power models consist of (a) peak power and (b) power consumption in wormhole switching. Power consumption in wormhole switching includes the RU traffic pattern, the traffic pattern in the FFT algorithm, and the traffic pattern in the Bitonic sorting algorithm. The 2D Hypermesh, BHC, and GHC are compared in terms of power, area, and energy consumption. To compare the NoCs we also propose a design metric, called the energy-area product, which includes both cost and performance metrics. At the end, the 2D Hypermesh, BHC, and GHC are evaluated in terms of energy-area product.

6.1 Related Works

The concept of a hypergraph-based Hypermeshes was presented in [81] [83]. In the Hypermesh network, all the nodes aligned along a hypercube dimension are interconnected by a distributed switching capability, which is formally modelled as a hypergraph hyperedge. Hypermeshes are particularly attractive for optical implementation, where large distributed optical *Broadcast-and-Select* switches can be realized. The optical implementation of Hypermeshes can reduce the complexity problems of electrical implementation [83]. Two possible Hypermesh implementations using centralized crossbar switch are proposed in [83]. One of them is a 2D Hypermesh with electrical crossbar switch and electrical transmission, while the other one is electrical crossbar switch and optical transmission. It has been shown in [83] that the Hypermesh topology provides better performance relative conventional graph-based networks, such as the Mesh, Torus and hypercubes.

Another implementation for Hypermeshes is to use a 'Distributed Crossbar Switch'

(DCS) [83] [85] [86]. In [86], the *DCS* Hypermesh was examined under uniform and non-uniform traffic models. In a network with N nodes, the *DCS* Hypermesh is constructed by $\sqrt{N}$ ($\sqrt{N}-1$)-to-1 Multiplexers. The *DCS* Hypermesh uses bit-pipelining to reduce latency. The major concerns in the *DCS* Hypermesh are the way the contention is handled in the buses and the wiring complexity. It has been shown that Hypermeshes generally outperforms meshes, Torus, and hypercubes under both uniform and non-uniform traffic models [81] [83] [90]. Reference [88] proposed a performance model to predict the mean average latency in wormhole-switched Hypermeshes in the presence of hot-spot traffic. The performance comparison between *DCS* Hypermesh and *spanning-bus hypercube* was done in [83] [89]. It has been shown that the *DCS* Hypermesh implementation outperforms the spanning-bus hypercube.

To date, most of the papers either have proposed cost and performance model for graph-based networks such as Mesh, Torus, or hypercube topology, or have been related to performance models of the Hypermesh topology. *None* of the prior works presents accurate analytic area, delay, and power models for the Hypermesh NoC realized in FPGA technologies, or compares different graph-based NoC topologies with the Hypermesh topology, when realized in FPGA technology.

6.2 Hypermesh Topology

In conventional graph-based network such as the Mesh, Torus, BHC, and GHC, each node has a router with direct connections to its nearest neighbours.

An n-dimension k-radix Hypermesh can be defined as a regular hypergraph which has k^n nodes. Each node is identified with an n digit vector in radix k arithmetic. All the nodes aligned along any of the n dimensions are interconnected with a distributed

low-latency switch, which is modelled as a hypergraph hyperedge. Each Hypermesh node is connected to all the other nodes whose positions differ from its own in exactly one dimension. Figure 6.2 shows the logical diagram of a 2D Hypermesh, where each bold vertical or horizontal line denotes a hyperedge. It is possible to use a centralized electronic crossbar switch to realize each hyperedge. Figure 6.4 shows a 4x4 2D Hypermesh implemented with centralized electrical crossbar switches. Each black box represents a centralized electrical crossbar switch, while each circle represents a node. The node contains a small router to allow packets to traverse the multiple dimensions, and to interconnect the hyperedges in 2 dimensions to the processors. The Hypermesh has higher bisection bandwidth when compared to the *spanning-bus* hypercube, since the single spanning buses are replaced by switches.

The logic diagram of a 2D Hypermesh with distributed switches is shown in figure 6.3 and its implementation is shown in figure 6.5. In this distributed implementation, multiplexers are connected to the long row and column channels.

The 2D Hypermesh is similar to a 2D Mesh in that each node can communicate with its nearest neighbors in one transmission or one hop. It is different from a 2D Mesh in that a transmission along a row (or a column) only encounters a queue (i.e., an input buffer with associated arbitration) once each hyperedge traversed. In contrast, a 2D Mesh with virtual channels may entail queuing in every input-queued router along a row or column. In a 2D Hypermesh the graph-theoretic diameter is fixed at 2 hops but the graph-theoretic diameter in the 2D Mesh is equal to $2\sqrt{N}$.

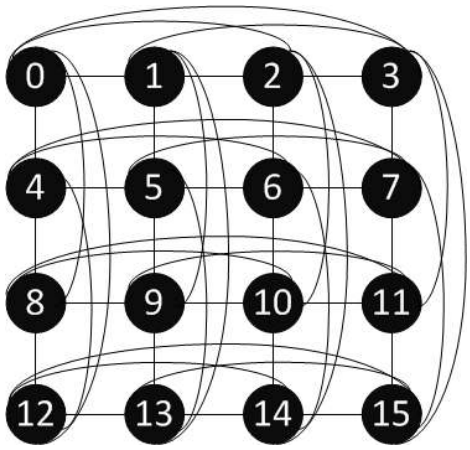


Figure 6.1: 2D GHC with 16 nodes

6.3 Area and Delay Analysis of Hypermesh and GHC NoCs

In this section, area and delay models of 2D GHC and Hypermesh are discussed.

6.3.1 2D GHC

A 2D Generalized Hypercube with 16 nodes is shown in figure 6.1. In a 2D GHC with N nodes, each node has $2 \times (\sqrt{N} - 1)$ connections to the neighbours. In other words, each node has a directed connection to other nodes in the same row and column, plus Y links to connect to the associated PEs. The area model of a 2D GHC NoC with N nodes can be obtained by Eq. 5.3, where the degree of routers is D= $2(\sqrt{N} - 1)$+Y.

6.3.2 2D Hypermesh

Hypermeshes cannot be modelled as conventional graphs with point-to-point edges. The 2D Hypermesh network with N nodes is modelled as a hypergraph $H(V, HE)$,

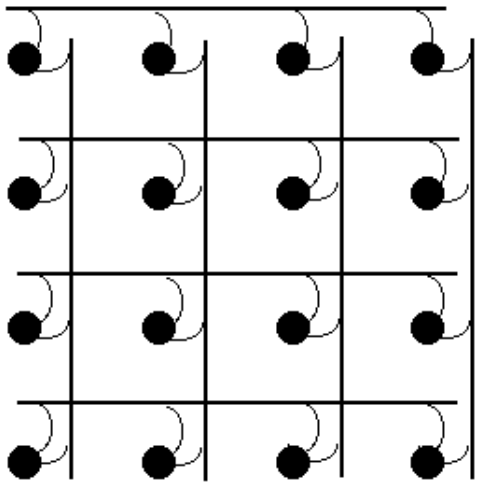


Figure 6.2: 2D Hypermesh with 16 nodes (logical diagram) [83].

where V is the set of nodes and HE is a set of hyperedges. Each hyperedge $h \in HE$ represents a set of nodes which are interconnected with multiple transmission channels (each of width W bits). In a 2D Hypermesh with N nodes, there is one hyperedge in each row and in each column for a total of $2\sqrt{N}$ hyperedges (when N is divisible by 4). A hyperedge with $\sqrt{N}$ nodes can represent a $\sqrt{N} \times \sqrt{N}$ crossbar switch.

In figures 6.4 and 6.5, each hyperedge is implemented by a 4x4 B&S crossbar switch. Each node is connected to the nodes in a same row and column by the 4x4 B&S crossbar switch (with datapath width W bits). Therefore, each node in the hyperedge can transmit data on its own dedicated transmission channel without conflict. Each node contains a simple *3x3* input-queued router, to connect its PEs to 2 hyperedges. Data can be transmitted from a node to another node in one dimension in one hop, so in a 2D Hypermesh data traverse the network in at-most 2 hops. In general, for large Hypermeshes the transmission channels in each hyperedge can contain pipeline latches. However, for small networks, unpipelined channels can be used.

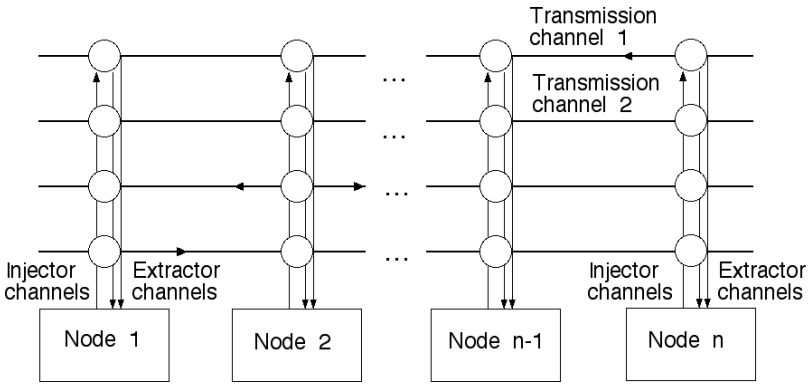


Figure 6.3: A hyperedge realized as a distributed crossbar switch [83][84][85].

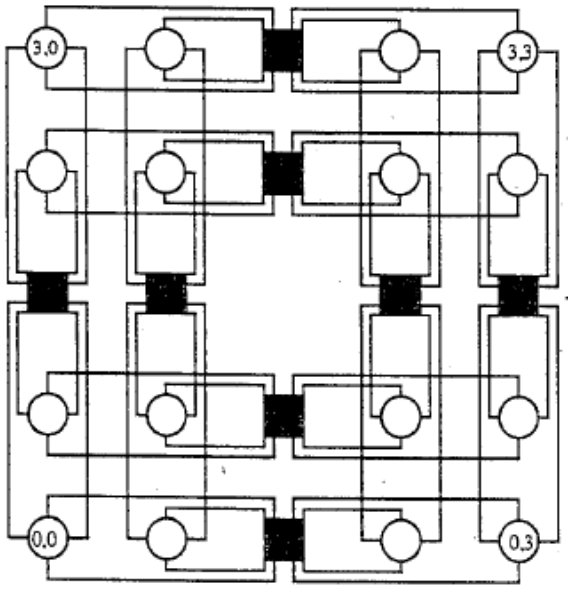


Figure 6.4: Hypermesh with 16 nodes using crossbar switches (or routers) as hyperedges [83].

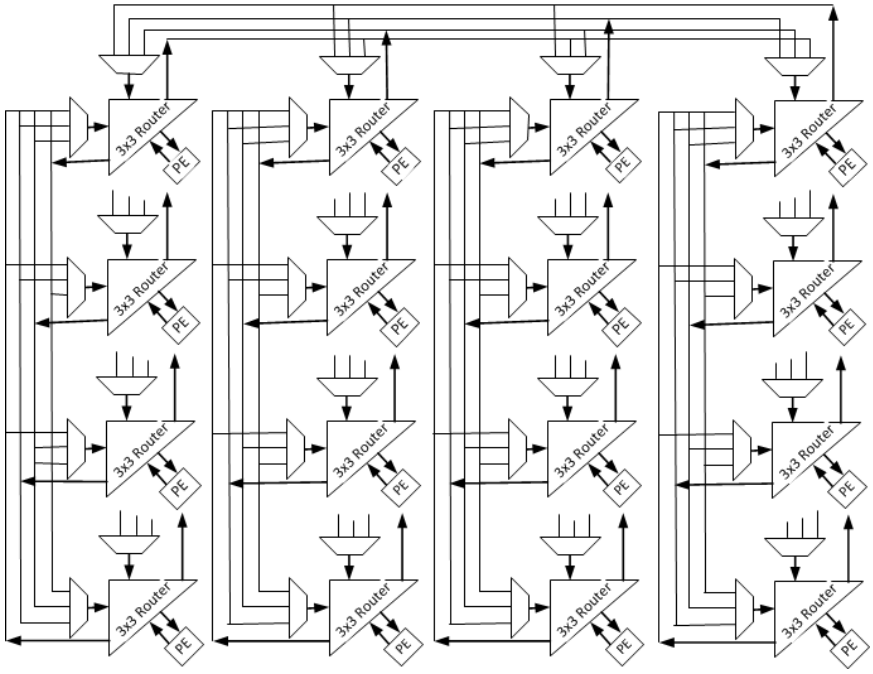


Figure 6.5: Hypermesh with 16 nodes using distributed crossbar switches as hyperedges (Triangles = 3x3 buffered inputs Router, Squares= PEs).

In a Hypermesh with N nodes as shown in figure 6.5, they are $2\sqrt{N}$ hyperedges in both dimensions. Each hyperedge can be implemented by a $\sqrt{N}$x$\sqrt{N}$ B&S crossbar switch. Totally there should be $2\sqrt{N}$ $\sqrt{N}$x$\sqrt{N}$ crossbar switches in the network for implementing row and column channels. Each node are connected to 2 crossbar switches in different dimension. Therefore, for connecting each node to the row and column hyperedges, there is a need of a *3x3* input-queued router. Hyperedges can be implemented with input buffers or without input buffers. In this thesis, the buffered hyperedge uses DFF registers as the input buffers for the hyperedge, and the capacity of each input buffer is 1 flit. Using DFF registers as input buffer for hyperedges is more cost-efficient than using EM blocks as DFFs in the existing LEs can be used and *no* more resources (LEs) are needed (DFFs are placed in the 3x3 routers by the Altera Quartus CAD tools). Therefore, the area usage of hypredges with input buffers and

without input buffers are equal.

The 2D Hypermesh network with N nodes and concentration M has M processors per router and one router per node as well as $2\sqrt{N}$ hyperedges. There are 2 main design options for interconnecting processors to the routers: (i) each of the M processors in a node has its own dedicated high-speed IO port on the local router, or (ii) all of the M processors in a node share Y high-speed IO ports on the local router, where $Y \leq M$. Either option can be analytically modeled; we will assume router option (ii).

The total area of 2D Hypermesh with N nodes is expressed by the following parametric equation:

$$\begin{aligned} A_{HM}(K, M, Y, W) &= N \times M \times C \\ &+ N \times A_{R,TDM}(3, W, K) \\ &+ N \times A_{Conc}(Y, M, W) \\ &+ 2\sqrt{N} \times A_{HE}(\sqrt{N}, W) \end{aligned} \tag{6.1}$$

where the first 3 terms come from Eq. 5.3 in chapter 5 and are related to the total area of a 2D Hypermesh topology with N nodes except the area of hyperedges (with router degree = 3, Datapath width = W, with K VCs per edge, with area of the processor core = C, with Concentration = M, and with number of injector and extractor channels = Y). The last term in Eq. 6.1 refers to the area of the hyperedges (A_{HE}). The area of a hyperedge constructed by B&S crossbar switch (A_{HE}) in a 2D Hypermesh with N nodes is given by:

$$A_{HE}(N, W) = N \times A_{Mux}(N - 1, 1, W) \tag{6.2}$$

where there are $\sqrt{N}$ *($\sqrt{N}$-1)-to-1* Muxes in a hyperedge.

6.3.3 Critical Path Delay

As stated in chapter 5, the maximum allowable frequency of a topology is mathematically modeled as the inverse of critical path delay in the topology. The delay of a 2D GHC using EM blocks as input buffers is expressed by Eq. 5.4. The delay of a 2D GHC using DFF registers as input buffers is also given by Eq. 5.5, where Z (the number of Node-Distance in the largest link connecting 2 nodes) is $\sqrt{N}-1$.

In 2D Hypermesh with unbuffered hyperedges, when EM blocks are used as input buffers, the critical path is defined as the delay of an input buffer, a crossbar switch, and a hyperedge. Let's assume that an Hypermesh has a 2D layout with a unity aspect ratio. Therefore, the delay of an Hypermesh is given by:

$$\begin{aligned} D_{HM-EM}(D,W) &= D_{S2}(3,W) + D_{EM} + D_{S2-HE}(\sqrt{N},W) \\ &+ D_{bus}(\sqrt{A_{HM}} - \sqrt{A_{HE}}) \end{aligned} \tag{6.3}$$

where D_{EM} is delay of an EM block, $D_{S2}(3,W)$ is the delay of an 3x3 router, $D_{S2-HE}(\sqrt{N},W)$ is the delay of a B&S switch design in an hyperedge, where $\sqrt{A_{HM}}$ is the length of the Hypermesh, and where $\sqrt{A_{HE}}$ is the length of the hyperedge. A hyperedge connects all the nodes in a row or column. We assume that a hyperedge has a 2D layout with a unity aspect ratio. The length of a wire connecting all the nodes in a row or column is approximated by the length of a 2D Hypermesh ($\sqrt{A_{HM}}$). Since the delay of a wire with length of $\sqrt{A_{HE}}$ is considered in the delay of the hyperedge, the delay of wire with length of an heyperedge ($\sqrt{A_{HE}}$) should be deducted

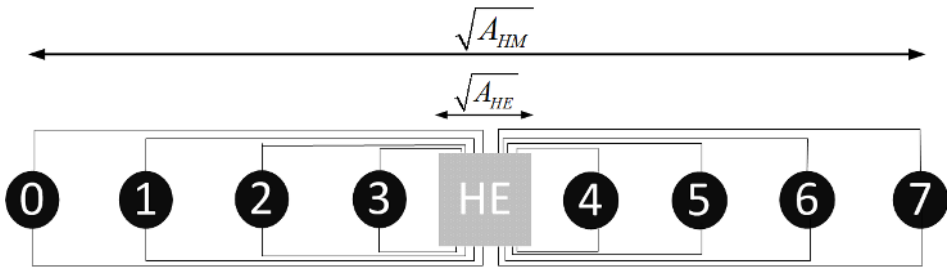


Figure 6.6: The wire length of a hyperedge ($\sqrt{A_{HE}}$) and the wire length of a 2D Hypermesh ($\sqrt{A_{HM}}$)

from the delay of a wire with length of the 2D Hypermesh ($\sqrt{A_{HM}}$) (See figures 6.6 and 6.7).

Delay of a 2D Hypermesh (with unbuffered hyperedges) using DFF registers as input buffers for the 3x3 routers is given by:

$$D_{HM-DFF}(D,W) = D_{S2}(3,W) + D_{S2-HE}(\sqrt{N},W) + D_{bus}(\sqrt{A_{HM}} - \sqrt{A_{HE}}) \quad (6.4)$$

where $D_{S2}(3,W)$ is the delay of an 3x3 router, $D_{S2-HE}(\sqrt{N},W)$ is the delay of a B&S crossbar switch in an hyperedge. In Eq. 6.4, Since the delay of a wire with length of $\sqrt{A_{HE}}$ is considered in the delay of the hyperedge, the wire with length of an heyperedge ($\sqrt{A_{HE}}$) should be deducted from the wire with length of the 2D Hypermesh ($\sqrt{A_{HM}}$) (See figures 6.6 and 6.7).

6.4 Analytic Power Models

As stated in chapter 5, the power of a graph-based NoC operating in steady-state is modelled by 3 components, the power consumed in the switches, input buffers, and wires, as shown in Eq. 5.6 .

The maximum power consumption of a 2D GHC NoC with N nodes is obtained by Eq. 5.7.

The expected power consumed by a 2D Hypermesh can be obtained by substituting $E[P_{wires}]$ by $E[P_{HE}]$ in Eq. 5.6. The maximum power consumption of a 2D Hypermesh NoC with N nodes is given by:

$$Max(P_{HM})(N,W) = N.P_{switch} + 2\sqrt{N}.P_{HE}(\sqrt{N},W) + N.N_{Port}.P_{buffer} \qquad (6.5)$$

where P_{switch}, P_{HE}, and P_{buffer} are the average power consumed by a 3x3 switch, a hyperedge, and an input buffers assuming 100% loaded, and where N_{Port}=3 is the degree of 3x3 router.

In the 'pseudo random walk' model, the unbuffered *DCS* is used as hyperedges. In a 2D Hypermesh constructed by unbuffered *DCS*s with N nodes (as shown in figure 6.5), the average power consumed by an unbuffered hyperedge (which is 100% loaded) is given by:

$$P_{HE}(\sqrt{N},W) = \sqrt{N}.P_{Mux}(\sqrt{N}-1,1,W) + \sqrt{N}.P_{Bus}(\sqrt{A_{HM}},W,\sqrt{N}-1) \quad (6.6)$$

where the second term is the power consumed in the $\sqrt{N}$ broadcast buses, which is found by Eq. 4.10. Note that in a 2D Hypermesh with N nodes, there are $\sqrt{N}$ *($\sqrt{N}$-1)-to-1* Muxes.

