A POWER ANALYSIS OF THE COMBINED INPUT AND CROSSPOINT QUEUED SWITCH

A POWER ANALYSIS OF THE COMBINED INPUT AND CROSSPOINT QUEUED SWITCH

BY

JIAN WANG, B.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 REQURIEMTNS

FOR THE DEGREE OF

MASTER OF APPLICED SCIENCE

Master of Applied Science (2009) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE: A Power Analysis of the Combined Input and Crosspoint Queued Switch

AUTHOR: Jian Wang

B. Sc., (Computer Engineering)

McMaster University, Hamilton, ON, Canada

SUPERVISOR: Dr. T. H. Szymanski

NUMBER OF PAGES: 139

Abstract

Crossbar switches are fundamental building blocks of digital networks, such as the Internet. An *Input-Queued* (*IQ*) switch with a set of queues at the input side is the most widely used crossbar switch due to the scalability and low design complexity. While *IQ* switches are used in commercial routers, the schedulers are relatively complex. An alternative switch design is to insert a FIFO queue in each crosspoint of an *IQ* switch. This architecture is named the Combined Input and Crosspoint Queued switch, denoted as *CIXQ*. The use of crosspoint queues simplifies the scheduling of traffic through the switch, at the cost of adding N^2 FIFO queues to the switch fabric.

In recent years, considerable research has been made to study the capability and flexibility of the *CIXQ* switch over high-speed networks. In this thesis, we will review the *CIXQ* switch by focusing on the switching performance, design implementation and power consumption. The switching performance is first evaluated using MATLAB. The analysis results show that the *CIXQ* switch with a simple Round Robin scheduling can achieve a high performance under various traffic patterns. In an attempt to analyze the power consumption, a *CIXQ* switch is then implemented in VHDL based on a broadcast-and-select architecture and a DEMUX-MUX architecture. The designs are tested and simulated on Altera Cyclone II FPGAs. The simulation results illustrate that the switches are operational correctly as specified in Chapter 4. As power has become a critical design issue, a power analysis for *IQ* and *CIXQ* switches is finally presented. Analytic power models for the switches in an FPGA environment are developed. As the major contribution of this thesis, we are the first to develop power models for the *CIXQ* switch. The power models allow the designers to explore the power efficiency of various crossbar switches in the early stages of design process. The verification indicates that the power models are quite accurate with an average error of less than 10%. The analysis also shows that a *CIXQ* switch with N^2 crosspoint queues consumes about three times as much power as the *IQ* switch for modest size switches (*4x4, 8x8*, and *16x16*) in an FPGA environment.

In conclusion, we present an analysis of the design implementation and power consumption of the *CIXQ* switch. The discussions illustrate a fact that the *CIXQ* switch with a FIFO queue in each crosspoint increases the cost and hardware complexity even though it will simplify the scheduling problem. The designers must analyze this trade-off when making architectural design decisions.

Acknowledgements

I would like to thank all those people who helped me throughout this work. First of all, I would like to express enormous appreciation to my supervisor, Dr. Ted. H. Szymanski, for his encouragement, guidance, patience, and support during the period of my studies. Dr. Szymanski provides me with the opportunity to work on such an interesting and exciting project. Without his priceless suggestions, this project would not have reached this stage.

Second of all, I would also like to thank Dr. Dave Gilbert, who not only helps me to organize and improve the thesis, but also give me valuable suggestions and encouragements during my research. Besides, I would like to thank my committee members, Dr. T. Todd and Dr. N. Nicolici, for their support, kindness, and insightful suggestion throughout this thesis.

I would like to extend my special thank to all the administrative and technical staff at the department of Electrical and Computer Engineering that made the completion of this project possible, especially to Terry Greenlay and Cosmin Coroiu for setting up the hardware and software for the experimental purpose, and Steve Spencer and Tyler Ackland for providing the experimental devices for my research. Furthermore, I would also like to thank the members of Altera support team for patiently answering my numerous questions about the design devices and tools.

Finally, I wish to thank my parents for their encouragement and support throughout all my academic years. I also thank my younger sisters and little brother for their understanding. Most importantly, I would like to dedicate this thesis to my beloved grandma, who passed away recently. Thanks you grandma for everything you have done for me.

Notation and Abbreviation

I_{SW}	Switching current
C_{int}	Intrinsic capacitance
C_{load}	Load capacitor
V_{DD}	Core voltage
I_{SC}	Short circuit current
f_{clk}	Clock frequency
f_{duty}	Duty cycle
N_{trans}	Average number of transitions per clock cycle
μm	Micron
$P_{static\_per\_component}$	Static power dissipated by each component in the design
K	Power coefficient
$K_{DEMUX1m}$	Power constant for 1-to-m DEMUX cell
$K_{DEMUX12}$	Power constant for 1-to-2 DEMUX cell
K_{MUXm1}	Power constant for m-to-1 MUX cell
K_{MUX21}	Power constant for 2-to-1 MUX cell
K_{FIFO}	Power constant for a FIFO queue
K_{Bus1N}	Power constant for broadcast bus power in an *NxN* switch
S_i	Average switching activity
$S_{DEMUX1m}$	Synthesis efficiency for 1-to-m DEMUX cell
$S_{DEMUX12}$	Synthesis efficiency for 1-to-2 DEMUX cell
S_{MUX21}	Synthesis efficiency for 2-to-1 MUX cell
C_i	Capacitance related to the particular logic block or module
$N_{LE-DEMUX}$	Logic elements used in the 1-to-N DEMUX