In the wormhole-switched Hypermesh, Altera Quartus CAD tool will lay out the hyperedges as the centralized switches (as shown in figure 6.4).

In wormhole model of Hypermesh, we assumed that each hyperedge is a $\sqrt{N}\times\sqrt{N}$ router. Figure 6.7 shows an Hypermesh NoC constructed by centralized switches

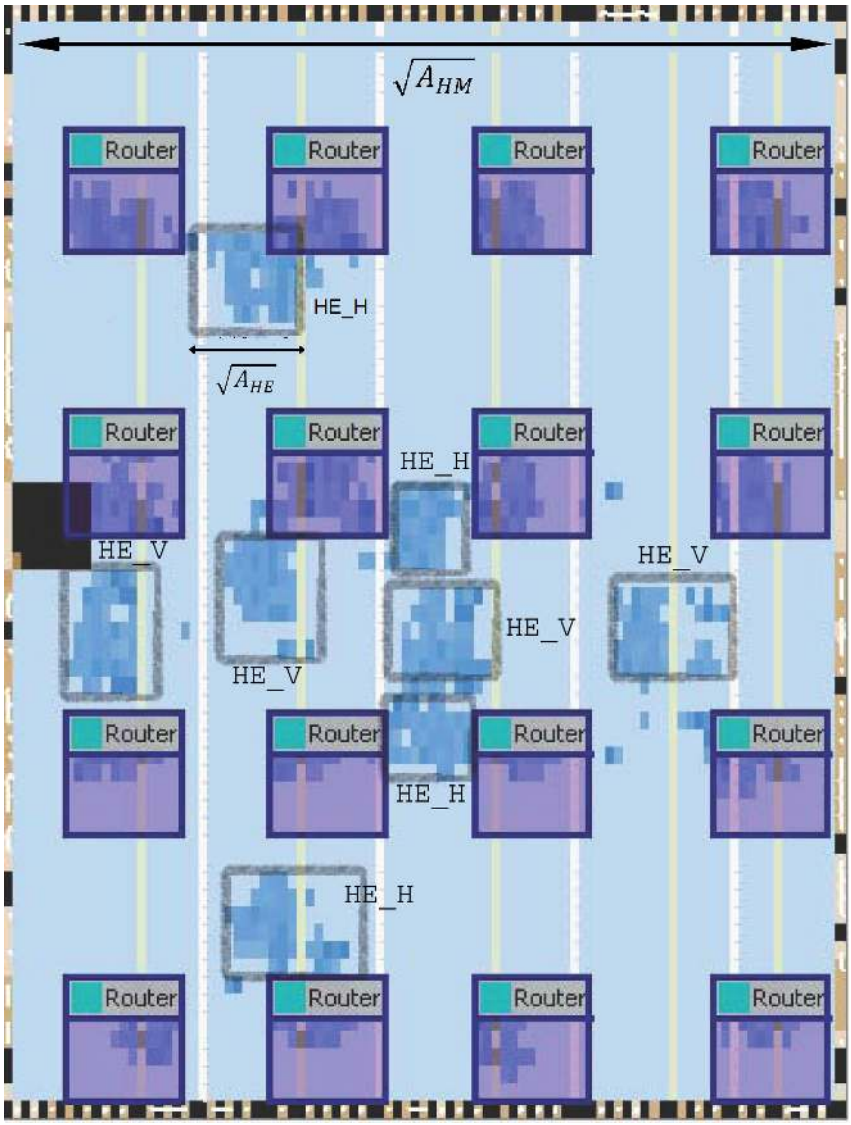


Figure 6.7: An Hypermesh NoC after Synthesis and Placement in an Altera FPGA (HE denotes hypergraph). The light gray boxes are the crossbar switches (HEs), which have been centralized by the CAD tool.

(as the hyperedges) after synthesis, placement, and optimization. The square '*HE*' denotes hyperedges (H=Horizental, V=Vertical hyperedge). Every hyperedge has an arbitration and routing unit. The Altera Quartus CAD tools will lay out the hyperedge with an arbitration and routing unit using the centralized crossbar switches (regardless of input buffers).

Let $\Gamma_{HM} = \sqrt{\frac{A_{HM}}{N}}$ be an Node-Distance in 2D Hypermesh with N nodes and $K = \sqrt{N}$ be the number of nodes in each dimension. The power consumption of an input-queued hyperedge includes the power consumed in K input buffers, an KxK B&S crossbar switch, and *2.K* wires connecting 3x3 routers in a row or column to the hyperedge.

Synthesizer always tries to minimize the length of wires in the hyperedges to reduce the delay and power consumption. To minimize the wire length, the synthesizer places the centralized B&S crossbar switches of hyperedges in the centre of each row and column as shown in figure 6.7. Figure 6.7 illustrates the placement of the hyperedges (after synthesis, placement, and optimization) in each row and column. As seen in figure 6.7, the hyperedge switches are approximately placed in the middle of each row and column. We assume that every hyperedge has a 2D layout with a unity aspect ratio (as approximately shown in figure 6.7). This implementation is similar to the figure 6.4.

To determine the length of *2.K* wires connecting 3x3 routers to a hyperedge, we assume that the hyperedge is placed in the centre of a row/column as shown in figure 6.4. Now we can estimate the total length of *2.K* wires. For ease of understanding, we currently ignore the area of the hyperedges (We will consider the area of the hyperedges later).

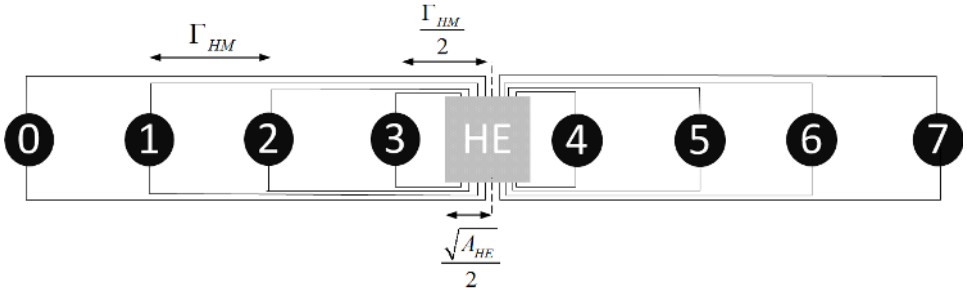


Figure 6.8: An example of a hyperedge connecting 8 routers in a row

In a 2D Hypermesh with $K = \sqrt{N}$ nodes in each dimension, the node position in a row is defined as $p = 0, ..., k-1$. Let i and j represent the number of Node-distances ($\Gamma_{HM} = \sqrt{\frac{A_{HM}}{N}}$) in the left and right directions respectively. The total wire distance connecting the output ports of the *3x3 routers* in a row/column to the input ports of a hyperedge is given by (expressed in terms of the Node-distance):

$$\begin{aligned} \sum_{i=0}^{\frac{K}{2}-1}(i+\frac{1}{2}) + \sum_{j=0}^{\frac{K}{2}-1}(j+\frac{1}{2})) &= 2 \times \Big(\sum_{i=0}^{\frac{K}{2}-1}(i+\frac{1}{2})\Big) \\ &= 2 \times \Big(\frac{(\frac{K}{2}-1).(\frac{K}{2})}{2} + (\frac{K}{2}) \times \frac{1}{2}\Big) \\ &= \frac{K}{2} \times \frac{K}{2} \end{aligned} \tag{6.7}$$

Figure 6.8 shows an example for a centralized hyperedge connecting 8 routers. For example, the length of an I/O wire connecting the output of the router 0 to the input port of the hyperedge is $\frac{7}{2}.\Gamma_{HM}$.

Using the same methodology, the total wire distance connecting the output ports of a hyperedge to the input ports of the *3x3 routers* in a row/column is equal to Eq. 6.7. Therefore, the total wire distance required (expressed in terms of the Node-Distance) to connect a hyperedge to the all *3x3 routers* in a row/column is given

by:

$$L = 2 \times (K/2) \times (K/2) = K \times (K/2) \tag{6.8}$$

The Eq. 6.8 includes the length of a hyperedge as well (as shown in figure 6.7). To accurately determine the length of wires connecting a hyperedge to the all *3x3 routers* in a row/column, the length of a hyperedge must be considered in Eq. 6.8. As we assume that hyperedges have a 2D layout with unity aspect ratio, the length of a hyperedge is equal to $\sqrt{A_{HE}}$. For every I/O wire connecting a hyperedge to a *3x3 routers*, $\approx \frac{\sqrt{A_{HE}}}{2}$ wire length should be removed (See figures 6.7 and 6.8). Therefore, the total wire distance (expressed as multiples of LE-Length) connecting an hyperedge to 3x3 routers in one row or column is equal to $K \times (K/2) \times \Gamma_{HM} - 2K \times \frac{\sqrt{A_{HE}}}{2}$.

The average length of *2K* wires connecting a hyperedge to all the 3x3 routers in a row or column (expressed as multiples of LE-Length) is given by:

$$\hat{L} = \frac{K \times (K/2) \times \Gamma_{HM} - K \times \sqrt{A_{HE}}}{2K} \tag{6.9}$$

where the dominator (*2K*) is the number of wires connecting a hyperedge to the routers in a row/column.

As stated earlier, the power consumption of a *centralized* input-queued hyperedge (as shown in figure 6.7) includes the power consumed in K input buffers, an KxK B&S crossbar switch, and *2.K* wires connecting 3x3 routers in a row or column to the hyperedge.

In total, the analytic power model for an KxK *centralized* hyperedge including its

input buffers (which is 100% loaded) is given by:

$$\begin{aligned} P_{HE}(K,W) &= P_{S2}(K,W) + K.P_{IB} \\ &+ 2.K.P_{Bus}(\hat{L},W,1) \end{aligned} \tag{6.10}$$

where $\hat{L}$ is the average average length of *2K* wires connecting a hyperedge to all the 3x3 routers in a row or column, and is given by Eq. 6.9.

The power consumption in a centralized hyperedge *without* input buffers is obtained by modifying Eq. 6.10 (in Eq. 6.10, the power consumption of the input buffers is removed).

6.5 Analytic Power Model for Traffic Patterns

In this section, a power analysis of the 2D GHC and Hypermesh under 3 deterministic traffic patterns will be studied; (1) the RU traffic pattern, (2) the traffic pattern in the Bitonic Sorting algorithm, and (3) the traffic pattern in the Cooley-Tukey FFT algorithm.

For all power analyses in this section, the following assumptions are made:

1. The NoC uses a deterministic ordered-dimension XY routing algorithm, and packets are delivered along minimum hops paths.

2. The loads are evenly distributed over the nodes.

3. Packets are assumed to have a fixed length of m flits.

4. PEs generate traffic independently of each other. The traffic follows a Poisson

process with mean rate of λ_P per node per clock cycle. The flit arrival rate can be estimated as $\lambda_F = \lambda_P \times m$ per node per clock cycle.

To find the power consumption of a 2D GHC in the wormhole-switched model, Eq. 5.8, 5.9, 5.10, 5.11, and 5.12 from chapter 5 are used.

In the following subsections, the analytic power model for 2 wormhole-switched Hypermeshes are studied; the Hypermesh using (1) hyperedges with input buffers (*HM-Buf*) where *DFF registers* are used as the input buffers of hyperedges and the buffer capacity is *one* flit, and (2) the Hypermesh using hyperedges without input buffers (*HM*).

As stated previously, the analytic power model for *HM-Buf* and *HM* are identical, except that in hyperedges without input buffers (*HM*) the power consumption of input buffers in Eq. 6.10 is removed.

6.5.1 Power Model in Wormhole Switching

Let suppose that each PE generates a flit with rate λ_F per node per clock cycle. Let λ'_F be the effective arrival rate, which can be found by Eq. 5.8. The total effective arrival rate of flits to the Hypermesh NoC is $N \times \lambda'_F$. Based on the assumption that the loads are uniformly distributed across the network, the effective arrival rate of flits to the network ($N \times \lambda'_F$) are distributed among the hyperedges. As stated before, the number of hyperedges in a Hypermesh with N nodes is $2 \times K$, where $K = \sqrt{N}$. Each active IO port pair in a hyperedge consumes a fraction $\frac{1}{K}$ of the fully-loaded hyperedge of degree K. Let's suppose that in wormhole switching a flit traverses H hops. Therefore, the effective flit arrival rate to one IO port of a hyperedge is estimated as the effective arrival rate of flits to the network ($N \times \lambda'_F$) weighted by the

number of hyperedges traversed by a packet on average (H) divided by the number of hyperedges ($2K$) times the number of ports per hyperedge (K).

$$\lambda_{HE} = (N \times \lambda'_F) \times \frac{H}{2 \times K} \times \frac{1}{K} = \lambda'_F \times \frac{H}{2} \tag{6.11}$$

where H is the average number of hops traversed by a packet in a given traffic pattern.

Power consumption in an Hypermesh NoC is expressed by the power consumed in switches, input buffers, and hyperedges. Given the traffic patterns, the analytic power model for a 2D Hypermesh NoC can be approximated by modifying Eq. 6.5 to reflect the reduced load (effective load) of each component, and is given by:

$$P_{HM} = \lambda_{Xbar}.N.P_{Switch} + \lambda_{IB}.N.N_{Port}.P_{IB} + \lambda_{HE}.2.K.P_{HE} \tag{6.12}$$

where λ_{Xbar}, λ_{IB}, and λ_{HE} are the effective flit arrival rate for a switch, input buffer, and hyperedge and are determine by Eq. 5.10, 5.11, 6.11 respectively. N_{Port}=3 is the degree of the 3x3 routers. P_{HE}, P_{Switch}, and P_{IB} are the average power consumption in a hyperedge, switch, and input buffer receptively (when they are 100% loaded).

6.5.2 Wire Lengths and Hop Count

To determine the power consumption of NoC topologies under a given traffic pattern, we need to determine (1) the expected distance traversed by a packet (L_{X+Y}) in the given traffic pattern, and (2) the expected number of hops H per packet in the given traffic patterns. Three traffic patterns are explored; (1) Random Uniform traffic, (2) the traffic pattern in the Cooley-Tukey FFT Algorithm, and (3) the traffic pattern in the Bitonic sorting algorithm.

We also use the list of definitions for each symbol in Table 5.2.

6.5.3 Random Uniform Traffic

For the RU traffic pattern, each node is equally likely to send the packet to all nodes.

GHC

Consider a 2D GHC topology with N nodes placed in a square shape, as shown in figure 6.1. Each row or column has K=$\sqrt{N}$ nodes. To determine the expected wire distance traversed by a packet in the X dimension, we use the theorem 3.

Theorem 3: In a 2D GHC with the physical layout shown in figure 6.1, where packets are constrained to follow minimum hop and minimum distance paths, the expected distance traversed by a packet in the X dimension under the RU traffic pattern, expressed as multiples of the Node-Distance Γ_{GHC}, is given by:

$$E[L_X] = \frac{2}{K^2} \times \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \Big(K - (2i-1) \Big)^2 \tag{6.13}$$

Proof. (Theorem 3). The proof is established by enumeration. Consider any node p in a row in the X dimension for $0 \leq p \leq K-1$. The traffic leaving node p is uniformly distributed over the K nodes in the row. Let i and j represent the number of edge traversals in the left and right directions respectively. The expected distance traversed in the X dimension by traffic leaving node p is given by:

$$\frac{1}{K} \Big(\sum_{i=0}^{p} i + \sum_{j=0}^{K-p-1} j \Big) \tag{6.14}$$

By taking the expectation for nodes $0 \leq p < K$ and mathematically rearranging the

terms, the result is obtained. □

By symmetry, the expected distance traversed by a packet in vertical (Y) dimension is given by:

$$E[L_Y] = \frac{2}{K^2} \times \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \Big(K - (2i-1) \Big)^2 \tag{6.15}$$

Based on Theorem 3, the total expected distance traversed by a packet in a 2D GHC in both dimensions under the RU traffic pattern, expressed as multiples of the Node-Distance Γ_{GHC}, is given by:

$$E[L_{X+Y}] = \frac{4}{K^2} \times \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \Big(K - (2i-1) \Big)^2 \tag{6.16}$$

In 2D GHC, every node sends a packet to every other nodes in same row and column by an edge traversal. If the source and destination of a packet are in a row or column, the average number hops traversed by a packet is 1. In 2D GHC, every node has $2(\sqrt{N}-1)$ connections to other nodes. If the source and destination of a packet are not in a same row or column, the average number of hops traversed by a packet is 2. Therefore, the expected number of hops traversed by one packet is given by:

$$\begin{aligned} E[H_{GHC}] &= \frac{(2(\sqrt{N}-1) \times 1 + (N - 2(\sqrt{N}-1) - 1) \times 2}{N} \\ &= 2 - \frac{2\sqrt{N}}{N} \end{aligned} \tag{6.17}$$

These results are used in Eq. 5.13 to compute the 2D GHC power.

Hypermesh

Consider 2D Hypermesh in figure 6.4. If the source and destination of a packet are in a row or column, the average number hyperedges traversed by a packet is 1. If the source and destination of a packet are not in a same row and column, the average number of hyperedges traversed by a packet is 2. Hence, the expected number of hyperedges traversed by one packet is given by:

$$\begin{aligned} E[H_{HM}] &= \frac{(2(\sqrt{N}-1)\times 1 + (N-2(\sqrt{N}-1)-1)\times 2}{N} \\ &= 2 - \frac{2\sqrt{N}}{N} \end{aligned} \tag{6.18}$$

The results of Eq. 6.18 and Eq. 6.11 are used in Eq. 6.12 to compute the Hypermesh power.

6.5.4 Bitonic Sorting Algorithm

The Bitonic sorting algorithm is shown in figure 6.9. The Bitonic sorting algorithm consists of $\frac{\log_2 N.(\log_2 N+1)}{2}$ butterfly permutations (denoted E^j for $0 \leq j < \log_2 N$). In the following, we compute the expected distance and average number of hops traversed by a packet in the Bitonic sorting algorithm.

GHC

Assume the 2D GHC layout as shown in figure 6.1, and let $n = \log_2 N$. Each packet in a butterfly permutation E^j for $0 \leq j < n/2$ traverses one edge with distance 2^j, as stated in Property 2 in chapter 5. Each packet in a butterfly permutation E^j for $n/2 \leq j < n$ traverses one edge with distance $2^{(j-n/2)}$, as stated in Property 3 in

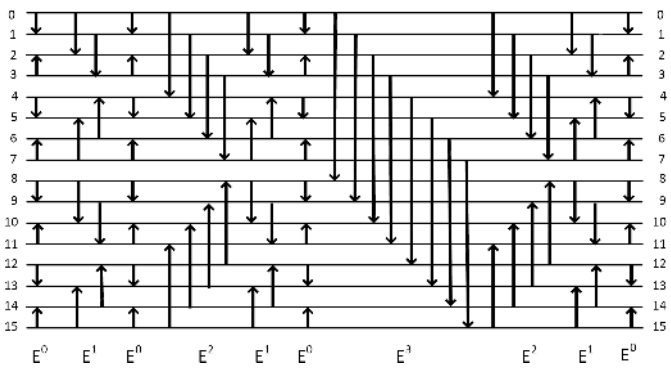


Figure 6.9: Bitonic sorting algorithm with 16 nodes

chapter 5. Therefore, the expected distance traversed by one packet in the Bitonic sorting algorithm in the GHC and 2D Mesh topologies are equal and is given by Property 6:

Property 6: In a 2D GHC with the layout in figure 6.1, where packets are constrained to follow minimum hop and minimum distance paths, the expected distance traversed by each packet in the X and Y dimensions under the Bitonic sorting algorithm, expressed as multiples of the Node-Distance Γ_{GHC}, is given by:

$$E[L_{X+Y}] = \frac{2}{n.(n+1)}.(\sum_{j=0}^{n/2-1}(n-j)\times 2^j + \sum_{j=n/2}^{n-1}(n-j)\times 2^{j-n/2}) \tag{6.19}$$

where the $(n-j)$ denotes the number of E^j permutations per node in Bitonic sorting algorithm, as shown in figure in figure 6.9. In the 2D GHC, each packet in a butterfly permutation E^j for $0 \leq j < n$ is realized by one edge traversal. The average number of hops (H) traversed by one packet under Bitonic traffic pattern is given by:

$$E[H_{GHC}] = 1 \tag{6.20}$$

These results are used in Eq. 5.13 to compute the GHC power given the Bitonic sorting algorithm.