N_{LE-MUX}	Logic elements used in the N-to-1 MUX
V_{swing}	Swing voltage
V_{supply}	Supply voltage
P_{total}	Total power
P_{static}	Static power
$P_{static-per-component}$	Static power dissipated by each component
$P_{dynamic}$	Dynamic power
P_{MAX}	Maximum allowed power dissipation in the device
$P_{DEMUX-MUX}$	Power of the CIXQ switch in the DEMUX-MUX design
P_{DEMUX}	DEMUX power
P_{MUX}	MUX power
P_{BUS}	Broadcast bus power
$P_{(Bus1N)}$	Broadcast bus power in an *NxN* switch
P_{XQ}	Crosspoint queue power
$P_{B\&S}$	Power of the CIXQ switch in the broadcast-and-select design
$P(XQ)$	Crosspoint queued switch power
$P(IQ)$	Input queued switch power
$P_{DEMUX1N}$	1-to-N DEMUX power
$P_{DEMUX1m}$	1-to-m DEMUX power
P_{MUXN1}	N-to-1 MUX power
P_{MUXm1}	m-to-1 MUX power
$P_{(FIFONw)}$	FIFO power with N w-bit words
$P(I)$	Probabilities of being in IDEL state of a FIFO queue
$P(W)$	Probabilities of being in WRITE state of a FIFO queue

$P(R)$	Probabilities of being in READ state of a FIFO queue
$P(RW)$	Probabilities of being in READ and WRITE state of a FIFO queue
T_A	Ambient temperature
θ_{JA}	Thermal resistance
T_C	Case temperature
θ_{JC}	Junction-to-case thermal resistance
p	Input traffic load
$f(x)$	Probability function
x	Bernoulli process
p_{ij}	The traffic load for a cell at input i destined for output j
ASIC	Application Specific Integrated Circuit
B&S	Broadcast-and-Select structure
BMX	Bus Matrix Switch
CAD	Computer-Aid Design
CDFG	Control Data Flow Graph
CES	Chip Estimation System
CIXQ	Combined Input and Crosspoint Queued switch
CMOS	Complementary Metal-Oxide Semiconductor
CoS	Classes of Service
DEMUX	Demultiplexer
DEMUX-MUX	Demultiplexer-and-Multiplexer structure
DRRM	Dual Round-Robin Matching
DSP	Digital Signal Processing
FIFO	First Input and First Out
FPGA	Field Programmable Gate Array
Gpbs	Gigabits per second
HOL	Head of Line
i.i.d	Independent and identical distribution

IP	Input Port
IQ	Input Queue or Input-Queued switch
k	Crosspoint Queue Size
LB	Logic Block
LE	Logic Element
LQF	Longest Queued First
LQFS	Longest Queue First Served
LUT	Look up table
m	1-to-m DEMUX cell
MUX	Multiplexer
MWM	Maximum Weighted Matching
N	Switch size
n	Number of Iterations in iSLIP
NAP-NET	Network Access Processor Network
OCF	Oldest Cell First
OP	Output Port
OQ	Output Queue or Output-Queued switch
PDSS	Profile Driven Synthesis System
PFA	Power Factor Approximation
PIM	Parallel Iterative Matching
QoS	Quality of Service
RR	Round Robin scheduling
RTL	Register-Transfer Level
RTT	Long Round-Trip Time
S	Speedup factor
SAR	Segmentation and Reassembly
SoC	System-on-a-Chip
T	Time Interval
Tbps	Terabits per second

TDM	Time Division Multiplexer
u	Unbalanced probability
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLSI	Very-Large-Scale Integration
VOQ	Virtual Output Queue
w	Data path width
WRR	Weighted Round-Robin
XP	Crosspoint
XQ	Crosspoint Queue or Crosspoint-Queued Switch

Table of Contents

Table of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

Crossbar switches have been widely used in packet switching networks due to their internally non-blocking capability and the ability to support multicast. Queues are use to resolve the contention that remains in the crossbar switches [1]. In general, the crossbar switches are categorized into three classes based on the location of queueing functions. They are Input-Queued (*IQ*) switches, Output-Queued (*OQ*) switches and Crosspoint-Queued (*XQ*) switches with a queueing function at the input port (*IP*), output port (*OP*) and crosspoint (*XP*), respectively.

OQ switches with speedup *N* are the ideal packet switching architecture for providing *Quality of Service (QoS)* guarantees [2]. However, the implementation is not scalable due to the required speedup. *IQ* switches place a queueing function at each input port to overcome the scalability problem. Many papers prove that *IQ* switches with appropriate scheduling algorithms can achieve 100% throughput [7, 8, 9, 10, 11, 12]. While *IQ* switches are used in commercial routers, the schedulers are relatively complex. *XQ* switches with making a *First-In-First-Out* (FIFO) queue in each crosspoint have been introduced to simplify the scheduling since they can schedule the packets at the input and outputs independently and in parallel. A special version of *XQ* switches that combines crosspoint queueing and input qeueueing, denoted as *CIXQ*, has recently been proposed to lower the memory required in the crossbar and maximize throughput [19].

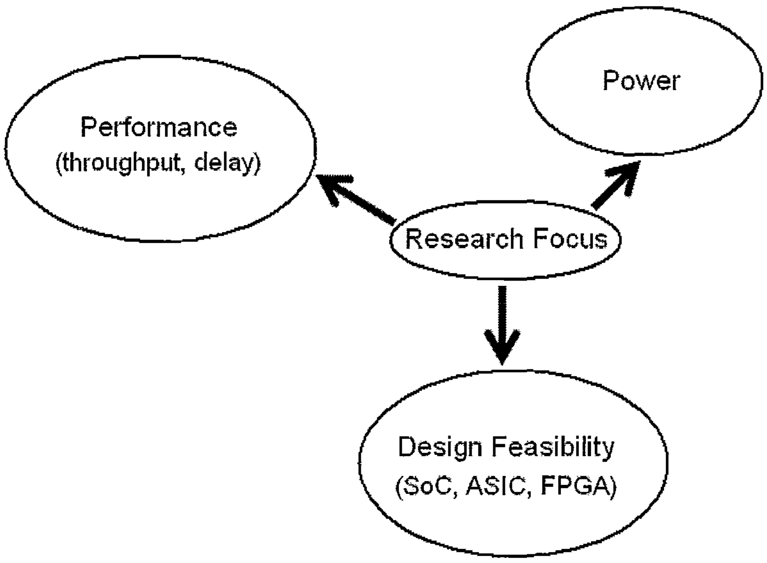


FIGURE 1.1: Rearch focus of *CIXQ* switches

CIXQ switches have been the subject of considerable research over the last decade. The research focus in *CIXQ* switches lies in three areas: the performance, design feasibility and power consumption (see FIGURE 1.1). The majority of the literature is focused on the performance in terms of throughput and delay. The simulations results in [19, 20, 21, 22, 23, 24, 25, 26] indicate *CIXQ* switches exhibit a marginally lower delay than *IQ* switches while providing a comparably high throughput. These studies claim that *CIXQ* switches with less complex scheduling have great potential to complement or even replace *IQ* switches in the future high-speed networks.

Unfortunately, the technology to build *CIXQ* switches appears to be an obstacle because it is difficult and expensive to deploy a queue in each crosspoint. *CIXQ* switches were only considered theoretical architectures before the *Complementary Metal-Oxide Semiconductor* (CMOS) technology allowed the integration of a large amount of memory inside a single chip [19]. The advancement of CMOS technology significantly reduces the cost associated with the development and construction of a buffered crossbar. Recently, a number of groups have analyzed the feasibility of building *CIXQ* switches on a CMOS *Very-Large-Scale Integration* (VLSI) chip, such as *System-on-a-Chip* (SoC), *Application Specific Integrated Circuit* (ASIC) or *Field Programmable Gate Array* (FPGA). The analysis in [68, 70] reveal that the *CIXQ* switch architectures are feasible, and should be available for use in the switching networks.