Hypermesh

In a 2D Hypermesh, each packet in a butterfly permutation E^j for $0 \leq j \leq n-1$ traverses one hyperedge. Therefore, the average number of hops traversed by a packet under Bitonic sorting algorithm is given by:

$$E[H_{HM}] = 1 \tag{6.21}$$

The results of Eq. 6.21 and Eq. 6.11 are used in Eq. 6.12 to compute the Hypermesh power.

6.5.5 Cooley-Tukey FFT Algorithm

The Cooley-Tukey FFT algorithm is shown in figure 5.4. The FFT algorithm consists of n Butterfly permutations (denoted E^j for $0 \leq j < n$) followed by a BR permutation. In the following, we compute the expected distance and average number of hops traversed by a packet in the FFT algorithm.

GHC

Referring to the GHC in figure 6.1, each butterfly permutation E^j for $0 \leq j < n$ is realized by one edge traversal. Therefore, the expected number of hops traversed by

Table 6.1: Average number of hops and distances under BR permutation

		Network Size		
Topology	Results	16	64	256
GHC (Fig. 6.1)	Hops $E[H_{BR}]$	1.5	1.75	1.875
	Distance $E[L_{BR}]$	2.5	5.25	10.62
Hypermesh	Hops $E[H_{BR}]$	1.5	1.75	1.875

one packet under FFT algorithm is given by:

$$E[H_{GHC}] = \frac{\sum_{j=0}^{n-1} 1 + E[H_{BR}]}{n+1} = \frac{n + E[H_{BR}]}{n+1} \tag{6.22}$$

where $E[H_{BR}]$ is the expected number of hops traversed by a packet in the 2D GHC under the BR permutation.

Table 6.1 shows the results for $E[L_{BR}]$ and $E[H_{BR}]$ in the BR permutation, which are determined experimentally in a Matlab program. According to Table 6.1, in a 2D GHC the expected number of hops per packet in the BR permutation ($E[H_{BR}]$) is exactly equal to the expected hops per packet in the RU traffic pattern, which is given in Eq. 6.17. Similarly, $E[L_{BR}]$ in 2D GHC is exactly equal to the expected distance per packet in the RU traffic pattern, given in Eq. 6.16. According to Matlab simulations, the same equalities holds for the 2D Hypermesh.

Referring to the GHC in figure 6.1, each packet in a butterfly permutation E^j for $0 \leq j < n/2$ traverses one edge with distance 2^j, as stated in Property 2 in chapter 5. Each packet in a butterfly permutation E^j for $n/2 \leq j < n$ traverses one edge with distance $2^{(j-n/2)}$, as stated in Property 3 in chapter 5. Therefore, the expected distance traversed by one packet under the FFT algorithm in the GHC and 2D Mesh topologies are equal and is given by Property 7.

Property 7: In a 2D GHC with the layout in figure 6.1, where packets are

constrained to follow minimum hop and minimum distance paths, the expected distance traversed by each packet in the X and Y dimensions under the FFT algorithm, expressed as multiples of the Node-Distance Γ_{GHC}, is given by:

$$E[L_{X+Y}] = \frac{1}{(n+1)} * (2.(2^{(n/2)} - 1) + E[L_{BR}]) \tag{6.23}$$

where $E[L_{BR}]$ is the expected distance traversed by each packet in the BR permutation, which is exactly equal to the expected distance per packet in the RU traffic pattern, given in Eq. 6.16.

These results are used in Eq. 5.13 to compute the GHC power.

Hypermesh

In 2D Hypermesh as shown in figure 6.4, each packet in a butterfly permutation E^j for $0 \leq j \leq n-1$ traverses one hyperedge traversal. The expected number of hops traversed by a packet under FFT algorithm is given by:

$$E[H_{HM}] = \frac{\sum_{j=0}^{n-1} 1 + E[H_{BR}]}{n+1} = \frac{n + E[H_{BR}]}{n+1} \tag{6.24}$$

where $E[H_{BR}]$ is the expected number of hops traversed by a packet under the BR permutation in the Hypermesh NoC, which is exactly equal to the expected distance per packet in the RU traffic pattern, given in Eq. 6.18.

Table 6.2: Average number of hops traversed by a packet for NoC topologies

		Network Size		
Topology	Traffic Pattern	16	64	256
Hypermesh	RU	1.5	1.75	1.875
	Bitonic	1	1	1
	FFT	1.1	1.1	1.1
GHC	RU	1.5	1.75	1.875
	Bitonic	1	1	1
	FFT	1.1	1.1	1.1
BHC	RU	2	3	4
	Bitonic	1	1	1
	FFT	1.2	1.28	1.33

6.6 Results

In this section, analytic and experimental results of different topologies, i.e. the 2D Hypermesh, 2D GHC, and 2D BHC, are studied. According to chapter 5, the BHC outperforms 2D Mesh and Torus, and is a good candidate for comparison in this chapter. The results of BHC come from chapter 5. The resource usage reported for the NoCs includes the resource usage of switches and input buffers in terms of LEs. The additional resource usage by the processors is not considered. In our experiments, each NoC is clocked at the same clock frequency $f_{clk} = 50$ MHz. For the analytic results, we assume that the clock frequency of 50 MHz is feasible for all the topologies. The additional power consumed by the processors is not modelled, to focus on the power used in the NoC topology.

Table 6.2 shows the expected number of hops traversed by a packet under various traffic patterns, for NoCs with different sizes. As seen in this table, the GHC and Hypermesh have the the lowest number of hops.

Table 6.3: Analytic and Experimental results for the topologies (Input Buffers=EM blocks, Pseudo random walk model, W= 16)

NoC	Switch Design	Anlyt. Area (LE)	Exp. Area (LE)	Area Error (%)	Anlyt. Power (mW)	Exp. Power (mW)	Power Error (%)	Anlyt. Fmax (MHz)	Exp. Fmax (MHz)	Fmax Error (%)
HM	B&S	1984	1984	0%	95.22	92.59	3%	150	135	11%
GHC	B&S	7616	7616	0%	281.22	263.52	7%	130	132	1%
BHC	B&S	2880	2880	0%	158.05	149.89	6%	141	143	1%

6.6.1 Experimental Results

To validate our analytic models, we compare analytic and experimental results of the topologies under the 'pseudo random walk' and the wormhole model. In the pseudo random walk model we assume the *DCS* Hypermesh with *unbuffered* hyperedges. For all experimental results, the number of nodes is 16, the datapath width is 16 bits, and clock frequency is 50MHz. The capacity of input buffers are 4 flits and each flit is 16 bits.

Table 6.3 and 6.4 show the experimental and analytic results for the 3 topologies with 16 nodes under the pseudo random walk model. In table 6.3 the input buffers are built with EM blocks, while in table 6.4 the input buffers are built with DFFs. The datapath width is 16 bits. In these tables, area, power and maximum working frequency (at 0^oC and 1.2V model) of 2D BHC, GHC, and Hypermesh are studied using the pseudo random walk model, assuming all data widths W = 16 bits. . The power results in these 2 tables is the peak power, where links, hyperedges and routers are 100% loaded. Comparison between analytic and experimental results shows an excellent match, on average with 6% error.

As illustrated in Tables 6.3 and 6.4, realizing the input buffers using EM will reduce the cost (in LEs) and will require fewer memory bits (in terms of LEs), but

Table 6.4: Analytic and Experimental results for the topologies (Input Buffers= DFFs, Pseudo random walk model, W=16)

NoC	Switch Design	Anlyt. Area (LE)	Exp. Area (LE)	Area Error (%)	Anlyt. Power (mW)	Exp. Power (mW)	Power Error (%)	Anlyt. Fmax (MHz)	Exp. Fmax (MHz)	Fmax Error (%)
HM	B&S	5632	5119	10%	69.07	64.51	7%	219	226	3%
GHC	B&S	16128	15054	7%	217.65	213.38	2%	195	191	2%
BHC	B&S	8960	8826	2%	112.09	112.01	1%	251	246	2%

it will decrease the maximum frequency. This happens since (i) the EM blocks are built by SRAM cells and the access time to SRAM cells is more than that for register buffers, and (ii) the placement of EM blocks are fixed within the FPGA floorplan, which will increase the delay of the critical path.

Table 6.5 illustrate the comparison between analytic and experimental power results for 2D Hypermeshes, BHC, and GHC under RU traffic pattern, traffic pattern in Bitonic sorting algorithm, and FFT algorithm in wormhole model, assuming all data widths W = 16 bits.

As seen in table 6.5, there are 2 types of Hypermeshes; (1) the Hypermesh using hyperedges with input buffers (*HM-Buf*) where *DFF registers* are used as the input buffers of hyperedges and the buffer capacity is *one* flit, and (2) the Hypermesh using hyperedges without input buffers (*HM*).

In all power analysis in wormhole-switched NoCs, it is assumed that the packet injection rate is *0.0078* packet/node/clock cycle and the length of a packet is *m=32* flits. We analyse the experimental results using Batch-Mean method. Each simulation is divided to 8 batches and the first batch is ignored as a warm-up period. Each batch includes 6000 clock cycles. The comparison between analytic and experimental power results in table 6.5 shows an acceptable agreement, typically 8%.

Table 6.5: Analytic and experimental power results for wormhole-switched NoCs with 16 nodes (Input buffer= EM blocks, W= 16, Freq.=50 MHz)

NoC	Traffic Pattern	Analytic Power (mW)	Experimental Power (mW)	Error (%)
HM-Buf	RU	20.64	19.43	6%
	FFT	16.85	15.72	9%
	Bitonic	15.9	14.73	8%
HM	RU	19.76	18.72	6%
	FFT	16.21	14.86	9%
	Bitonic	15.32	14.3	7%
BHC	RU	23.46	22.11	7%
	FFT	17.08	15.94	7%
	Bitonic	15.45	14.16	9%
GHC	RU	24.55	22.93	7%
	FFT	20.35	18.64	9%
	Bitonic	19.18	17.59	9%

6.6.2 Evaluation Methodology of the NoCs

In this sub-section (1) the NoCs are compared based on their area, and energy consumption, and then, (2) the NoCs are evaluated by their *energy-area product* which reflects *cost* and *performance* metrics of the NoCs.

To have a fair comparison between the NoC topologies, each NoC is normalized to have an equal bisection bandwidth of O(N) bits/sec. We set the datapath width of GHC (W_{GHC}) to 16 bits, and adjust the widths of the BHC and Hypermesh as described in chapter 2. For example, for N=16 nodes, $W_{GHC} = 16$, $W_{BHC} = 32$, and $W_{Hypermesh} = 32$ bits.

Area

Figure 6.10a and 6.10b illustrate area comparison of the 3 topologies with 256 nodes under equal bisection bandwidth. The resource usages are in terms of Logic Elements

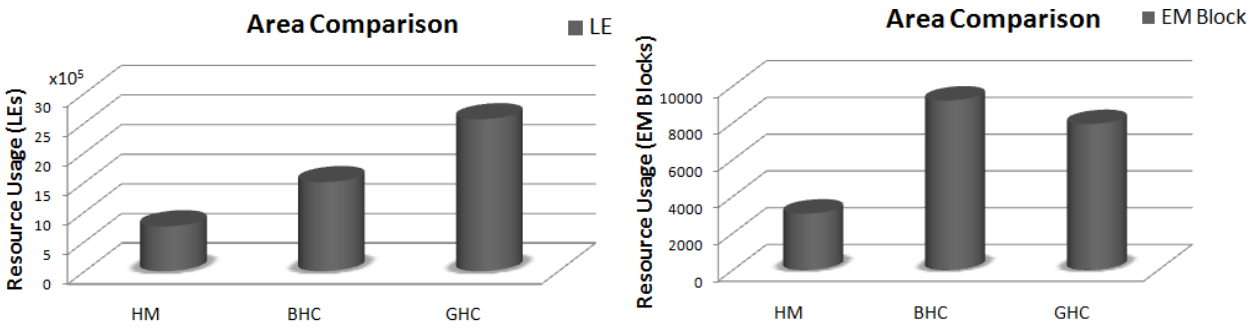


Figure 6.10: Area analysis of the NoCs with 256 nodes a) in terms of LEs b) in terms of EM blocks (With Equal Bisection Bandwidth, Switch=S1 (B&S1))

.

(LEs) and EM blocks. For N=256 nodes and with equal bisection bandwidth, each router in the GHC, BHC, and Hypermesh uses 31, 36, and 12 EM blocks respectively. Based on our assumption that the buffers in hyperedges are *one* flit, the area usage of *HM* and *HM-Buf* is equal. The buffers in the hyperedges do not require any more area, since they are constructed by DFFs and they use the unused DFFs in the 3x3 routers.

As seen in figures 6.10a and 6.10b, the 2D Hypermesh outperforms the 2D BHC and GHC in terms of LEs and EM blocks with significant margin. Under equal bisection bandwidth, the area usage in terms of LEs of the 2D Hypermesh with 256 nodes is 30% of the area of the GHC and 50% of the area of the BHC. Under equal bisection bandwidth, the area usage (in terms of EM blocks) of the 2D Hypermesh with 256 nodes is 38% of the area of the GHC and 33% of the area of the BHC.

Energy Per Algorithm

In this sub-section the NoCs are evaluated based on their energy consumption. Two parallel algorithms are explored; (1) the Bitonic sorting algorithm, and (2) the FFT algorithm.

The completion time of an NoC (expressed in terms of ns) given an algorithm is expressed by:

$$T_{Algorithm} = EX_{Algorithm} \times T_{Clock} \tag{6.25}$$

where T_{Clock} is the minimum clock period ($1/F_{max}$) that a network achieves, and where $EX_{Algorithm}$ is the execution time (in terms of clock cycles) to perform the given algorithm. The $EX_{Algorithm}$ (in terms of clock cycles) is given by:

$$EX_{Algorithm} = T_P \times N_{Perm} + T_C \times N_{Perm} \tag{6.26}$$

where T_P is the average network latency for a packet, and where N_{Perm} is the number of packet transmissions in a given algorithm. For example in the FFT algorithm $N_{Perm} = \log_2 N + 1$, and in the Bitonic sorting algorithm $N_{Perm} = \log_2 N.(\log_2 N + 1)/2$. Referring to Eq. 6.26, T_C is the average time taken to perform a computation by a PE.

Let's suppose that a packet has m flits and a flit size is equal to the datapath width. The network latency of a packet in a wormhole-switched NoC with H hops is given by [54]:

$$T_P = H \times (T_{Routing} + T_{Xbar} + T_{Link}) + (m-1) \times (T_{Xbar} + T_{Link}) + Bl \tag{6.27}$$

where $T_{Routing}$, T_{Xbar}, and T_{Link} are the delay for the routing, crossbar switch, and link traversal respectively. Referring to Eq. 6.27, the term Bl is the average blocking time seen by a header flit [54]. In low traffic using very long length packets, the contribution of the mean blocking time Bl to the network latency of a packet is negligible and it can be ignored [54].

The energy consumption of a topology given an algorithm is expressed by:

$$E_{Algorithm} = P_{Algorithm} \times T_{Algorithm} \tag{6.28}$$

where the $P_{Algorithm}$ and $T_{Algorithm}$ are the average power consumption and completion time of the topology using the algorithm respectively.

Figure 6.12a and 6.12b show the energy consumption of the 2D Hypermesh (implemented with unbuffered hyperedges), BHC, and GHC under the Bitonic sorting algorithm and FFT algorithm (with equal bisection bandwidth). All these three topologies can implement all the butterfly permutations in Bitonic and FFT algorithms without conflict. In both algorithms each network has 256 nodes and each packet is 10 *Kbits*. Input buffers are DFFs and have a capacity of 4 flits. The flit injection rate is 0.25 flit per node per clock cycle. Today, the performance of most digital systems is limited by their communication rather than computation [13]. Therefore, in our analysis we assume that the computation time T_C is negligible compared to the network latency for a packet and $T_C \approx 0$ ns. In our analysis, the routing delay ($T_{Routing}$) is 3 clock cycles. Once the routing is completed, the delay in forwarding a flit is 1 clock cycle per flit.

Figure 6.11a and 6.11b show the completion time of the algorithms ($T_{Algorithm}$) on the 2D Hypermesh, BHC, and GHC with 256 nodes. As seen in figures 6.11a and

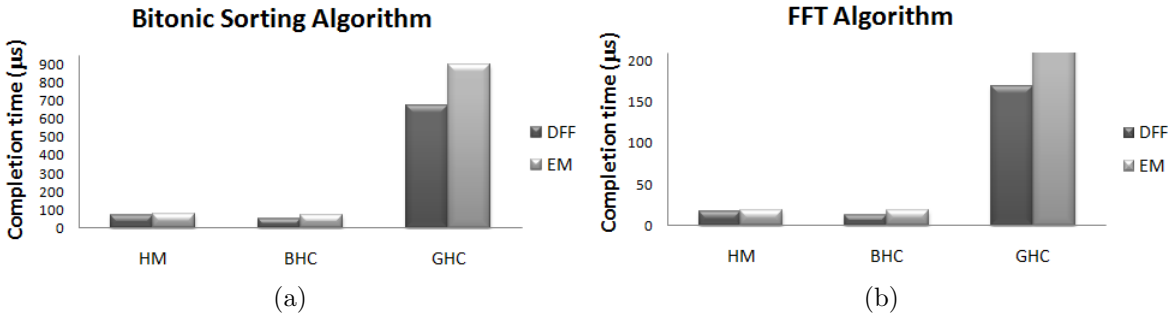


Figure 6.11: Completion time of the topologies with 256 nodes under a) Bitonic sorting algorithm b) FFT algorithm (under equal bisection bandwidth, Switch=S1 (B&S1), Pkt size=10Kbits)

.

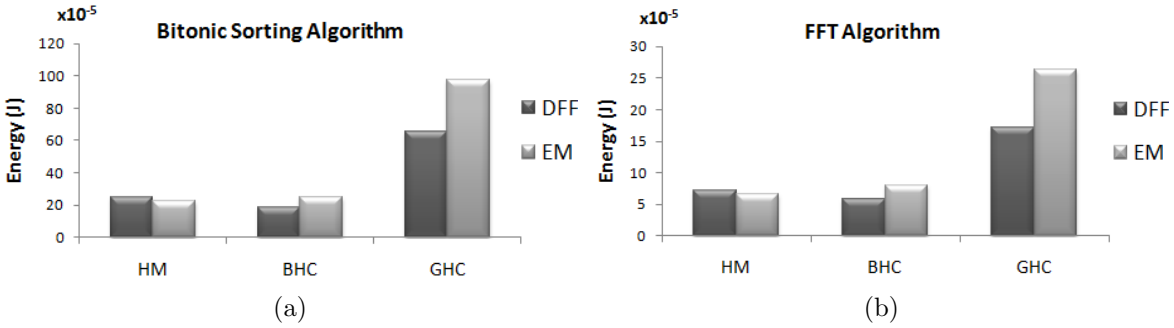


Figure 6.12: Energy analysis of the NoCs with 256 nodes under a) Bitonic sorting algorithm b) FFT algorithm (under equal bisection bandwidth, Switch=S1 (B&S1), Pkt size=10Kbits)

.

6.11b, the 2D Hypermesh and BHC have the lowest completion times under FFT and Bitonic algorithm. The completion time of the 2D BHC under the Bitonic sorting algorithm and FFT algorithm is approximately equal to the 2D Hypermesh since the interconnection network in the BHC is perfect for the butterfly permutations in the Bitonic sorting algorithm and FFT algorithm. However the performance of BHC is degraded significantly if the algorithm changes. In addition, the F_{max} of 2D BHC is approximately equal to the F_{max} of 2D Hypermesh.