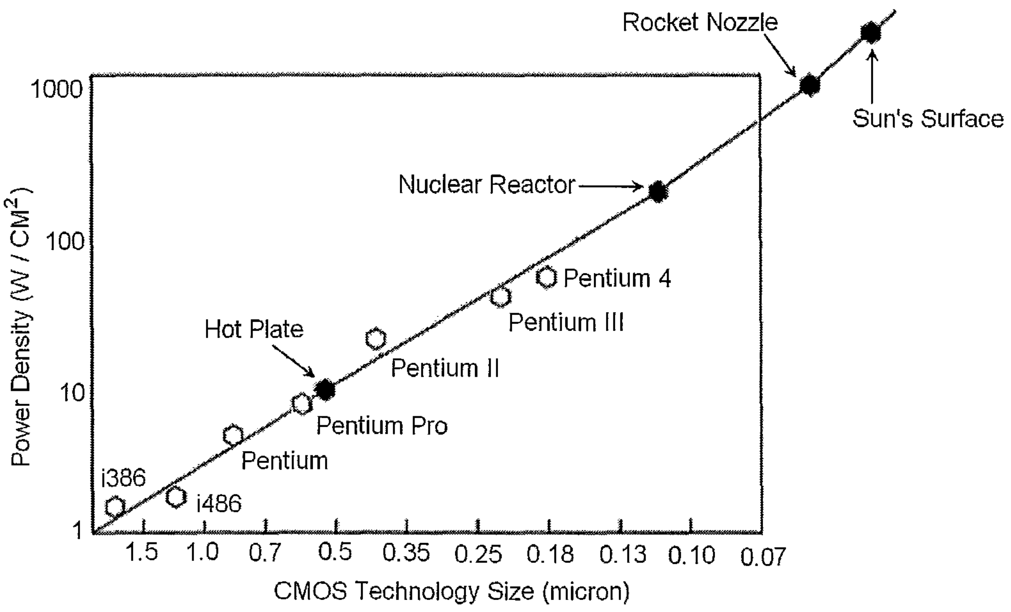


FIGURE 1.2: Power density trends of microprocessor

Power has become a first-class design constraint in the digital circuits, especially in the CMOS VLSI circuits [3]. The power dissipation rises exponentially while the CMOS technology continues to use smaller manufacturing process. The microprocessor is a VLSI device. FIGURE 1.2 depicts the power density trends of the Pentium microprocessor versus the transistor size [4]. The Pentium 4 has a power density of about 50 W/cm^2, which is seven times more than the Intel 486. The trend cannot continue because it is not practical to remove heat from a chip built using the 0.10-micron process with current design methods. Although it is crucial to develop techniques to reduce the power consumption, it is also essential to have power estimation tools to explore the power efficiency when the circuit designers make architecture trade-offs in the early design stages. Today, many semiconductor vendors have embedded power estimators in the *Computer-Aid Design* (CAD) tools, for example, Xilinx's *Xpower* [55] and Altera's *PowerPlay* [57].

As *CIXQ* switches are feasible to implement in a CMOS VLSI circuit, the question of whether they will have a power consumption problem becomes a very important concern. Excessive power consumption will not only limit the use of *CIXQ* switches, but will also degrade the performance due to overheating. A sophisticated power analysis tool needs to be developed for investigating the *CIXQ* switches. Accordingly, the designers can determine how well the devices meet the power specifications without a costly redesign process. Recently, researchers have focused on this goal, but only a limited number of publications are available. To our best knowledge, D. G. Simos [70] was the first person to measure the power consumption of an ASIC implementation of *CIXQ* switch. To date, no power model has been developed for any *CIXQ* switch. This is a major contribution made by this thesis.

1.2 Research Motivation and Objective

The purpose of this thesis is to analyze the power consumption of *CIXQ* switches using current CMOS technology. The performance of *CIXQ* switches will be studied using

MATLAB under various traffic conditions. Different hardware architectures of *CIXQ* switches are also created and analyzed in FPGA environment. In particular, we wish to study the power consumption of *CIXQ* switches realized in an FPGA. Analytical models are developed to estimate the power of the designs at Register-Transfer Level (RTL). To verify the analytical models, the actual power of several switches is measured by Altera™ *PowerPlay* power tool. The models should be accurate and flexible to explore the power in the early stage of design process.

1.3 Contributions

The contributions of this thesis are summarized as follows:

1. A preliminary performance evaluation of *CIXQ* switches using MATLAB. The experiments indicate that the *CIXQ* switch with round robin arbitration has marginally better performance than an *IQ* switch with iSLIP, which is consistent to the discussions in [21, 22]. However, no thorough study of scheduling algorithm was undertaken, and no conclusion should be drawn from these simulations. A thorough study of scheduling algorithms is beyond the scope of this thesis.

2. A VHDL implementation of *CIXQ* switches employing the Broadcast-and-Select architecture and DEMUX-MUX architecture are described. The simulation results show the designs are functional properly as described in this thesis.

3. Analytic power models for the *CIXQ* switch in an FPGA environment are developed and have been published at the 22nd Canada Conference on Electrical and Computer Engineering [5]. To the best of our knowledge, no power model for the *CIXQ* switch has been made prior to our work.
4. The power models are accurate with an average error of about 10%. They can allow the designers to check the power efficiency of different implementations and make architectural design decisions early in the design process, without actually going through chip fabrication.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides a literature review of *CIXQ* switches considering the switching performance, design implementation and power consumption. It also reviews the unbuffered crossbar switches, which include *Input-Queued* (*IQ*) switches and *Output-Queued* (*OQ*) switches. Chapter 3 presents a preliminary performance evaluation of the *CIXQ* switch using Round Robin (RR) arbitration in comparison of the *IQ* switch using the iSLIP scheduler. Chapter 4 describes a VHDL implementation of the *CIXQ* switch using Altera Cyclone II FPGAs. Chapter 5 proposes the analytical power models for *IQ* and *CIXQ* switches. Finally, Chapter 6 draws the conclusion of the thesis and presents possible direction of the future research.

Chapter 2 Literature Review

2.1 Introduction

The *Combined Input- and Crosspint-Queued* (*CIXQ*) switches simplify the scheduling by adding N^2 crosspoint queues to the basic crossbar switch. Their performance, implementation and power consumption have recently captured the interest of many researchers. In this chapter, we present a review of *CIXQ* switches in the literature. The chapter is organized as follows: first we review the traditional *Input-Queued* (*IQ*) switch as well as the *Output-Queued* (*OQ*) switch. Next, we describe the history and development of the *CIXQ* switch followed by a discussion about its implementation using current CMOS technology. Then, we discuss the advantages and disadvantages of having N^2 crosspoint queues in *CIXQ* switches. We also provide a survey about the techniques and CAD tools for estimating the power consumption of CMOS VLSI designs; we particularly review the publications that have been done so far to analyze the power consumption of crossbar switches. Finally, we present a summary about the research focus of *CIXQ* switches.

2.2 Unbuffered Crossbar Switches: *OQ* and *IQ* Switches

Before Bakka and Dieudonne [13] proposed the idea of employing a FIFO queue in each crosspoint, *OQ* and *IQ* switches were the two main crossbar switch designs discussed in the literature. In an *OQ* switch, each output port maintains a queue to store the incoming packets. This architecture requires an internal bandwidth or speedup that is equal to the number of ports ($S = N$), i.e. an *NxN OQ* switch is able to move up to N packets from N inputs to a single output port in one packet time slot (see FIGURE 2.1). S is defined as the speedup factor. The speedup requirement not only limits the scalability of *OQ* switches but also makes them too expensive to build for large N. Consequently, *OQ* switches are not practical for packet switching network.

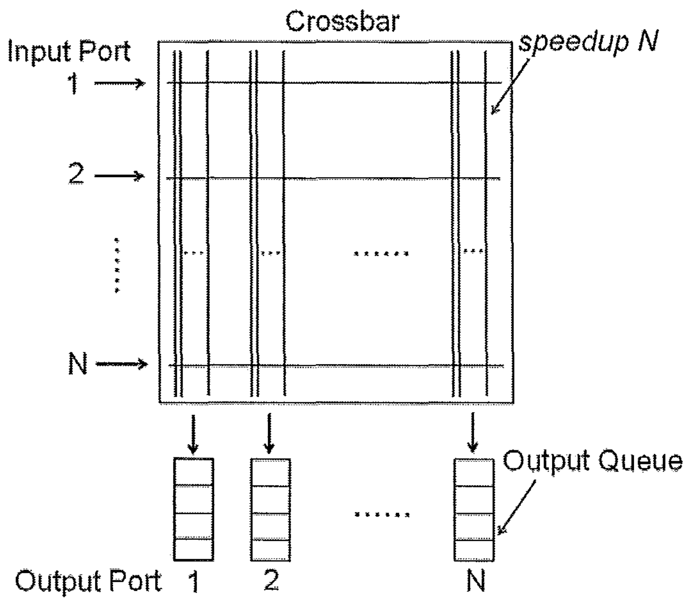


FIGURE 2.1: An OQ switch with speedup N

IQ switches overcome the scalability problem by having a queueing function at each input port. In fact, they are dominant in high speed switching. The performance of *IQ* switches depends very much on the use of the input queues. If a FIFO queue is applied at each input port, the throughput is limited to about 58.6% due to the *Head of Line* (HOL) blocking problem for the random uniform traffic with *independent and identical* *distribution* (i.i.d) [6]. In fact, the HOL blocking can be eliminated entirely using a scheme of *Virtual Output Queues* (VOQs) in which each input port maintains a separate queue for each output. It has been shown that the throughput of an *IQ* switch with VOQs can be increased up to 100% [10]. The block diagram of an *NxN IQ* switch with *N* VOQs at the input side is illustrated in FIGURE 2.2.

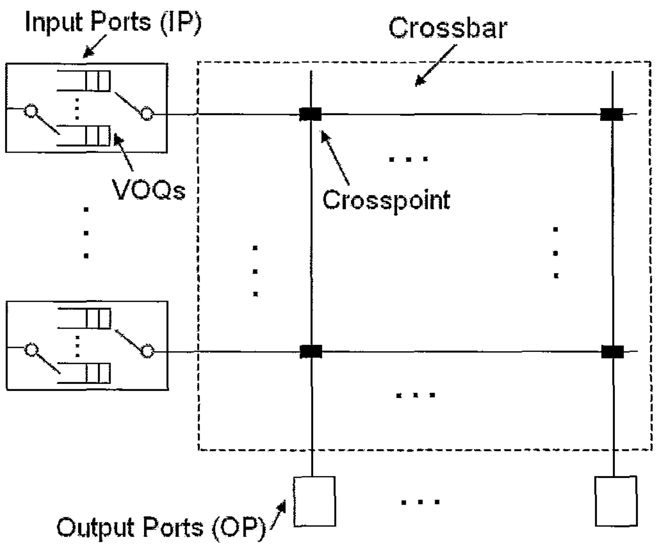


FIGURE 2.2: Input-queued crossbar switch with VOQs

In an *NxN IQ* switch, the packets are typically fragmented into fixed-size cells at the input side. The cells are transmitted through the switch while attempting to meet *Quality of Service* (QoS) constraints. A scheduling algorithm is required to direct the transmission of cells from the input side to output side. According to [7, 8, 9, 10], scheduling in *IQ* switches is a difficult problem. In each cell time slot, the scheduler must identify up to *N* cells to transmit through the switch, while simultaneously meeting three constraints: 1) no input port transmits more than one cell per time-slot, 2) no output port receives more than one cell per time-slot, and 3) each traffic flow receives relatively fair service.