Referring to figure 6.12a and 6.12b, under the FFT algorithm the 2D Hypermesh uses 32% of the energy that the 2D GHC uses and approximately equal to the energy that the 2D BHC uses. Under Bitonic sorting algorithm, the 2D Hypermesh uses 32% of the energy that the 2D GHC uses and 125% of the energy that the 2D BHC consumes. The 2D BHC consumes lower energy under the Bitonic algorithm as the inter-router links in the BHC are perfect for the butterfly permutations in the Bitonic sorting algorithm and FFT algorithm.

Energy-Area Product

The most important design metrics in NoC domain are area, power and delay (latency). NoC architectures are supposed to offer high performance while the area and power budgets are limited. Therefore, NoCs should be designed under very tight area and power budgets while considering performance as well. One of the useful metrics to evaluate the NoCs is *energy-area product*, which reflects both the *cost metrics* (Power and Area) and a *performance metric* (latency) of NoCs.

Figure 6.13a and 6.13b show the *energy-area product* of the topologies under the Bitonic sorting algorithm and FFT algorithm respectively (with equal bisection bandwidth). As seen in these figures, the 2D Hypermesh (implemented with unbuffered hyperedges) outperforms all the topologies in terms of *energy-area product* with a significant margin. The *energy-area product* of 2D Hyperemsh with 256 nodes under Bitonic sorting algorithm is 13% of 2D GHC, and 53% of 2D BHC. The *energy-area product* of the 2D Hyperemsh under the FFT algorithm is 13% of the 2D GHC, and 45% of the 2D BHC.

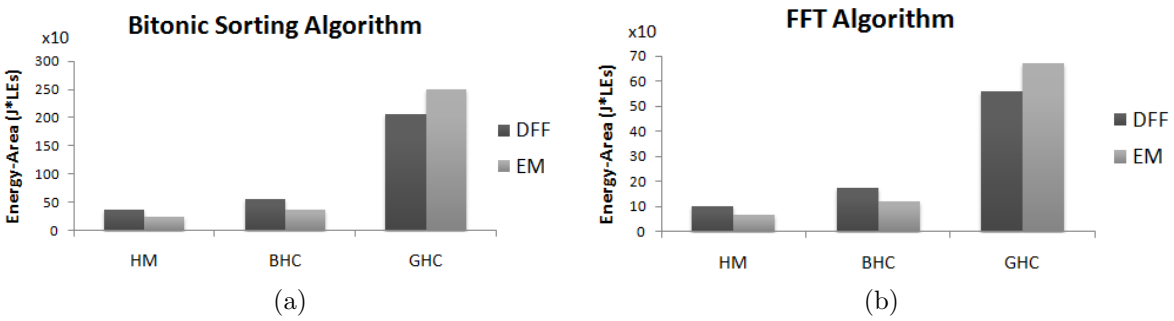


Figure 6.13: Energy-Area product of the NoCs with 256 nodes under a) Bitonic sorting algorithm b) FFT algorithm (under equal bisection bandwidth, Switch=S1 (B&S1), Pkt size=10Kbits)

.

6.7 Conclusions

Analytic models for the area, delay and power of 2D Hypermesh and GHC NoCs realized in the Altera Family of FPGAs, have been developed. The analytic models are accurate, have a theoretical basis and allow for the early design-space exploration of different NoC topologies, and for early design optimization. The analysis indicates that under the equal bisection bandwidth model, the 2D Hypermesh NoC will generally have considerably lower *energy-area product* compared to the graph-based NoCs including the 2D BHC and GHC, primarily due to the elimination of many buffers and large routers used in the NoCs. For example, the *energy-area product* of the 2D Hyperemsh under the FFT algorithm is 13% of the 2D GHC, and 45% of the 2D BHC. This conclusion indicates that the Hypermesh NoC topology offers considerable benefits over conventional graph-based NoCs. The CRAY TITAN supercomputer uses a 3D Hypermesh type of interconnection network. This chapter shows that the 2D Hypermesh is also very attractive in an NoC/SoC environment.

Chapter 7

Conclusion

FPGAs are considered as suitable platforms to accommodate System on Chip. As FPGAs capacity and capability grow, they can be increasingly used to build a wider range of SoCs. Advantages such as the increasing logic density with higher operating frequencies and a wide range of functional and memory blocks enable FPGAs to be a suitable replacement of ASICs in several high performance applications. NoCs can overcome the limitations of traditional bus-based and point-to-point on-chip communications used in SoCs. FPGA-based NoCs is a new field of research that takes the advantage of both FPGAs and NoCs characteristics. Therefore, exploration of the NoCs in FPGA needs more research in order to offer more efficient solutions for future designs. One of the methods to explore NoC designs in FPGAs is analytic models.

This thesis tries to analytically explore the design space of FPGA-based NoCs. In conclusion, we review the primary findings of this thesis.

7.1 Thesis Contributions

1. In this thesis, analytic models for basic components of routers and seven wormhole switched router designs in FPGA technology have been proposed. The analytic models are compared to extensive simulations and shown to be very accurate which normally are within 5-10% of error margin. These switches are compared in terms of area per throughput and power per throughput to choose the best switch for different applications. It is shown that Demux-Mux switch design has the lowest power per throughput, while Broadcast-and-Select designs offer the lowest area per throughput.

2. Based on the router analytic models, analytic models for 5 NoC topologies realized in FPGA technology have been proposed. These topologies are the 2D Mesh, the 2D Torus, the Binary Hypercube, the Generalized Hyperchube, and the hypergraph Hypermesh. The analytic models show an excellent agreement with simulation results, typically within 5-10% of error margin.

3. Analytic models for 3 traffic patterns are proposed, including the Random Uniform, the traffic pattern in the Bitonic Sorting algorithm, and the traffic pattern in the Cooley-Tukey FFT algorithm. These models are used to estimate and evaluate energy and power consumption of different topologies under these traffic patterns.

4. Novel analytic power models for wormhole-switched NoCs are proposed. The NoC analytic power models are compared to experimental results and shown to be within 8-12% error margin. The power model can be used for any specific algorithm, using the expected distance and number of hops traversed by a packet

in the algorithm.

5. Input buffers are one of the most important components in NoCs. In FPGAs, buffers can be constructed by either SRAM Embedded Memory blocks or DFFs. It has been observed that the use of Embedded Memory blocks as buffers significantly decreases the area used in the routers and NoCs compared to DFFs. However it minimizes the maximum allowable frequency due to the fix placement of the blocks in the FPGA die. Our analysis indicates that Embedded Memory blocks are more power efficient than DFFs when NoCs use multiple virtual channels per physical channel.

6. Conventional hyperedge designs use the Broadcast-and-Select switch design. To improve the performance of Hypermeshes, two new hyperedge designs are proposed, which are explained in Appendix C. The Hypermesh using the new hyperedge designs outperforms all Hypercubes significantly.

7. Our analysis indicates that under the equivalent bisection bandwidth, high-degree hypercubes and Hypermeshes consume much less power than the 2D Torus and Mesh. This is primarily due to the fact that packets traverse fewer routers which significantly lowers the required power. Moreover under equal bisection bandwidth, the 2D Hypermesh NoC uses the lowest area and energy and outperforms the other topologies.

8. The energy-area product is proposed as a proxy for comparison between topologies, which reflects both the cost and performance metrics in the NoC domain. The NoCs are evaluated by their *energy-area products* and the 2D Hypermesh NoC has the lowest *energy-area products*.

9. Our analytic models and extensive experimental results show that the current FPGAs allow the development of reasonably large *Single Instruction stream Multiple Data stream* parallel processing architectures and that the Hypermesh NoC topology offers considerable benefits over conventional graph-based NoCs.

7.2 Future work

As the early design exploration and evaluation of NoCs are very important, evaluation through the analytic models will continue to present opportunities and challenges for the researches. In this thesis, we have explored different designs for the routers and topologies in the NoC domain. Our research will be extended in 2 directions:

1. *3D ASICs*: The 3 dimensional (3D) Integrated Circuits (ICs) have better performance and package density as compared to traditional 2D ICs. Moreover, combining the advantages of 3D ICs and NoCs provides significant gains in power, area, and performance of 3D NoCs. The analytic models proposed in this thesis are very accurate and have theoretical basis. These analytic models will be extended to support the 3D deep-submicron ASIC technology.

2. *3D FPGAs*: The next generation of FPGAs will use 3D technologies and tri-gate sub-micron technologies. Recently, Altera announced a new family of FPGAs based on Intel 14 nm Tri-Gate process technology, providing designers with upto a 10X increase in programmable gates [25]. Xilinx also announced a new 3D FPGA (Virtex-7 HT) with 6.8 billion transistors, providing designers with access to 2 million logic cells [28]. These evolutions allow the designers to accommodate more complex SoCs and NoCs to be realized in FPGAs. To

explore the NoC designs in the new generation of FPGAs, analytic models for power, area, and delay are required. Therefore, the analytic models in this thesis will be extended to support the 3D FPGAs.

Appendix A

Pseudo Random Walk Simulator

The following shows how the pseudo random walk simulator works. As stated in chapter 3, the pseudo random walk simulator guarantees that every link is 100% loaded (maximum load for NoCs). Therefore, packets should traverse all the links. In this simulator, the following constraints are made:

1. Flit injection rate is 1 flit per node per clock cycle.
2. Each packet has 32 flits.
3. Crossbar Switches works with 100% throughput. In other words, in a NxN crossbar switch, there are always N outputs.
4. There are *no* packet blocking at the intermediate routers due to the link contention. Packets don't use the same links.
5. The arbitration signals are chosen from predefined states and these states are identical for all the switches. The selected state applies to all the switches.
6. All the Muxes and Demuxes in a switch use the same arbitration signals (states).

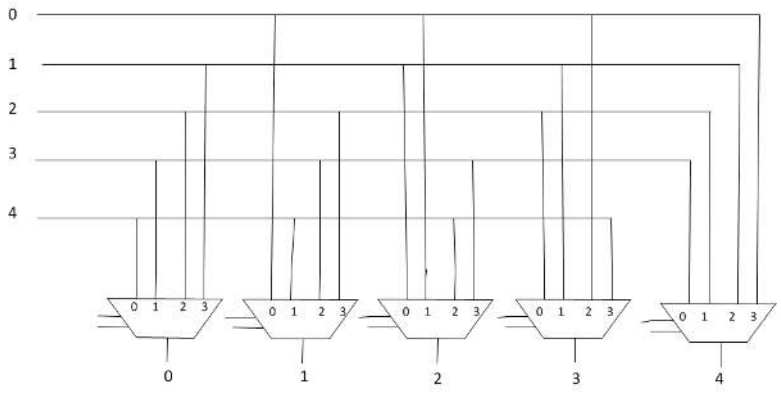


Figure A.1: Interconnections of a 5x5 B&S Crossbar Switch

7. The arbitration signals (states) change every 32 clock cycles.

The 2D Torus, GHC, and Mesh are selected to illustrate how this simulator works. For ease of understanding, the crossbar switch are Broadcast-and-Select (B&S).

A.0.1 2D Torus

Figure A.1 shows the interconnections in a 5x5 B&S crossbar switch in a 2D Torus. Usually in a 5x5 B&S crossbar switch, 5-to-1 Muxes are required. However a packet coming from a port will not exit from the same port. Therefore, instead of using 5-to-1 Muxes, 4-to-1 Muxes are used. The wiring pattern in figure A.1 is the same for every router. To force the crossbar switch works with 100% load all the selectors of Muxes are derived by identical signals.

Figure A.2 shows a 2D Torus with 16 nodes. In this figure every link is assigned a number. Let assume that a link connected to a router denotes 2 unidirectional links in opposite directions (one for the input port and one for the output port). The number shown on the link indicates the port number in the associated router. For example, if the number shown on a link is 1, it means that the link is connected to

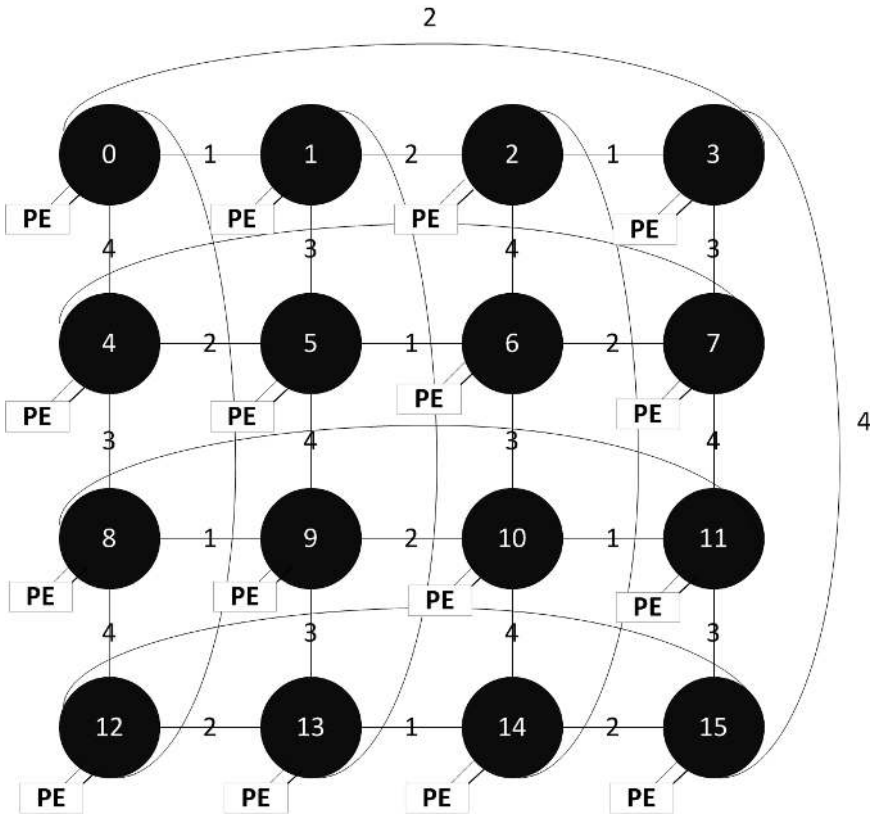


Figure A.2: Numbering of links in 2D Torus with 16 nodes

the port number 1 in the routers. In all the routers in the network, the injector and extractor channel is port 0.

As seen in figureA.2, a link connecting 2 routers has only one number. This pattern of link numbering ensures that for every packet, output (input) port of current router and input (output) port of the next router have the same number. In other words, a packet leaving output port A of the current router enters into input port A of the next router. In figure A.1 assume that the output port of a switch is the input port of the next switch. Now it is seen that the interconnections in the switch is in a circular sequence. Starting with input port '0', the following states take place in the routers:

1. State 1: Selectors of Muxes=‘00’: 0→1→2→3→4→0
2. State 2: Selectors of Muxes=‘01’: 0→2→4→1→3→0
3. State 3: Selectors of Muxes=‘10’: 0→3→1→4→2→0
4. State 4: Selectors of Muxes=‘11’: 0→4→3→2→1→0

As a result, using any arbitration value (state) for the crossbar switches, a generated packet from any node traverses 4 hops. (Note the arbitration value (state) for all routers is identical.)

In a 2D Torus with N nodes, there are 4N links. Since in the pseudo random walk simulator for the 2D Torus each packet traverses 4 different links, in a 2D Torus with N nodes, all the 4N links are traversed by the packets. Figures A.3 and A.4 show an example for a 2D Torus with 16 nodes which all the switches are in the state 1 (the selectors of Muxes are ‘00’). Note that in these figures each edge denotes 2 unidirectional links in opposite directions. As seen in these figures, each packet traverses 4 different links, and consequently all the links are 100% loaded.

A.0.2 2D GHC

Let’s examine another example, a 2D GHC with 16 nodes as shown in figure A.5, and note the pattern of the link numbering. In a link connecting 2 routers, the output (input) port of a router and input (output) port of the second router have the same number. For example in node ‘0’, link 3 connects the input (output) port 3 of router ‘0’ to output (input) port 3 of router ‘2’. In a 2D GHC with 16 nodes each router has 7 ports. The interconnections in a 7x7 router is shown in figure A.6. These router are constructed by seven 6-to-1 Muxes. Let’s assume that the output port of switch

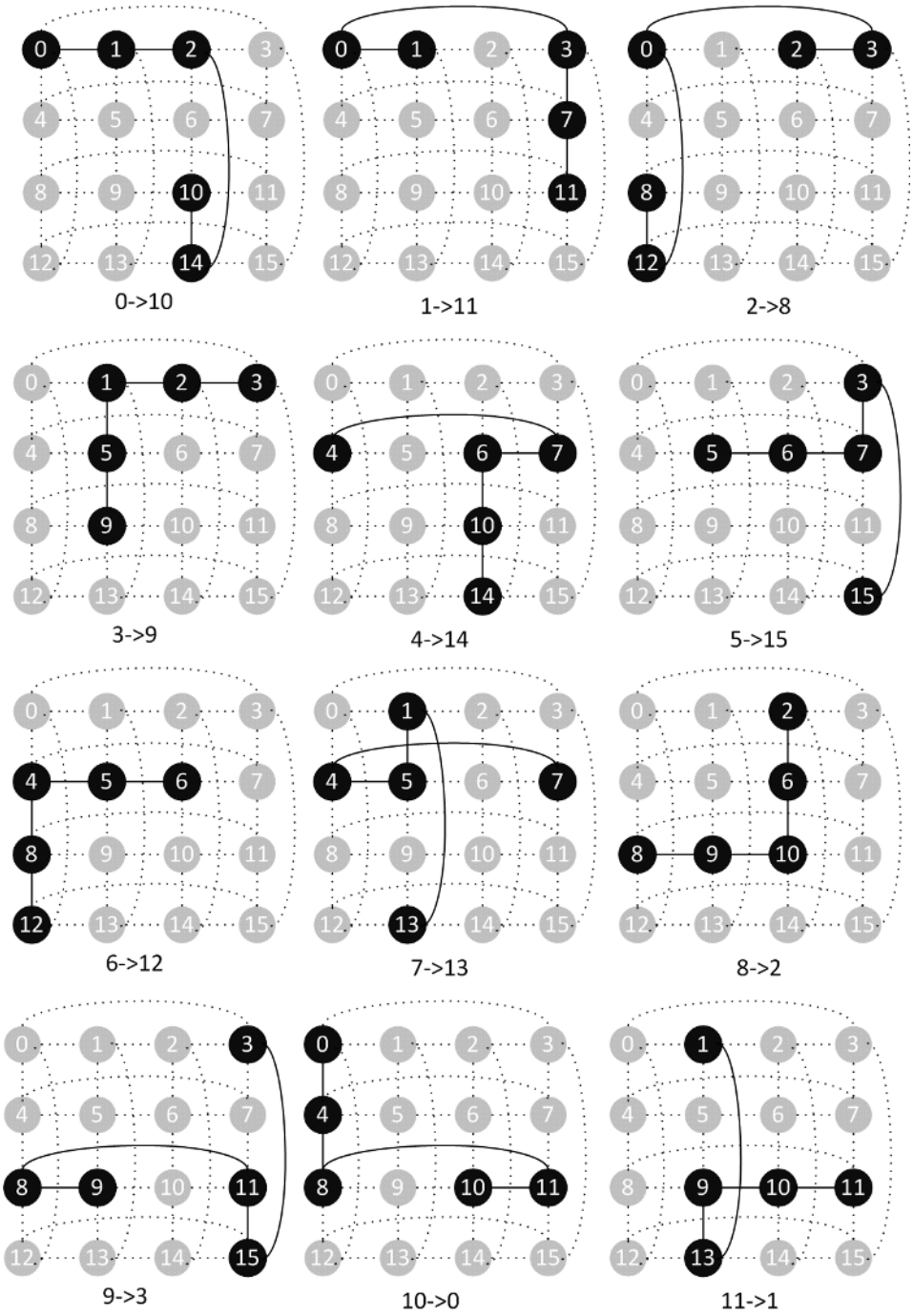


Figure A.3: A 2D Torus with 16 nodes in the state 1 (Packets 0 to 11) (Every packet traverses 4 edges and evey directed edge is used once in a every clock cycle)