The scheduling in an *IQ* switch can be formulated a *Bipartite Graph Matching* problem [7]. During each time slot, a scheduling algorithm determines the transmission of cells through the switch by finding a matching on a bipartite graph. It has been established that the *Maximum Weighted Matching (MWM)* algorithm can provide 100% throughput in *IQ* switches with unity speedup for all admissible i.i.d arrivals [10]. However, the complexity of such algorithms renders them impractical. Given a line rate of 40 Gbps and a cell size of 64 bytes, the time-slot duration is 12.8 nanoseconds. The time complexity of MWM algorithm is $O(N^3)$. This requirement renders MWM far too computationally complex for current IP switches or routers. A number of algorithms have been proposed to approximate the performance of MWM, for example, *Parallel Iterative Matching* (PIM) [9], iSLIP [7] and *Dual Round-Robin Matching* (DRRM) [8]. These historic scheduling algorithms do not have as good performance as MWM.

The recent theoretical result [11, 12] shows that *IQ* switches with a speedup of *S* (where *S* is the speedup factor) can achieve optimal performance as *OQ* switches. These switch designs can remove up to S cells from each input and move up to S cells to each output in a cell time slot. They are generally called Combined Input and Output Queued (*CIOQ*) switches because a queue is required at each output port. S. T. Chuang *et al.* [11] and I. Stocia *et al.* [12] both stated that a *CIOQ* switch with a speedup of two ($S = 2$) can exactly emulate an *OQ* switch. The scheduling problem remains complicated, and the use of CI*OQ* switches is thus discouraged in the packet switching networks.

2.3 History

The switches with a FIFO queue in each crosspoint have been proposed to simplify the scheduling problem. Prior to discussing the design difficulties of *CIXQ* switches, it is important to understand how they were developed. The idea of inserting a queue at each crosspoint of a crossbar switch was first introduced by Bakka and Dieudonne [13] in 1982, and these switch architectures are called Crosspoint-Queued (*XQ*) switches. Nojimma *et al.* [14] were the first to study the performance of *XQ* switches in packet switching networks. In 1987, they implemented a *Bus Matrix* (BMX) switch, where the crosspoint queues were made from dual-port memories allowing asynchronous and individual operation at input and output ports. The performance of BMX switch was evaluated under the *Network Access Processor Network* (NAP-NET). The end-to-end delay of voice packets was measured in the experiments. The voice delay was less than 20 ms for the 64-byte packets after five-hop transmission. The experimental results demonstrated that the BMX switch had high performance.

The memory required at the crosspoints can be effectively reduced if FIFO queues are employed at the input side of *XQ* switches [15]. A significant amount of work has been done to investigate the performance of *XQ* switches with FIFO queues at the input side. A. Gupta *et al.* [16] claimed that a *XQ* switch could achieve 87.5% throughput for uniform traffic with the effect of *HOL* blocking. The same authors increased the throughput to 91% by proposing a *HOL* priority selection scheme at the input sides [17]. The performance of XQ switches was further studied by R. Fantacci *et al.* [18]. These researchers proved that a random selection scheme at the input side can establish 100% throughput in *XQ* switches for uniform traffic.

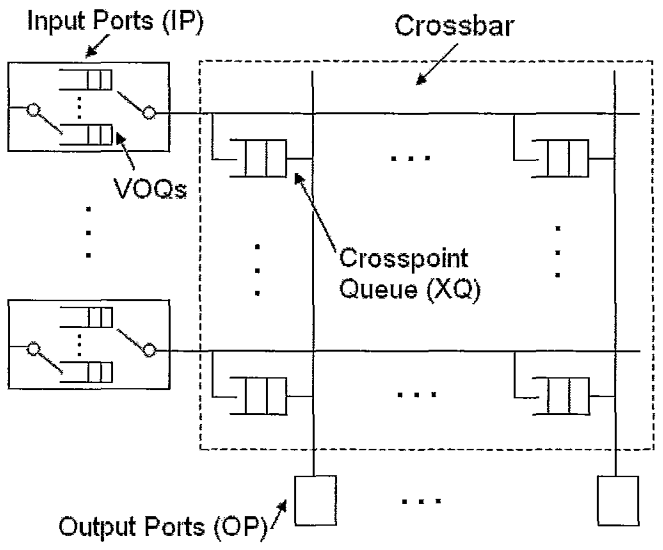


FIGURE 2.3: Crosspoint-queued crossbar switch with V*OQ*s

In 2000, switches combining *IQ* switches with V*OQ*s and *XQ* switches were first proposed by Nabeshima [19], and he referred to such architectures as *Combined Input- and Crosspoint-Queued switches*, short for *CICQ*. In this thesis, we use the terminology *"CIXQ"* instead. FIGURE 2.3 illustrates a *CIXQ* switch, which differs from an *IQ* switch by putting a FIFO queue in each crosspoint of the crossbar. *CIXQ* switches entirely eliminate *HOL* blocking with the use of V*OQ*s at the input side. More importantly, they reduced the amount of memory required in the crossbar significantly. Several scheduling algorithms for *CIXQ* switches have been introduced in the past ten years. K. Yoshigoe *et al.* [20] used *Round-Robin* (RR) arbitration at the inputs and outputs of a *CIXQ* switch. The simulation results showed that the *CIXQ* switch obtained a lower queueing delay than an *IQ* switch using the iSLIP scheduler for the fixed-size and variable-size packets. A *CIXQ* switch with one-cell crosspoint queues and RR arbitration at inputs and outputs was proposed by Rojas-Cessa *et al.* [21, 22]. This switch design provides a 100% throughput under uniform traffic, but it has a relatively low performance for non-uniform traffic. Y. Zheng *et al.* [23] then applied a *Dual Round-Robin* (DRR) algorithm to increase the performance of *CIXQ* switches under non-uniform traffic. The *CIXQ* switch with DRR achieved a satisfactory performance under both uniform and non-uniform traffics.

The impact of different scheduling algorithms on the performance of *CIXQ* switches were analyzed by M. Pejanovic *et al.* [24]. The scheduling algorithms at inputs and outputs of the switch were designed with the combinations of RR, *Longest Queued First* (LQF), and *Oldest Cell First* (OCF). The comparison results demonstrated that the *CIXQ* switches with various scheduling algorithms resulted in almost the same performances under various traffics. The results also implied that RR arbitration is a better solution for scheduling scheme in a *CIXQ* switch because its implementation is simple. Furthermore, the performance of *CIXQ* switches can be improved with the use of speedup. L. Mhamdi et al. [25] and R. B. Magill [26] had proved that a *CIXQ* switch with speed up of two can exactly emulate an *OQ* switch.

CIXQ switches have a 25 year history. The complexity of constructing buffered crossbars is reduced by maintaining a set of queues at each input port. Moreover, they are proven to obtain high throughput with reasonable delay under uniform and non-uniform traffic according to the studies in [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

2.4 Design and Implementation

CIXQ switches show promises for future networks because they can simplify the scheduling algorithm. However, the feasibility of implementing N^2 crosspoint queues remains an open question. Although the enhancement of CMOS technology makes the queueing design problem less difficult, the designers still face a lot of challenges in making a buffered crossbar switch.

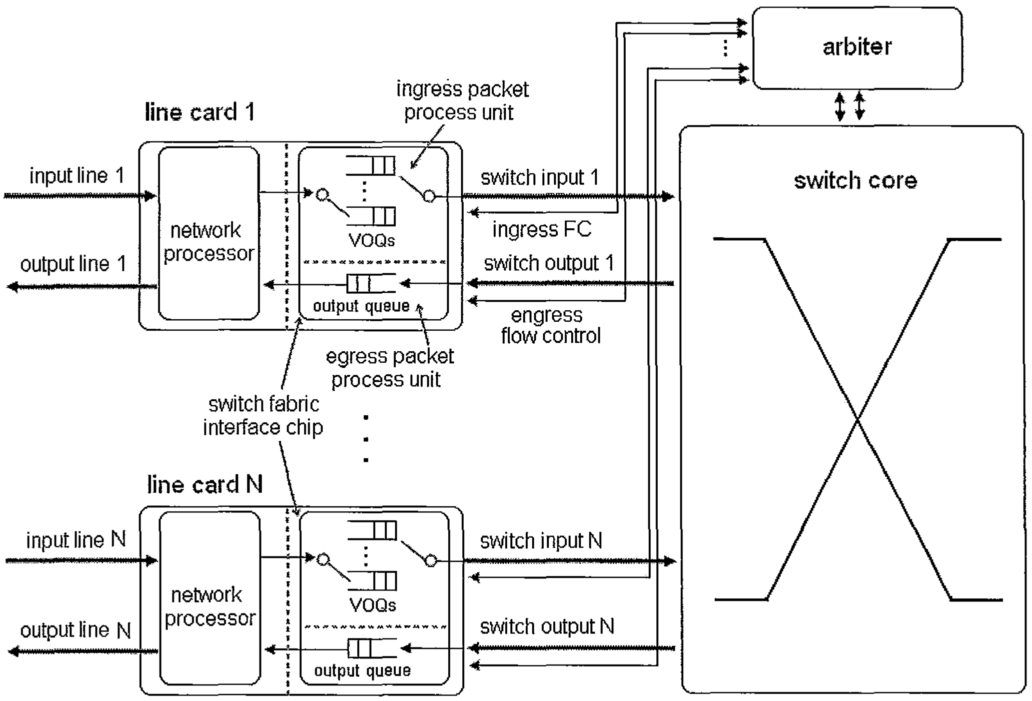


FIGURE 2.4: Generic Switch Architecture

From the hardware perspective, a network switching device normally consists four parts: 1) the ingress packet process unit, 2) the egress packet process unit, 3) the arbitration unit (arbiter), and 4) the switch core. FIGURE 2.4 shows the block diagram of switch/router architecture [27]. Each Input-Output (*IO*) port with the *OQ*s and V*OQ*s is typically realized on one line-card, which has several Gigabytes of queueing memory and one or more network processors to manage and schedule the traffic. The crossbar switching functions of IP routers are typically implemented on a separate "switch-core" VLSI substrate [28, 29]. In a large IP router, crossbar switch substrates with capacities of hundreds of Gigabits per second (Gpbs) up to several Terabits per second (Tbps) are required.