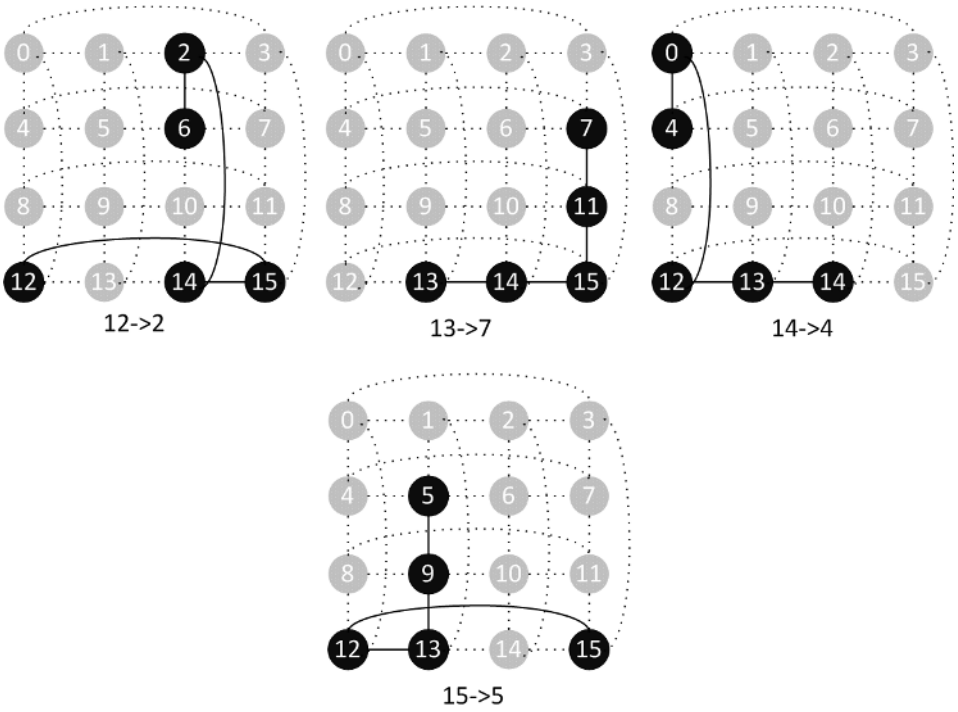


Figure A.4: A 2D Torus with 16 nodes in the state 1 (Packets 12 to 15) (Every packet traverses 4 edges and evey directed edge is used once in a every clock cycle)

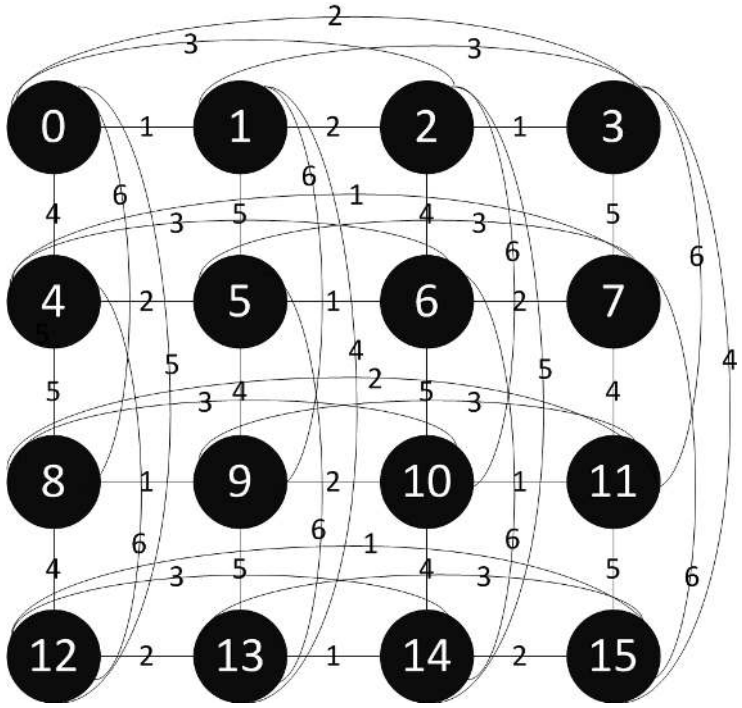


Figure A.5: Numbering of links in 2D GHC with 16 nodes

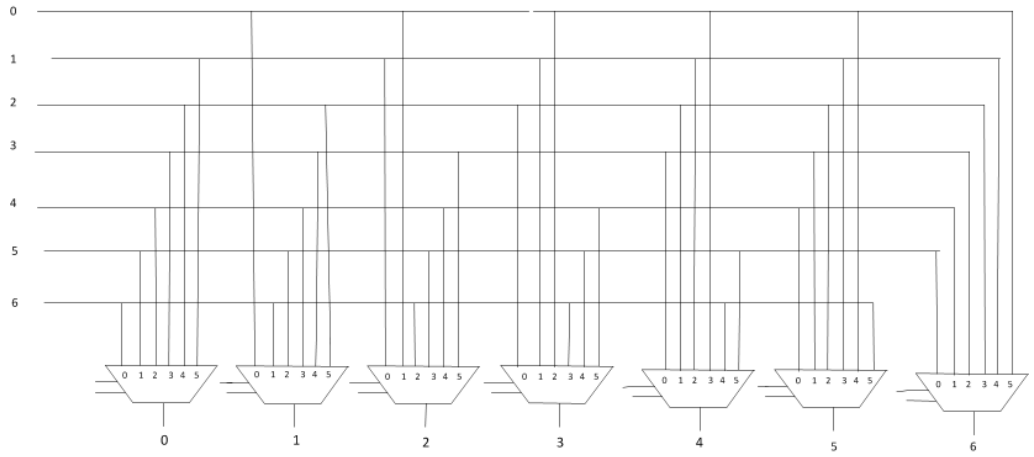


Figure A.6: Interconnections of a 7x7 B&S Crossbar Switch

is the input port of next switch. Starting with input port '0', the following states occur in the routers:

1. State 1: Selectors of Muxes='000': 0→1→2→3→4→5→6→0
2. State 2: Selectors of Muxes='001': 0→2→4→6→1→3→5→0
3. State 3: Selectors of Muxes='010': 0→3→6→2→5→1→4→0
4. State 4: Selectors of Muxes='011': 0→4→1→5→2→6→3→0
5. State 5: Selectors of Muxes='100': 0→5→3→1→6→4→2→0
6. State 6: Selectors of Muxes='101': 0→6→5→4→3→2→1→0

It is seen that the interconnections in the figure A.6 is in a circular sequence. In other words, a packet generated by an injection port 0 traverses all numbers (from 1 to 6) and then returns to port 0. Let's assume that in figure A.5 node 0 generates a packet. Depending on the arbitration signal, one of the 6 states happens and a packet in each state traverses 6 hops (links).

Figures A.7 and A.8 show an example for a 2D GHC with 16 nodes which all the switches are in the state 1 (the selectors of Muxes are '000'). Note that in these figures each edge denotes 2 unidirectional links in opposite directions. As seen in these figures, each packet traverses 6 different links, and consequently all the links are 100% loaded.

In total, when all 16 nodes generate packets, $6 \times 16 = 96$ different links are traversed.

The number of links in a 2D GHC with N nodes is $(2(\sqrt{N}-1)) \times N$. The number of links in a 2D GHC with 16 nodes is $6 \times 16 = 96$. Since the packets traverse 96

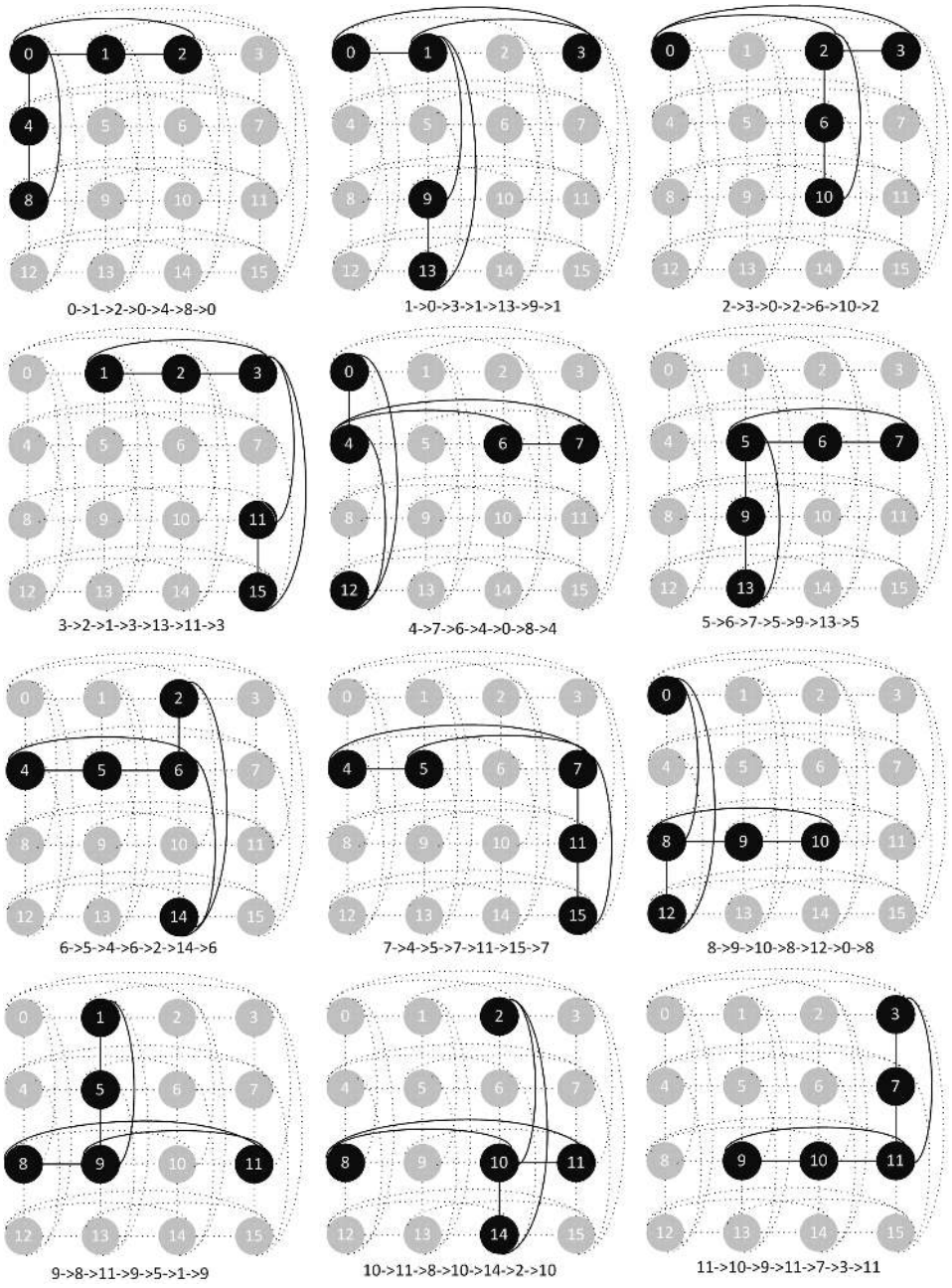


Figure A.7: A 2D GHC with 16 nodes in the state 1 (Packets 0 to 11) (Every packet traverses 6 edges and evey directed edge is used once in a every clock cycle)

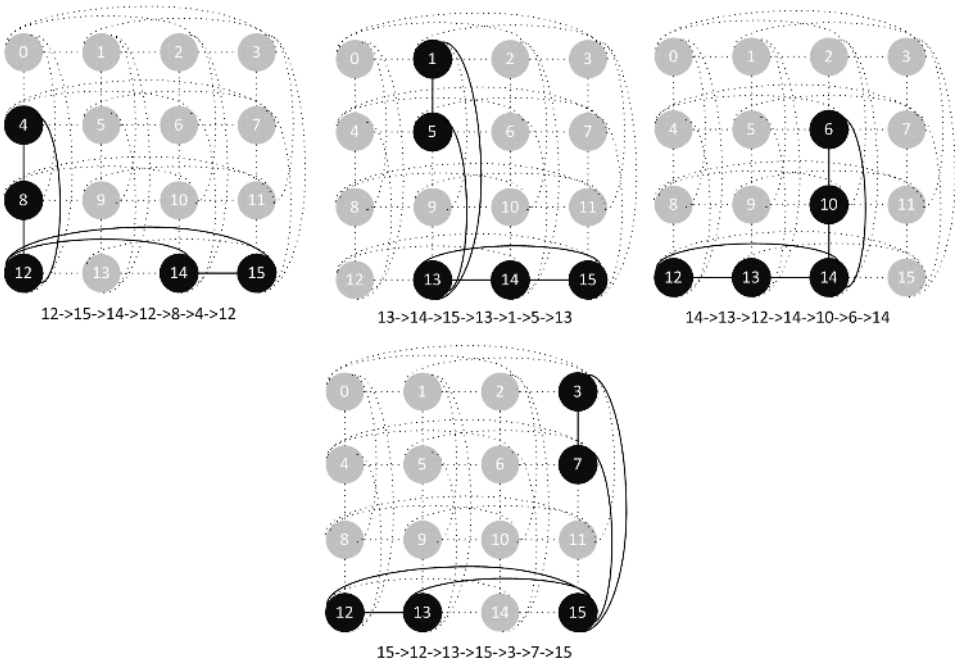


Figure A.8: A 2D GHC with 16 nodes in the state 1 (Packets 12 to 15) (Every packet traverses 6 edges and evey directed edge is used once in a every clock cycle)

different links, all the links are 100% loaded.

A.0.3 2D Mesh

Figure A.9 shows a 2D Mesh with 16 nodes. As seen in this figure, every link is assigned a number. In a 2D Mesh, crossbar switches have different sizes. The crossbar switches in corners are 3x3, the switches in edges are 4x4, and the other switches are 5x5. Figures A.10a and A.10b show the interconnection in 4x4 and 3x3 B&S crossbar switches. In the 2D Mesh, all the crossbar switches are derived by an identical arbitration signal. Note that in the 4x4 crossbar switches where the Muxes are 3-to-1, the arbitration signals (states) can be '00', '01', and '10'. When the arbitration signal is '11' the state of the 3-to-1 Muxes is the same as '10'. In the 3x3 crossbar

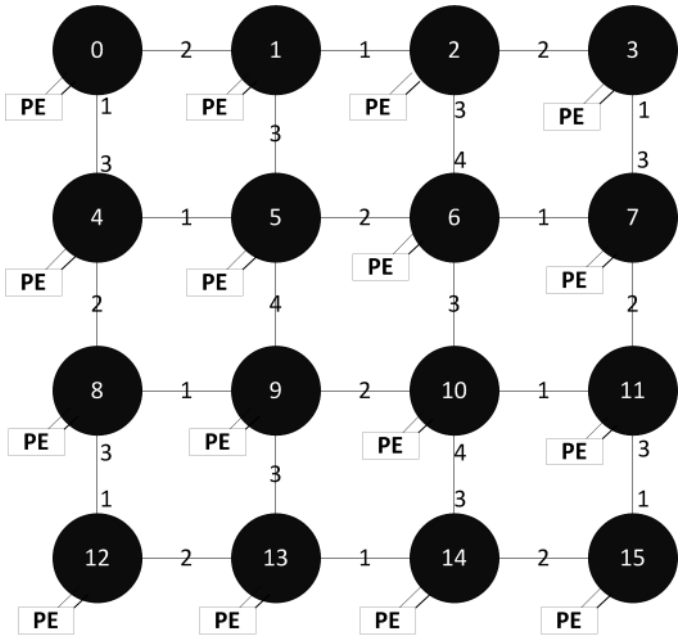


Figure A.9: Numbering of links in a 2D Mesh with 16 nodes

switches where the Muxes are 2-to-1, the Muxes are derived by the least significant bit (LSB) of the arbitration signals.

The following shows the number of different links traversed by packets, given different arbitration signals (states):

1. State 1: Selectors of Muxes='00': The total number of links traversed by 16 packets is 48.

2. State 2: Selectors of Muxes='01': The total number of links traversed by 16 packets is 48.

3. State 3: Selectors of Muxes='10': The total number of links traversed by 16 packets is 48.

4. State 4: Selectors of Muxes='11': The total number of links traversed by 16

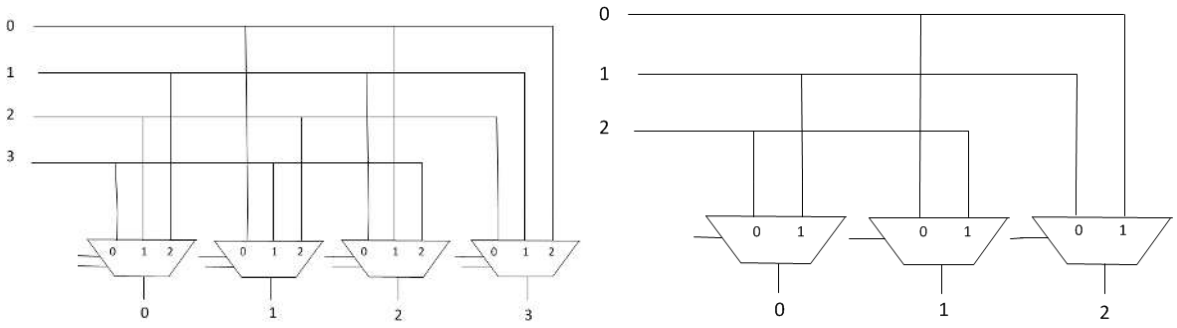


Figure A.10: Interconnections in (a) a 4x4 B&S Crossbar Switch, (b) a 3x3 B&S Crossbar Switch.

packets is 48.

Using any arbitration signal (selectors of Muxes), 48 different links are traversed by the packets. In a 2D Mesh with 16 nodes, there are 48 links. Therefore, all the links are 100% loaded.

Appendix B

P_B Definition

In this section the blocking probability (P_B) is defined. Let $P_B(i)$ be the blocking probability at clock cycle i. The general definition of $P_B(i)$ at clock cycle i is given by:

$$P_B(i) = \frac{N_{Blocked}(i)}{N_{UnBlocked}(i) + N_{Blocked}(i)} \tag{B.1}$$

where $N_{Blocked}(i)$ is the number of blocked flits at clock cycle i, and where $N_{UnBlocked}(i)$ is the number of header flits that they are not blocked at clock cycle i. Referring to Eq. B.1, the dominator is the total number of flits generated by source nodes at time i.

In the wormhole switching, the number of blocked flits at time i ($N_{Blocked}(i)$) is determined by the number of *header flits* that are blocked because their requested output ports are busy. In the wormhole switching, the number of unblocked flits at time i ($N_{UnBlocked}(i)$) is determined by the number of flits injected at clock cycle i which their *header flits* are not blocked at clock cycle i.

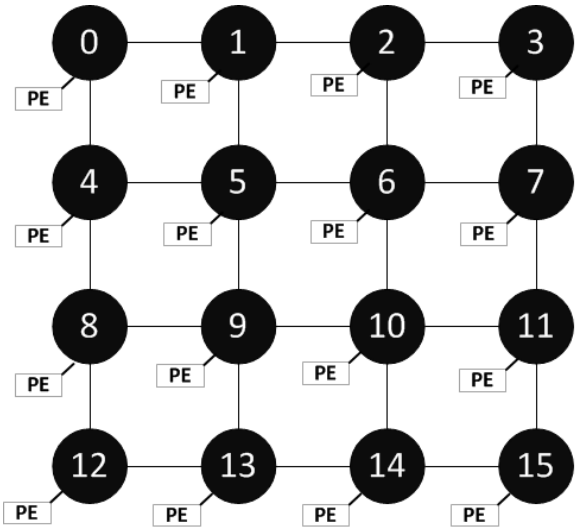


Figure B.1: 2D Mesh with 16 nodes

When a network operates in steady-state, the blocking probability P_B is defined as the average of all $P_B(i)$ over a long period of time, and is given by:

$$P_B = \frac{N_{Blocked}}{N_{UnBlocked} + N_{Blocked}} \tag{B.2}$$

where the $N_{Blocked}$ is the average of all $N_{Blocked}(i)$ over a long period of time, and where the $N_{UnBlocked}$ is the average of all $N_{UnBlocked}(i)$ over a long period of time.

For better understanding, we will see the following example. Figure B.1 shows a 2D Mesh with 16 nodes.

Assume the following nodes generate 4 packets as shown in figure B.2:

1. Packet 0: Node 0 to node 3 (λ_F =1 flit per cc)

2. Packet 1: Node 1 to node 7 (λ_F =1 flit per cc)

3. Packet 2: Node 2 to node 11 (λ_F =1 flit per cc)

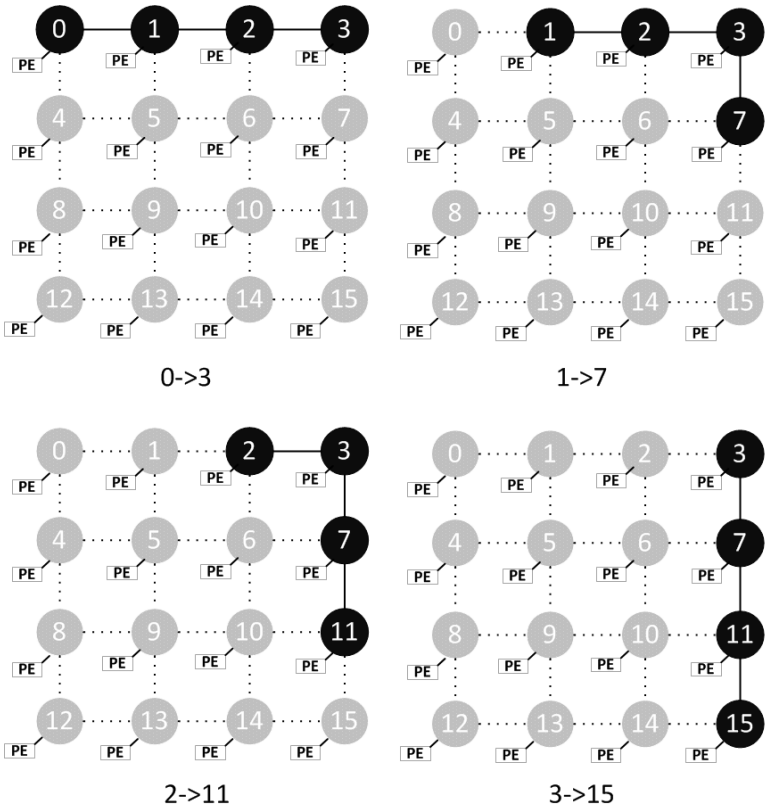


Figure B.2: The 4 packets in the example

4. Packet 3: Node 3 to node 15 (λ_F =1 flit per cc)

Let assume that each packet has 32 flits and packets traverse the networks using the wormhole switching. We assume that routing delay is 1 clock cycle. Once the routing is completed, the delay in forwarding a flit is 1 clock cycle per flit. Therefore, a header flit moves forward in 2 clock cycles and other flits move forward in 1 clock cycle. In this example we ignore the delay of input buffers.

Figure B.3 shows the complete timing for the generated packets. The notation $A \rightarrow B$ in figure B.3 denotes the input buffer of node B from the node A side. For example $0 \rightarrow 1$ means the input buffer of router 1 which is connected to router 0.

Figure B.3 also shows the number of blocked flits at clock cycle i ($N_{Blocked}(i)$). As

CC	ROUTERS						Blocked flits
	0->1	1->2	2->3	3->7	7->11	11->15	
2	HF	HF	HF	HF			
4	F2F1 HF	F2F1 HF	F2F1HF	F2F1	HF		3
6	F3F2F1 HF	F3F2F1 HF	F3F2F1HF	F4F3	F2F1	HF	3
8	F3F2F1 HF	F3F2F1 HF	F3F2F1HF	F6F5	F4F3	F2F1	3
10	F3F2F1 HF	F3F2F1 HF	F3F2F1HF	F8F7	F6F5	F4F3	3
33	F3F2F1 HF	F3F2F1 HF	F3F2F1 HF	TF F30	F29 F28	F27 F26	3
34	F3F2F1 HF	F3F2F1 HF	F4F3F2F1	HF TF	F30 F29	F28 F27	2
35	F3F2F1 HF	F3F2F1 HF	F5F4F3F2	F1HF	TF F30	F29 F28	2
37	F3F2F1 HF	F3F2F1 HF	F7F6F5F4	F3F2F1	HF	TF F30	2
39	F3F2F1 HF	F3F2F1 HF	F9F8F7F6	F5F4F3	F2F1	**Done**	2
61	F3F2F1 HF	F3F2F1 HF	TF F30F29F28	F27F26F25	F24F23		2
62	F3F2F1 HF	F4F3F2F1	HF TF F30F29	F28F27F26	F25F24		1
65	F3F2F1 HF	F7F6F5F4	F3F2F1 HF	TF F30F29	F28F27		1
67	F3F2F1 HF	F8F7F6F5	F4F3F2F1	HF TF	F30F29		1
68	F3F2F1 HF	F9F8F7F6	F5F4F3F2	F1 HF	TF F30		1
70	F3F2F1 HF	F11F10F9F8	F7F6F5F4	F3F2F1	**Done**		1
72	F3F2F1 HF	F13F12F11F10	F9F8F7F6	F5F4F3			1
90	F3F2F1 HF	TF F30F29F28	F27F26F25F24	F23F22F21			1
94	F7F6F5F4	F3F2F1HF	TF F30F29F28	F27F26F25			
96	F8F7F6F5	F4F3F2F1	HF TF F30	F29F28F27			
98	F10F9F8F7	F6F5F4F3	F2F1 HF	TF F30F29			
100	F12F10F9F8	F8F7F6F5	F4F3F2F1	TF			
101	F13F12F10F9	F9F8F7F6	F5F4F3F2	**Done**			
127			TF F30F29F28				
130			TF				
131			**Done**				

Figure B.3: Complete timing of wormhole-switched 2D Mesh with 16 nodes for the 4 generated packets

state earlier, a packet includes 32 flits. We use the packet format shown in figure 3.3 in Chapter 3, and explain it as follows:

1. *HF* denotes the header flit.
2. *TF* denotes the tail flit.
3. *Fi*, denotes the body flits, where i=1,...,30 is the assigned number of each body flit in a packet.

At *cc=0* four header flits are injected to their local routers. It takes 2 clock cycles that these 4 header flits move forward to get to their next router as their requested output port is *not* busy.

At *cc=2* the 4 header flits are buffered in the next routers toward their destinations (shown in figure B.4a). For example the header flit (*HF*) of the packet 0 is buffered at the router 1. The routing delay is one clock cycle to decode the requested output port of the header flits. At *cc=4* only the header flit of the packet 3 moves forward to get to the next intermediate router and therefore the header flits of packet 0, 1, and 2 are blocked (shown in figure B.4b) as the requested output ports are reserved by other packets. For example the header flit of packet 0 is blocked by the header flit of packet 1. Therefore, 3 out of 4 packets are blocked at cc=4 ($P_B(4) = 3/4$).

Figure B.5a shows the state of the 2D Mesh at clock cycle 6. The PEs of the blocked packets generate flits until all the buffers in their path are filled. As seen in this figure, the header flit (*HF*) of the packet 3 moves forward to its destination (router 15). At cc=6, $P_B(6) = 3/4$. From clock cycle 4 to 33 only the flits of packet 3 move forward and the flits of other 3 packets are blocked. Therefore, 3 out of 4 flits are blocked. Figure B.5b shows the state of the network at clock cycle 33. At

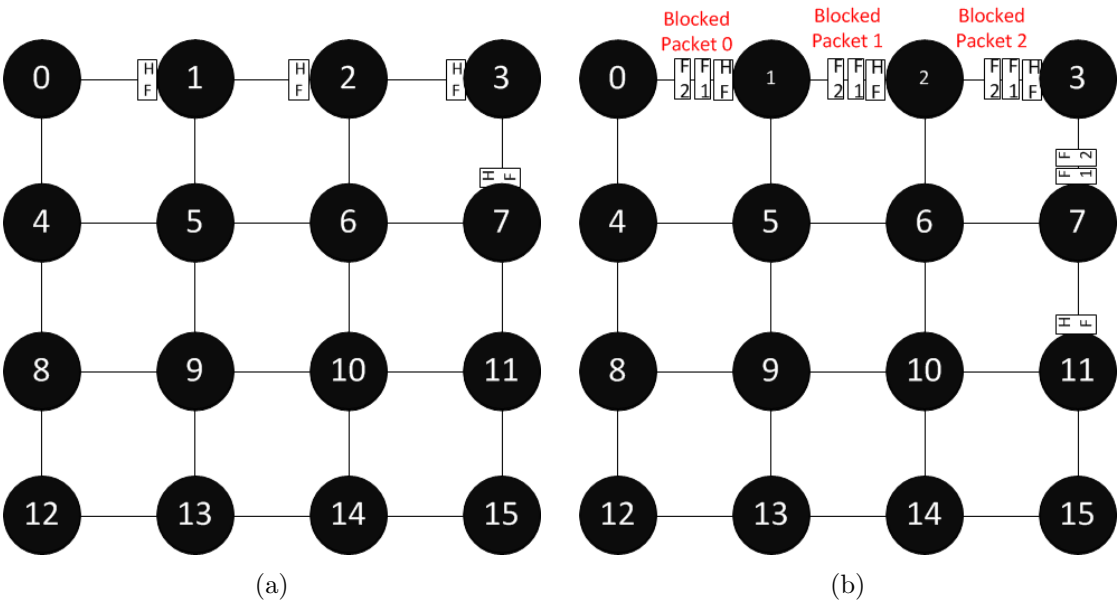


Figure B.4: State of the 2D Mesh (a) at CC 2, (b) at CC 4.

cc=33, the tail flit (TF) of the packet 3 leaves the router 3 and the output port of the router 3 is released. At the next clock cycle (cc=34) the header flit of the packet 2 is unblocked and can move forward (as shown in figure B.6b). Therefore, 2 out of 3 packets are blocked at cc=34 ($P_B(34) = 2/3$).

Figures B.6b, B.7a, and B.7b show the states of the 2D Mesh at clock cycle 35, 37, and 39. At *cc=39* the packet 3 is completely delivered. From clock cycle 34 to 61 only the flits of the packet 2 move forward and the flits of other 2 packets are blocked. Therefore, at clock cycle 39 two out of three flits are blocked ($P_B(39) = 2/3$). At *cc=61* the tail flit (TF) of the packet 2 leaves the router 2 and releases the output port of the router 2. Now the requested output port of the packet 1 is available. At *cc=62* the header flit (HF) of the packet 1 moves forward to the next router (as shown in figure B.8a). From clock cycle 62 to 90 only flits of packet 0 are blocked and flits of packet 1 move toward to their destination. At *cc=90* the tail flit (TF) of

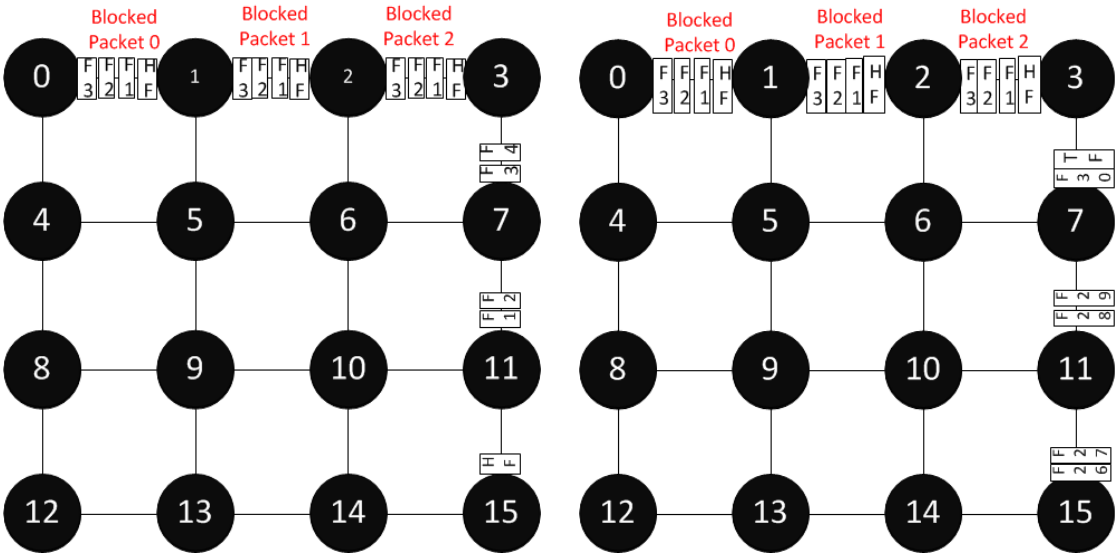


Figure B.5: State of the 2D Mesh (a) at CC 6, (b) at CC 33.

packet 1 departs the router 1 and releases the output port of the router 1. At cc=91 the header flit (HF) of packet 0 leaves the router 1 (as shown in figure B.8b) because its requested output port is available.

From clock cycle 91 until end of delivery of the packet 0, no packet blocking occurs.

To conclude the example, we note that 4 packets are successfully delivered to their destinations and each packet has 32 flits. Therefore the total number of delivered flits ($N_{UnBlocked}$) is 128.

There are some clock cycles during which the header flits are blocked and they cannot move forward. According to figure B.3, the total time it takes to deliver all the packets is 131 clock cycles. The number of the blocked flits at clock cycle i is given by:

1. $N_{Blocked}(i) = 3$, $P_B(i) = 3/4$, $3 < i < 34$

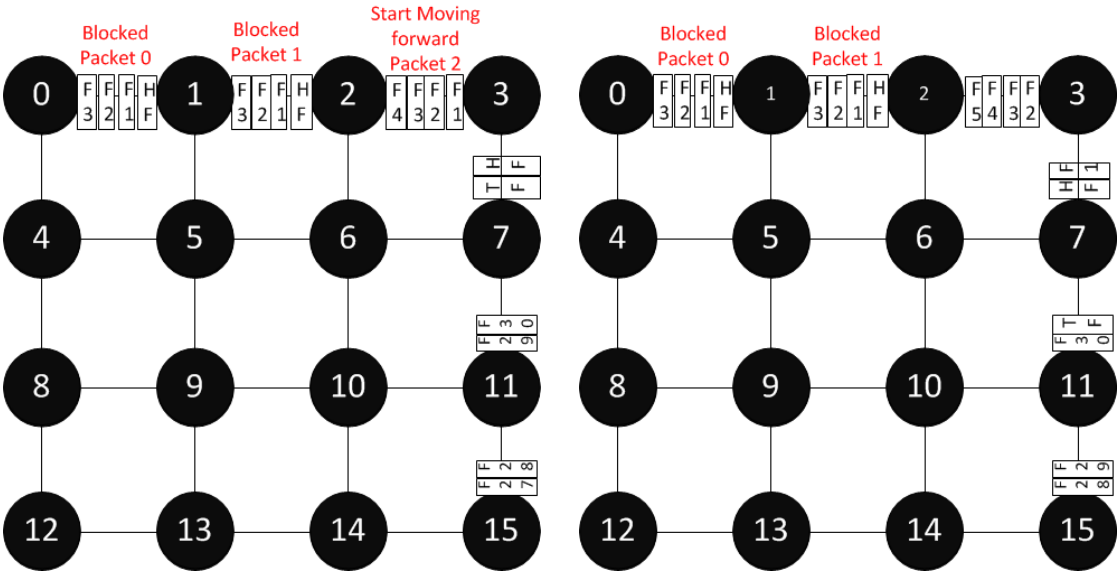


Figure B.6: State of the 2D Mesh (a) at CC 34, (b) at CC 35.

2. $N_{Blocked}(i) = 2$, $P_B(i) = 2/3$, $34 \leq i < 62$

3. $N_{Blocked}(i) = 1$, $P_B(i) = 1/2$, $62 \leq i < 91$

4. $N_{Blocked}(i) = 0$, $P_B(i) = 0$, $91 \leq i \leq 131$

The total number of the blocked flits ($N_{Blocked}$) in interval 0 to 131 clock cycle is given by:

$$N_{Blocked} = 30 \times 3 + 28 \times 2 + 29 \times 1 = 175 \tag{B.3}$$

Let's assume that the total number of generated flits is the sum of the successfully delivered flits and the blocked flits. Referring to Eq. B.2, the P_B in the 131 clock

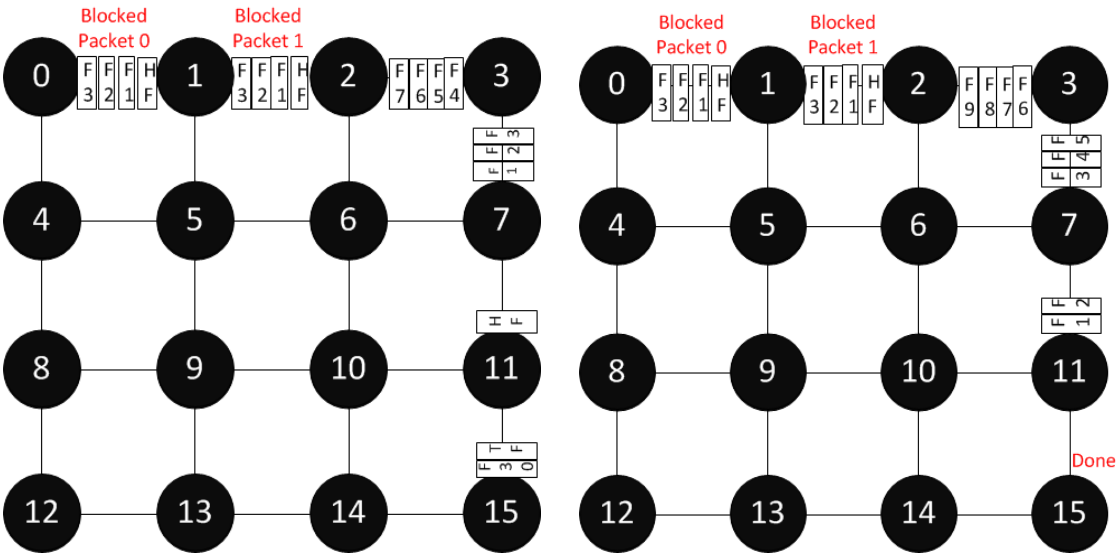


Figure B.7: State of the 2D Mesh (a) at CC 37, (b) at CC 39.

cycles is given by:

$$P_B = \frac{N_{Blocked}}{N_{UnBlocked} + N_{Blocked}} = \frac{175}{175 + 128} = \frac{175}{303} = 0.577 \tag{B.4}$$

As previously stated in chapter 5, the blocking probability P_B can be defined as the expected waiting time that a header flit waits in the blocking queues over the packet latency, when its requested output links are busy. The network latency of a packet in a wormhole-switched NoC with H hops is given by [54]:

$$T_P = H \times (T_{Routing} + T_{Xbar} + T_{Link}) + (m - 1) \times (T_{Xbar} + T_{Link}) + Bl \tag{B.5}$$

where m is the average number of flits in a packet, and where $T_{Routing}$, T_{Xbar}, and T_{Link} are the delay for the routing, crossbar switch, and link traversal respectively.

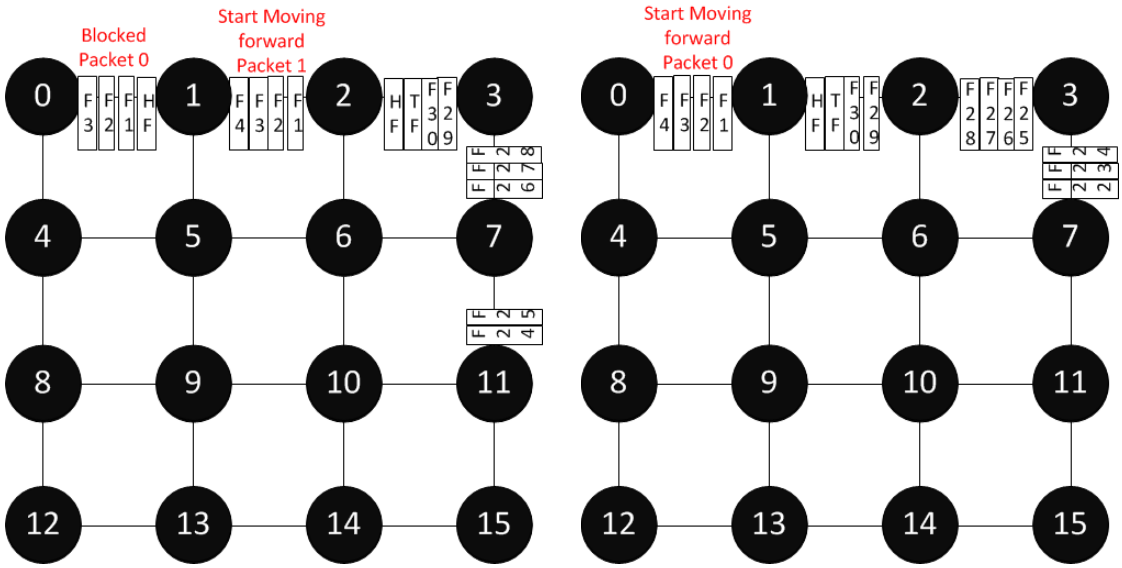


Figure B.8: State of the 2D Mesh (a) at CC 62, (b) at CC 91.

Referring to Eq. B.5, the term Bl is the average blocking time seen by a header flit [54]. Without any blocking, each packet should have reached its destination in 37 clock cycles.

In our example, the T_P of 4 packets is determined as the average network latency of the 4 packets. According to figure B.3, T_P of the 4 packets is given by:

$$T_P = \frac{39 + 70 + 101 + 131}{4} = 85.25 \tag{B.6}$$

According to the definition of Bl in Eq. B.5, the P_B can be defined as the expected waiting time that a header flit waits in the blocking queue over the packet latency, when their requested output link is busy. Therefore, the P_B is given by:

$$P_B = \frac{Bl}{TP} = \frac{85.25 - 37}{85.25} = 0.566 \tag{B.7}$$

where the result is approximately equal to the result in Eq. B.4.

In the 2D Mesh shown in figure B.1 there are 16 nodes. According to figure B.3, the time it takes to deliver all the flits is 131 clock cycles. The average flit injection rate per node per clock cycle to a network with N nodes (before blocking is considered) is given by:

$$\lambda_F = \frac{N_{UnBlocked} + N_{Blocked}}{T} * \frac{1}{N} \tag{B.8}$$

where T is the total time takes to deliver all the flits, where the numerator is the total injection flit in the time interval T clock cycles, and where $\frac{1}{N}$ is to take the average injection rate of N nodes. The average flit injection rate per node per clock cycle for the 2D Mesh with 16 nodes (in the example) is given by:

$$\lambda_F = \frac{N_{UnBlocked} + N_{Blocked}}{T} * \frac{1}{N} = \frac{303}{131} * \frac{1}{16} = 0.144 \tag{B.9}$$

Referring to Eq. 5.8, the average effective rate of flits arrivals to the network per node per clock cycle (λ'_F) is given by:

$$\lambda'_F = \lambda_F * (1 - P_B) = 0.144 \times (1 - 0.577) = 0.061 \tag{B.10}$$

which reflects the average effective rate of λ_F to the network.

Appendix C

Evaluation of different hyperedge designs in 2D Hypermesh NoCs

In chapter 6, the hyperedges are constructed by the B&S1 crossbar switches. Using the B&S1 crossbar switches in the hyperedges has some advantages such as area-efficiency. For small size Hypermesh NoCs using the B&S1 as the hyperedges is an efficient design. However, as the size of the network increases the energy efficiency and performance of the Hypermesh using the B&S1 hyperedges reduce. The performance degradation is due to the long buses in the hyperedges.

C.1 Pipelined Hyperedges

To improve the performance of Hypermeshes, many hyperedge designs have been explored. Two new hyperedge designs which are suitable the most are selected, and are explained in the following sub-sections.

C.1.1 HE2: The Special Pipelined Demux-Mux Switch (S-PDM)

Figure C.1a illustrates a special pipelined Demux-Mux (S-PDM) switch design. The S-PDM design consists of Demuxes that connect to the input ports, and Muxes that connect to the output ports. When we say 'Special', we mean that the switch is 'Externally Pipelined', i.e., N pipeline latches (with W DFFs each) are inserted at the input ports of Demuxes and N pipeline latches are at output ports of Muxes. These latches are shown by the black squares in figure C.1a. In our experiments, the most of DFFs in the existing LEs can be used, and only few new resources (LEs) are needed, which they can be ignored. The total resource usage of a S-PDM switch (W bits wide) uses the area models in Eq. 4.6 and Eq. 4.1, and is given by C.1.

$$A_{HE2}(N,W) = N.A_{Dmux}(1,N-1,W) + N.A_{Mux}(N-1,1,W) \tag{C.1}$$

The total power consumed by the S-PDM switch uses the power models in Eq. 4.5 and Eq.4.7 and is given by Eq. C.2:

$$\begin{aligned} P_{HE2}(N,W) &= N.P_{Dmux}(1,N-1,W) + N.P_{Mux}(N-1,1,W) \\ &+ W.2.N.P_{DFF} + (N-1).P_{bus}(\bar{L},W,Z) \end{aligned} \tag{C.2}$$

where $\bar{L}$ is the average wire length in a bus between a Demux and Mux tree $\approx \sqrt{A_{HE2}(N,W)}$ and $Z = 1$. The power of a DFF P_{DFF} is given by $P_{DFF} = K_{DFF}.f_{clk}.f_{duty}$, where K_{DFF} represents the capacitance switched by one DFF per transition.

The delay of a S-PDM switch of size NxN is equal to the delay of a large Demux

tree plus the delay of a large Mux tree and is given by Eq. C.3:

$$D_{HE2}(N,W) = D_{Mux}(N-1,1,W) + D_{Dmux}(1,N-1,W) \quad \text{(C.3)}$$

C.1.2 HE3: The Special Pipelined Broadcast-and-Select Switch (S-B&S)

Figure C.1b illustrates a S-B&S switch design. When we say 'Special', we mean that the switch is 'Externally Pipelined', i.e., in this switch, N pipeline DFFs are inserted at the input port of the switch and N pipeline DFFs are inserted at the output ports of the switch. These latches are shown by black squares and use the DFFs already available in the LEs, and do not incur an additional cost. The total area of NxN S-B&S switch is equal to the area of N Muxes, and is given by Eq. C.4.

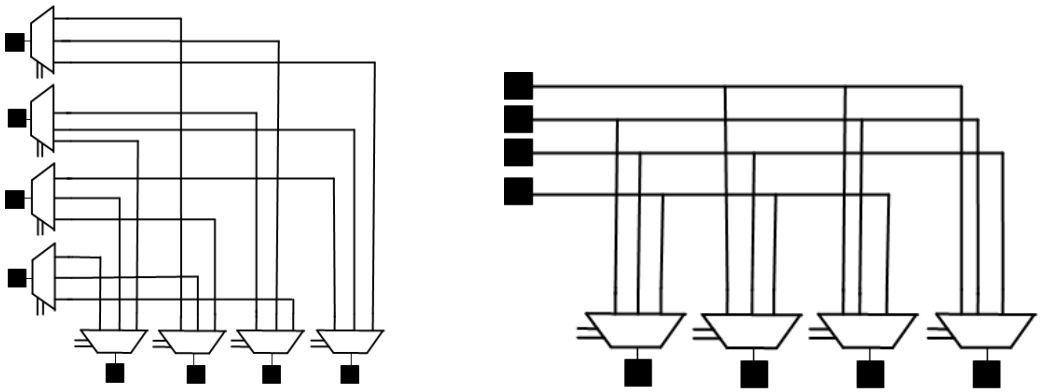
$$A_{HE3}(N,W) = N.A_{Mux}(N-1,1,W) \quad \text{(C.4)}$$

The total power consumed uses the models Eq. 4.4 and Eq. 4.10 and is given by Eq. C.5:

$$P_{HE3}(N,W) = N.P_{Mux}(N-1,1,W) + 2.N.W.P_{DFF} + N.P_{Bus}(L,W,N-1) \quad \text{(C.5)}$$

where the first term is the power consumed in N Muxes, where the second term is the power consumed in the $2.N$ pipeline DFFs, and where the last term is the power dissipated in the N broadcast buses. The length L is equal to $N * \sqrt{X}$, where X is the average number of LEs in the Mux N-to-1.

The delay of $S - B\&S$ switch NxN is equal o the delay of a large Mux tree



(a) (b)

Figure C.1: 2 pipelined hyperedges (a) Special B&S, (b) Special PDM (The special switches are externally pipelined.)

($D_{Mux}(N, 1, W)$) plus the delay of the broadcast bus of length L, and is given by Eq. C.6:

$$D_{HE3}(N, W) = D_{Mux}(N, 1, W) + D_{bus}(\sqrt{A_{HE3}(N, W)}) \tag{C.6}$$

C.2 Area and Delay Analysis of the Pipelined Hypermesh NoCs

C.2.1 Area

The total area of 2D Hypermesh with N nodes (with router degree = 3, datapath width = W, with K VCs per edge, with area of the processor core = C LEs, with concentration = M, and with the number of injector and extractor channels = Y) is expressed by Eq. 6.1.

C.2.2 Critical Path Delay

In the pipelined designs (S-PDM and S-B&S), the pipelined latches break the critical path. The path with higher delay yields the critical path. When EM blocks are used as input buffers, the delay of an Hypermesh using the *pipelined* hyperedge designs is given by:

$$D_{HM-EM}(N,W) = D_{S2}(3,W) + D_{EM} + D_{bus}((\sqrt{A_{HM}} - \sqrt{A_{HE}})/2) \qquad \text{(C.7)}$$

where $D_{S2}(3,W)$ is the delay of an 3x3 crossbar switch B&S1, and $D_{bus}((\sqrt{A_{HM}} - \sqrt{A_{HE}})/2)$ is the delay of longest wire connecting PEs to the hyperedge.

When DFF registers are used as input buffers, the delay of an Hypermesh using the *pipelined* hyperedge designs is given by:

$$D_{HM-DFF}(N,W) = D_{Si}(3,W) + D_{bus}((\sqrt{A_{HM}} - \sqrt{A_{HE}})/2) \qquad \text{(C.8)}$$

where $D_{S2}(3,W)$ is the delay of an 3x3 crossbar switch B&S1, and $D_{bus}((\sqrt{A_{HM}} - \sqrt{A_{HE}})/2)$ is the delay of longest wire connecting PEs to the hyperedge.

C.3 Evaluation Methodology of the NoCs

In this sub-section (1) the NoCs are compared based on their power, area, and energy consumption, and then, (2) the NoCs are evaluated by their *energy-area product*. We assume that the crossbar switches are the B&S1 switch design. Three different hyperedge designs are built by the switch designs in section C.1. The Hypermeshes using the 3 hyperedge designs are compared to the GHC and BHC. In evaluation of

NoCs, we assume that each PE uses 4K LEs.

Area

Area is one of the most important metrics in both NoCs and FPGAs domains. One of the most important advantages of the Hypermesh topology is the area-efficiency while offering high performance. Figure C.2a, C.2b, C.2c illustrate the area comparison of the NoCs with 256 nodes under equal bisection bandwidth. The resource usages are in terms of Logic Elements (LEs) and EM blocks. In figure C.2a and C.2b we assume that the input buffers are EM blocks, while in figure C.2c the input buffers are DFF registers (which are expressed in terms of LEs).

Most of studies in NoC domain concentrate to the buffers since the buffers are one of the most power and area hungry component in the design. The Hypermesh NoCs use fewer amount of buffers since the routers' radix is constant and equal to 3. For example for N=256 nodes and with equal bisection bandwidth, the Hypermesh uses 33% of the EM blocks that the 2D BHC uses and 38% of the EM blocks that the GHC needs.

As seen in figure C.2, the 2D Hypermesh using B&S switch designs outperform the 2D BHC and GHC in terms of LEs and EM blocks with significant margin.

Table C.1 summarizes area comparison between different NoCs with 256 nodes with different type of buffers under equal bisection bandwidth. As seen in table C.1, the 2D BHC and GHC are compared to the 3 different Hypemeshes. The values reported in this table are the percentages of the area that a 2D Hypermesh uses while comparing to the 2D BHC and GHC. Under equal bisection bandwidth, the area usage (in terms of LEs) of the 2D Hypermesh with 256 nodes using B&S switch

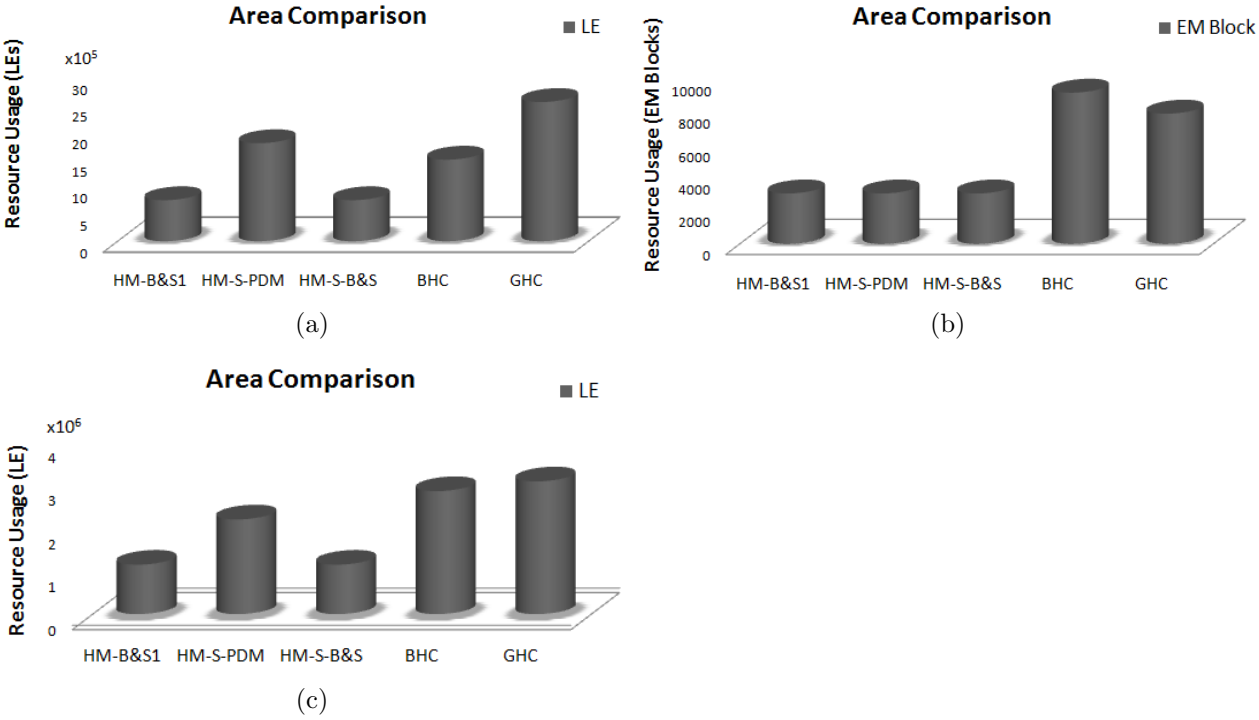


Figure C.2: Area analysis of the NoCs with 256 nodes a) in terms of LEs (using *EM block* as input buffers) b) in terms of EM blocks c) in terms of LEs (using *DFF Register* as input buffers) (With Equal Bisection Bandwidth, Switch=S1 (B&S1), PE=4K LEs)

.

designs is 30% to 39% of the area that the GHC uses and 42% to 50% of the area that the BHC uses.

Energy Per Algorithm

In this sub-section the NoCs are evaluated based on their energy consumption under the Bitonic sorting algorithm and the FFT algorithm.

The completion time of an NoC (expressed in terms of ns) given an algorithm is expressed by Eq. 6.25

Figure C.3 shows the completion time of the 2D Hypermeshes, BHC, and GHC

Table C.1: Area comparison between 2D Hypemesh NoCs and the 2D BHC and GHC with 256 nodes (under equal bisection bandwidth, Switch =B&S1, PE=4K LEs)

Topology		HM B&S1		HM S-B&S2		HM S-PDM	
	Buffer	LEs	EM	LEs	EM	LEs	EM
GHC	EM	30%	38%	30%	38%	70%	38%
BHC	EM	50%	33%	50%	33%	120%	33%
GHC	DFF	39%	-	39%	-	72%	-
BHC	DFF	42%	-	42%	-	77%	-

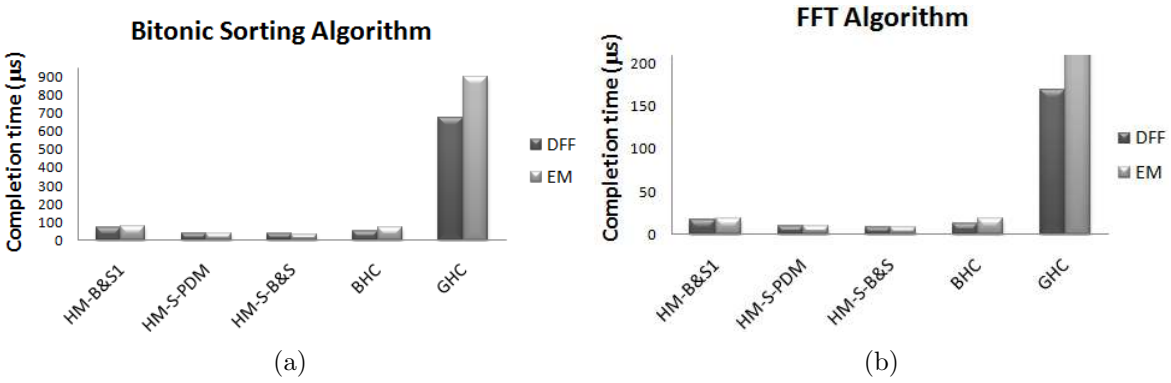
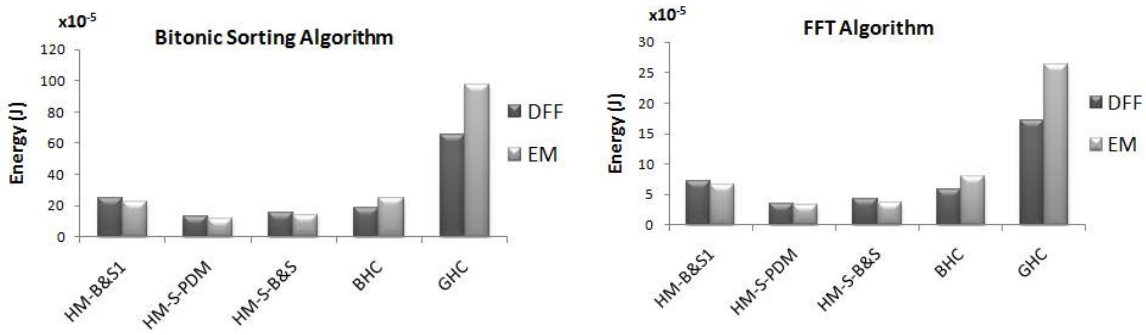


Figure C.3: Completion time of the topologies with 256 nodes under a) Bitonic sorting algorithm b) FFT algorithm (under equal bisection bandwidth, Switch=S1 (B&S1), PE=4K LEs)

.

with 256 nodes under the Bitonic sorting algorithm and FFT algorithm (with equal bisection bandwidth). As seen in figure C.3, the 2D Hypermeshes outperform the 2D BHC and GHC with significant margin. For example under FFT algorithm and using EM blocks as input buffers, the 2D Hypermesh with 256 nodes needs 42-98% of the time that the BHC needs to perform the FFT algorithm and 5-8% of the time that the GHC needs to perform the FFT algorithm.

Figure C.4a and C.4b show the energy consumption of the 2D Hypermeshes, BHC, and GHC with 256 nodes under the Bitonic sorting algorithm and FFT algorithm (with equal bisection bandwidth).



(a) (b)

Figure C.4: Energy analysis of the NoCs with 256 nodes under a) Bitonic sorting algorithm b) FFT algorithm (under equal bisection bandwidth, Switch=S1 (B&S1), PE=4K LEs)

.

As seen in figures C.4a and C.4b, the 2D Hypermesh using the S-PDM and S-B&S hyperedge designs considerably consumes lower energy than 2D BHC and GHC. For example under FFT algorithm, the 2D Hypermesh with 256 nodes using the S-PDM hyperedge design consumes 59% of the energy that the BHC uses and 20% of the energy that the GHC uses.

Energy-Area Product

Figures C.5a and C.5b show the *energy-area product* of the 2D Hypermeshes, BHC, and GHC with 256 nodes under the Bitonic sorting algorithm and FFT algorithm (with equal bisection bandwidth). As seen in figure C.5, the 2D Hypermeshes have significantly lower *energy-area product* compared to the BHC and GHC.

Table C.2 summarizes the energy and energy-area product (*E-A*) comparison between different NoCs with 256 nodes under FFT and Bitonic sorting algorithms. As seen in table C.2, the 2D BHC and GHC are compared to the 3 different Hypemeshes.

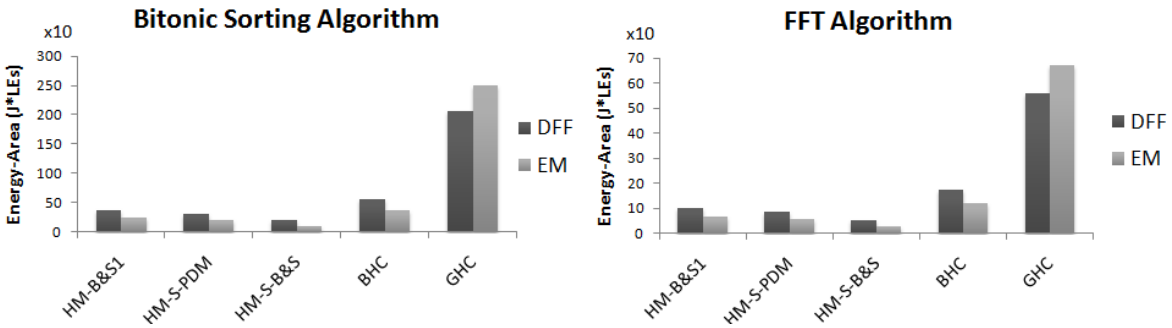


Figure C.5: Energy-Area product of the NoCs with 256 nodes under a) Bitonic sorting algorithm b) FFT algorithm (under equal bisection bandwidth, Switch=S1 (B&S1), PE=4K LEs)

.

The values reported in this table are the percentages of the energy and energy-area product (*E-A*) that a 2D Hypermesh uses while comparing to the 2D BHC and GHC. For example, a 2D Hypemesh using S-PDM hyperedge design consumes 18% of the energy that the 2D GHC uses.

Referring to table C.2, the energy consumption of the 2D Hypermesh depends on the hyperedge designs. In terms of energy, using S-PDM and S-B&S switches as the hyperedges are more suitable. As seen in table C.2, the 2D Hyperemsh outperforms the 2D GHC and BHC in terms of *energy-area product* with significant margin. The *energy-area product* of the 2D Hyperemsh with 256 nodes is 9-18% of the 2D GHC, and 29-67% of the 2D BHC.

Choosing the right switch design for the hyperedges depends on the design constraint. If the Hypermesh NoC is power-constrained, then using the S-PDM design as the hyperedges is optimal. If the Hypermesh NoC is the area-constrained, then using the B&S switch designs (HE1 and HE3) as the hyperedges are optimal, as they consistently have the lower area. In terms of energy per algorithm, using the S-PDM

Table C.2: Comparison of energy and energy-area product between 2D Hypemesh NoCs and the 2D BHC and GHC under 2 parallel algorithms with 256 nodes (under equal bisection bandwidth, PKT=10Kbits, Switch =B&S1, PE=4K LEs)

Topology		HM B&S1		HM S-B&S2		HM S-PDM	
	Alg.	Energy	E-A	Energy	E-A	Energy	E-A
GHC	FFT	41%	18%	24%	9%	20%	15%
BHC	FFT	120%	55%	71%	29%	59%	47%
GHC	Bitonic	38%	18%	23%	9%	20%	15%
BHC	Bitonic	133%	67%	83%	35%	70%	56%

and S-B&S switch designs (HE2 and HE3) as the hyperedges are optimal.

Bibliography

[1] William J. Dally, and Brian Towles, "Route packets, not wires: On-chip interconnection networks," in Proceedings of IEEE Design Automation Conference (DAC), pp. 684-689, 2001.

[2] Luca Benini, and Giovanni De Micheli, "Networks on chips: A new SoC paradigm", in Proceedings of IEEE Computer 35, no. 1, pp. 70-78, 2002.

[3] A. Hemani, A. Jantsch, S. Kumar, A. Postula, 1. Oberg, M. Millberg, and D. Lindqvist, "Network on a chip An architecture for billion transistor era", in Proceedings of IEEE NorChip Conference, pp. 166-173, Nov. 2000.

[4] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches", in Proceedings of ACM Transactions on Electron and System, vol. 12, no. 3, pp. 1-20, Aug. 2007.

[5] B. Li, L. S. Peh, and P. Patra, "Impact of process and temperature variations on network-on-chip design exploration," in Proceedings of IEEE Network on Chip (NoC), pp. 117-126, Apr. 2008.

[6] D. Rahmati, A. E. Kiasari, S. Hessabi, and H. Sarbazi-Azad, "A performance and power

analysis of WK-recursive and mesh networks for network-on-chips", in Proceedings of IEEE Design Automation and Test, pp. 142-147, Oct. 2006.

[7] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design tradeoffs for network-on-chip interconnect architectures", IEEE Transactions on Computer (TC), vol. 54, no. 8, pp. 1025-1040, Aug. 2005.

[8] H. S. Wang, L. S. Peh, and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks", in Proceedings of IEEE Design Automation and Test (DATE), pp. 1238-1243, Mar. 2005.

[9] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS", in Proceedings of the IEEE International Solid State Circuit Conference, 2007.

[10] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar operand networks: On-chip interconnect for ILP in partitioned architectures", in Proceedings of the International Symposium on High Performance Computer Architecture, Feb. 2003.

[11] Natalie Enright Jerger and Li-Shiuan Peh, "On-chip networks", Synthesis Lectures on Computer Architecture, no. 1, pp. 1-141, 2009.

[12] T.H. Szymanski, "Hypermeshes: Optical Interconnection Networks for Parallel Computing," Journal of Parallel Distributed Computing, 1995.

[13] W. Dally and B. Towles, "Principles and Practices of Interconnection Networks," San Francisco, CA: Morgan Kaufmann Publication Press, 2004.

[14] William J. Dally and Charles L. Seitz, "The torus routing chip", Journal of Distributed Computing, 1(3):187-196, 1986.

[15] Erricos John Kontoghiorghes, "Handbook of parallel computing and statistics", Vol. 184. CRC Press, 2006.

[16] L.M. Ni and P.K. McKinley, "A survey of wormhole routing techniques in direct networks," in Proceedings of IEEE Computer, pp. 62-76, 1993.

[17] Li-Shiuan Peh, and William J. Dally, "A delay model for router microarchitectures", in Proceedings of IEEE Micro, no. 1 pp. 26-34, 2001.

[18] R.R. Tamhankar, S. Murali, and G. De Micheli, "Performance driven reliable link design for networks on chips," in Proceedings of the Asia and South Pacific Design Automation Conference, pp. 749-754, 2005.

[19] William J. Dally, "Virtual-channel flow control," IEEE Transactions on Parallel and Distributed Systems, no. 2, pp. 194-205, 1992.

[20] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", IEEE Transactions on Computers, pp. 547-553, 1987.

[21] F. Feliciian and S.B. Furber, "An asynchronous on-chip network router with quality-of-service (QoS) support", in Proceedings of the IEEE International SOC Conference, pages 274-277, 2004.

[22] Xilinx Inc.,"FPGA vs. ASIC", http://www.xilinx.com/fpga/asic.htm.

[23] Altera Inc., "What is an FPGA?", http://www.altera.com/products/fpga.html.

[24] "Logic Structure Comparison Between Stratix II and Virtex-Based Architectures," Altera White Paper, 2004.

[25] "Innovations at 14 nm and 20 nm the Next-Generation Advantage," http://www.altera.com/technology/system-tech/next-gen-technologies.html.

[26] "PowerPlay Power Analyzer", http://www.altera.com/literature/hb/qts/qts_ qii53013.pdf.

[27] "Power Optimization, Quartus II Handbook version 13.0", http://www.altera.com/literature/hb/qts/qts_ qii52016.pdf.

[28] "The worlds first 3D heterogeneous all programmable product", EETimes 2012, "https://www.eetimes.com/electronics-products/electronic-product-reviews/fpga-pld-products/4374071/Xilinx-ships-the-world-s-first-heterogeneous-3D-FPGA".

[29] T. Bartic, A. Mignolet, J-Y. Nollet, T. Marescaux, D. Verkest, S. Vernalde, R. Lauwereins, "Topology adaptive network-on-chip design and implementation", in Proceedings of IEE on Computers and Digital Techniques, vol.152, no.4, pp.467-472, 2005.

[30] T. Marescaux, A. Bartic, D. Verkest, R. Lauwereins, S. Vernalde, and R. Lauwereins, "Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking On FPGAs", in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), pages 795-805, 2002.

[31] M. Saldana, L. Shannon, Jia Shuo Yue, Sikang Bian, J. Craig, P. Chow, "Routability of Network Topologies in FPGAs", IEEE Transactions on Very Large Scale Integration (TVLSI) Systems, vol.15, no.8, pp.948-951, Aug. 2007.

[32] C. Hilton and B. Nelson, "PNoC: a flexible circuit-switched NoC for FPGA-based systems", in Proceedings of IEE Computers and Digital Techniques, vol.153, no.3, pp.181-188, May 2006.

[33] Nachiket Kapre, Nikil Mehta, Michael Delorimier, Raphael Rubin, Henry Barnor, Michael J. Wilson, Michael Wrighton, and Andr Dehon, "Packet switched vs. time

multiplexed FPGA overlay networks", in Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 205-216, 2006.

[34] Axel Jantsch and Hannu Tenhunen, "Networks on Chip", Kluwer Academic Publishers, 2003.

[35] W.J. Dally and B. Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmann Publishers, 2004.

[36] D.E. Culler, J.P. Singh, and A. Gupta, "Parallel Computer Architecture:A Hardware/Software Approach", Morgan Kaufmann Publishers, 1998.

[37] R. Marculescu, U.Y. Ogras, L.S. Peh, N. Enright Jerger, and Y. Hoskote, "Outstanding research problem in NoC design: System, Micro Architecture, and Circuit perspectives", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), pp. 3-21, Vol. 28, Jan. 2009.

[38] D. Soudris, C. Piguet and C. Goutis, "Designing CMOS circuits for low power", Kluwer Academic Publishers, ISBN: 1-4020-7234-1, 2004.

[39] Janarthanan, Arun, Vijay Swaminathan, and Karen A. Tomko, "MoCReS: an Area-Efficient Multi-Clock On-Chip Network for Reconfigurable Systems", in Proceedings of IEEE Computer Society Annual Symposium on VLSI, pp. 455-456, 2007.

[40] H.-S. Wang, L.-S. Peh, and S. Malik, "A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers", in Proceedings of Hot Interconnects 10, pp. 21-27, Aug. 2002.

[41] A. B. Kahng, B. Li, L.-S. Peh and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration", in Proceedings of IEEE Design Automation and Test (DATE), pp. 423-428, 2009.

[42] M. DallOsso, G. Biccari, L. Giovannini, D. Bertozzi and L. Benini, "Xpipes: A Latency Insensitive Parameterized Network-on-Chip Architecture for Multiprocessor SoCs", in Proceedings of IEEE ICCD, pp. 536539, 2003.

[43] Kahng, Andrew B., Bill Lin, and Kambiz Samadi, "Improved on-chip router analytical power and area modeling", in Proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 241-246, 2010.

[44] S. E. Lee and N. Bagherzadeh, "A High Level Power Model for Network-on-Chip (NoC) Router", Elsevier journal on the VLSI INTEGRATION , pp. 1-7, 2009.

[45] TT. Ye , De Micheli, L. Benini, "Analysis of power consumption on switch fabrics in network routers", in Proceedings of IEEE Design Automation Conference (DAC), pp. 524-9, 2002.

[46] Raymond Sung, Andrew Sung, Patrick Chan, Jason Mah, " $http : //www.ece.ualberta.ca/\ elliott/ee552/studentAppNotes/1999f/Drivers_Ed/lfsr.html$".

[47] Booksim simulator, "$http : //nocs.stanford.edu/cgi - bin/trac.cgi/wiki/Resources/BookSim$".

[48] Wormsim simulator, "$http : //www.ece.cmu.edu/\ sld/wiki/doku.php?id = shared : worm_sim$".

[49] Noxim simulator, "$http : //noxim.sourceforge.net/$".

[50] Nirgam simulator, "$http : //nirgam.ecs.soton.ac.uk$".

[51] Atlas simulator, "$https : //corfu.pucrs.br/redmine/projects/atlas$".

[52] Xmulator simulator, "$http : //sourceforge.net/projects/xmulator/$".

[53] R. Marculescu, U.Y. Ogras, L.S. Peh, N. Enright Jerger, and Y. Hoskote, "Outstanding research problem in NoC design: System, Micro Architecture, and Circuit perspectives", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), pp. 3-21, Vol. 28, Jan. 2009.

[54] Mohammad Arjomand, and Hamid Sarbazi-Azad, "Power-performance analysis of networks-on-chip with arbitrary buffer allocation schemes", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), no. 10, pp. 1558-1571, 2010.

[55] Ted H. Szymanski, Honglin Wu, and Amir Gourgy, "Power complexity of multiplexer-based optoelectronic crossbar switches", IEEE Transactions on Very Large Scale Integration (TVLSI) Systems, no. 5, pp. 604-617, 2005.

[56] James Balfour, and William J. Dally, "Design tradeoffs for tiled CMP on-chip networks", in Proceedings of ACM international conference on Supercomputing, pp. 187-198, 2006.

[57] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis", in Proceedings of IEEE DATE, pp. 1096-1101, 2007.

[58] U. Y. Ogras, P.Bogdan, and R. Marculescu, "An analytical approach for Network-on-Chip performance analysis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), pp.2001-2013, Dec. 2010.

[59] H. S. Wang, L. S. Peh, and S. Malik, "A power model for routers: Modeling Alpha 21364 and InfiniBand routers", in Proceedings of IEEE/ACM International Symposium on Microarchitecture, vol. 23, no. 1, pp. 26-35, Feb. 2003.

[60] Jian Wang, and T. H. Szymanski, "Power analysis of Input-Queued and Crosspoint-Queued crossbar switches", in Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, pp. 273-278, 2009.

[61] John Kim, James Balfour, and William Dally, "Flattened butterfly topology for on-chip networks", in Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 172-182, 2007.

[62] V. Dumitriu, G.N. Khan, "Throughput-Oriented NoC Topology Generation and Analysis for High Performance SoCs", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17, no.10, pp.1433-1446, Oct. 2009

[63] Murali, Srinivasan, and Giovanni De Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs", in Proceedings of IEEE Design Automation Conference (DAC), pp. 914-919, 2004.

[64] Terrence Mak, Crescenzo D'Alessandro, Pete Sedcole, Peter YK Cheung, Alex Yakovlev, and Wayne Luk, "Global interconnections in FPGAs: modeling and performance analysis", in Proceedings of ACM on System level interconnect prediction, pp. 51-58, 2008.

[65] Mak, Terrence, Pete Sedcole, Peter YK Cheung, and Wayne Luk, "Wave-pipelined intra-chip signaling for on-FPGA communications", Elsevier Journal on the VLSI Integration, no. 2, pp. 188-201, 2010.

[66] Wang, Hangsheng, Li-Shiuan Peh, and Sharad Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks", in Proceedings of IEEE in Design, Automation and Test (DATE), pp. 1238-1243, 2005.

[67] Kuon, Ian, and Jonathan Rose, "Measuring the gap between FPGAs and ASICs", IEEE

Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), no.2, pp. 203-215, 2007.

[68] Y. Lu, J. McCanny, and S. Sezer, "Exploring Virtual-Channel architecture in FPGA based Networks-on-Chip", in Proceedings of IEEE International SOC conference (SOCC), pp. 302-307, Sept. 2011.

[69] S. E. Lee and N. Bagherzadeh, "A high level power model for Network-on-Chip (NoC) router", Elsevier journal on Computers and Electrical Engineering, pp. 837-845, 2009.

[70] Bafumba-Lokilo, David, Yvon Savaria, and J-P. David, "Generic crossbar network on chip for FPGA MPSoCs", in Proceedings of IEEE in Circuits and Systems, pp. 269-272, 2008.

[71] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, "Fundamentals of Queueing Theory", Wiley- Interscience, New York, NY, USA, 4th edition, 2009.

[72] M. Binesh Marvasti, and T. H. Szymanski, "A power-area analysis of NoCs in FPGAs", in Proceedings of 25th IEEE International SOC Conference (SOCC), 2012.

[73] M. Binesh Marvasti, and T. H. Szymanski, "The performance of hypermesh NoCs in FPGAs," In Proceedings of 30th IEEE International Conference on Computer Design (ICCD), 2012.

[74] M. Binesh Marvasti and T.H. Szymanski, "An Analysis of Mesh and Hypercube NoCs in FPGAs," Submitted to IEEE Transactions on CAD, 2013.

[75] M. Binesh Marvasti and T.H. Szymanski, "An Analysis of Hypermesh NoCs in FPGAs," Submitted to IEEE Transactions on Parallel and Distributed Systems (TPDS), 2013.

[76] L. Kleinrock, "Queueing Systems", Volume II. Computer Applications. Wiley-Interscience, New York, 1976.

[77] T.H Szymanski, "The complexity of FFT and Related Butterfly Algoruthm on Meshes and Hypermeshes", in Proceedings of IEEE International Conference on Parallel Processing, 1992.

[78] N. Alzeidi, M. Ould-Khaoua, L. M. Mackenzie, and A. Khonsari, "Performance analysis of adaptively-routed wormhole-switched networks with finite buffers", in Proceedings of IEEE International Conference on Communications (ICC'07), pp. 38-43, 2007.

[79] Paul Bogdan, and Radu Marculescu, "Non-stationary traffic analysis and its implications on multicore platform design", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), no. 4, pp. 508-519, 2011.

[80] Z. Guz, I. harWalter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Network Delays and Link Capacities in Application-Specific Wormhole NoCs", Journal of VLSI Design Hindawi, 2007.

[81] Ted H. Szymanski, "A fiber optic hypermesh for SIMD/MIMD machines", in Proceedings of IEEE Supercomputing'90, pp. 710-719, 1990.

[82] Wittie, Larry D, "Communication structures for large networks of microcomputers", IEEE Transactions on Computers (TC), no. 4, pp. 264-273, 1981.

[83] T.H. Szymanski, "Hypermeshes: Optical Interconnection Networks for Parallel Computing", Journal of Parallel and Distributed Computing (JPDC), Vol. 26, pp. 1-23, Apr. 1995.

[84] T.H. Szymanski and H.S. Hinton, "Architecture of a Terabit Free-Space Photonic Backplane", JPDC, Vol. 55, No. 1, November 25, 1998, pp. 1-31.

[85] T.H. Szymanski and H.S. Hinton, U.S. Patent # 6,016,211, "Optoelectronic Smart Pixel Array for a reconfigurable intelligent optical interconnect", Issued Jan. 2000.

[86] M. Ould-Khaoua, "Distributed crossbar switch hypermeshes: Efficient networks for large-scale multicomputers", Journal of Parallel and Distributed Computing, no. 8, pp. 999-1012, 2001.

[87] Metzgen, Paul, and Dominic Nancekievill, "Multiplexer restructuring for FPGA implementation cost reduction", in Proceedings of the 42nd annual Design Automation Conference, pp. 421-426. ACM, 2005.

[88] R. Moraveji, H. Sarbazi-Azad, A. Nayebi, K. Navi, "Modelling the effects of hot-spot traffic load on the performance of wormhole-switched Hypermeshes", Elsevier Journal of Comp. and Electrical Engineering, pp. 1-23, Jan. 2011.

[89] S. Loucif, M. Ould-Khaoua, A. Al-Ayyoub, "Hypermeshes: implementation and performance", Journal of Systems Architecture, pp. 37-47, Sept. 2002.

[90] M. Ould-Khaoua, R. Sotudeh, "Communication locality in hypermeshes and tori", in Proceedings of IEEE Architectures for Parallel Processing, pp. 256-262, Jun. 1996.

[91] P-C Hu, and L. Kleinrock, "An analytical model for wormhole routing with finite size input buffers", 15th International Telegraphic Congress, 1997.