A crossbar switch can be made with one or more *System-on-a-Chip (SoC)* substrates or a large array of parallel *Field Programmable Gate Arrays (FPGAs)* in a state-of-the-art CMOS process. In the past decade, a number of groups analyzed the feasibility of buffered crossbar switches in silicon SoC and FPGA. Kariniemi *et al.* [30] designed a *4x4 XQ* switch with output queueing in FPGA for ATM Cable TV backbone networks. The crossbar and outputs were both buffered using the dual port SRAM. The *XQ* switch assumes RR arbitration for the crosspoint queueing and *Longest Queue First Served* (LQFS) scheme for the output queueing. The probability distribution of the occupancy of the buffers was measured, and the experimental results indicated that the *XQ* switch had a high performance under the ATM networks. Moreover, the performance measurements implied that the FPGA technology could provide sufficient logic and memory capacity for the *XQ* switch.

An IBM research group presented an ASIC implementation of a single-stage *CIXQ* switch [27]. The switch core was created to handle an aggregate throughput of *4 Tb/s* resulted from multiple OC-192 ($10Gb/s$) interfaces or OC-768 ($40Gb/s$) interfaces. RR arbitration is used at the inputs and the *Weighted Round-Robin* (WRR) algorithm is employed at the outputs. This combination allowed the switch to support up to 8 different *Classes of Service* (CoS). The chip was built in 0.11 micron CMOS process. The standard cell design methodologies, processes and packages were applied to keep the design time short. The testing results showed that the *CIXQ* switch can support cables of up to 100 feet between the line cards and the switch core, without performance degradation under any traffic patterns. This feature removed the effect of *Long Round-Trip Times* (RTT) when the switch was distributed over multiple shelves and racks. The proposed switch was proved to be valid in the CMOS technology under the requirements of a) a die size of up to $250mm^2$, b) up to 1000 signal inputs and outputs (totally 1500-pin package), and c) allowable maximum power consumption of $25W$.

In 2003, Yoshigoe *et al.* [31] developed a *CIXQ* switch in FPGA. *Round-Robin* (RR) arbitration was used for the input and output ports with a priority encoder based RR poller scheme. The target technology was the Xilinx Virtex II Pro Series. The switch was formed by six line cards, a crossbar and a bus motherboard. The crossbar consisted of four crossbar slices, each of which was connected to the line cards over the bus motherboard. The fully synchronized dual port memory on Virtex II Pro devices, named *BlockRAM*, was configured to implement V*OQ*s and crosspoint queues of the switch. The authors suggested that the off-the-shelf FPGA chips can provide sufficient resources for a *CIXQ* switch with up to 24 ports and $10Gb/s$ line speed. In addition, they expected that large *CIXQ* switch would become feasible in the future because the next generation of FPGAs presented larger amount of fast I/O circuits and memory resources.

M. Katevenis *et al.* [32] proposed another *CIXQ* switch in CMOS technology. The purpose of their work was to evaluate the performance of *CIXQ* switches operating directly on variable-size packets. A *32x32 CIXQ* switch with simple RR arbitration at the inputs and outputs was built in a 0.13 micron CMOS process. The input scheduler was realized at each ingress line card to minimize the scheduler latency. The design was synthesized using Synopsys [33] and placed and routed using Silicon Encounter [34]. After the layout and implementation cost of the circuit were analyzed, the authors concluded that a *32x32 CIXQ* switch with an aggregate capacity of $300Gbps$ was possible using the modern CMOS technology.

In short, the cost of producing *CIXQ* switches is more affordable as the CMOS technology is continuingly improving. Moreover, this author projects that a large buffered crossbar with hundreds of Gigabit or even Terabit line rate would be buildable in the future.

2.5 Advantages and Disadvantages

The *CIXQ* switches are configured with a FIFO queue in each crosspoint of the crossbar. Many studies have pointed out that *CIXQ* switches provide good performance in high-speed networks. According to [27, 28, 29, 30, 31, 32], the future CMOS technology should come with enough logic resources to build the buffered crossbar. These features suggest that *CIXQ* switches have great potential to be applied in future networks.

Considering the previous, traditional *IQ* switches, *CIXQ* switches have significant advantages as following:

1. The scheduling is simplified. FIGURE 2.5 depicts the transmission paths in the *IQ* and *CIXQ* switch. An *IQ* switch transmits data only if a connection between the input port and output port is established. The input and output matching depend on each other since each input can only be paired to a single output and vise versa. Instead, the *CIXQ* switch has two independent paths to move data through the switch: (1) data is forwarded from input port to crssspoint queue; and (2) data is transferred from corsspoint queues to output ports. Thus, no exchange of flow control information between the input and output scheduler is needed, therefore, simplifying the scheduling algorithm.

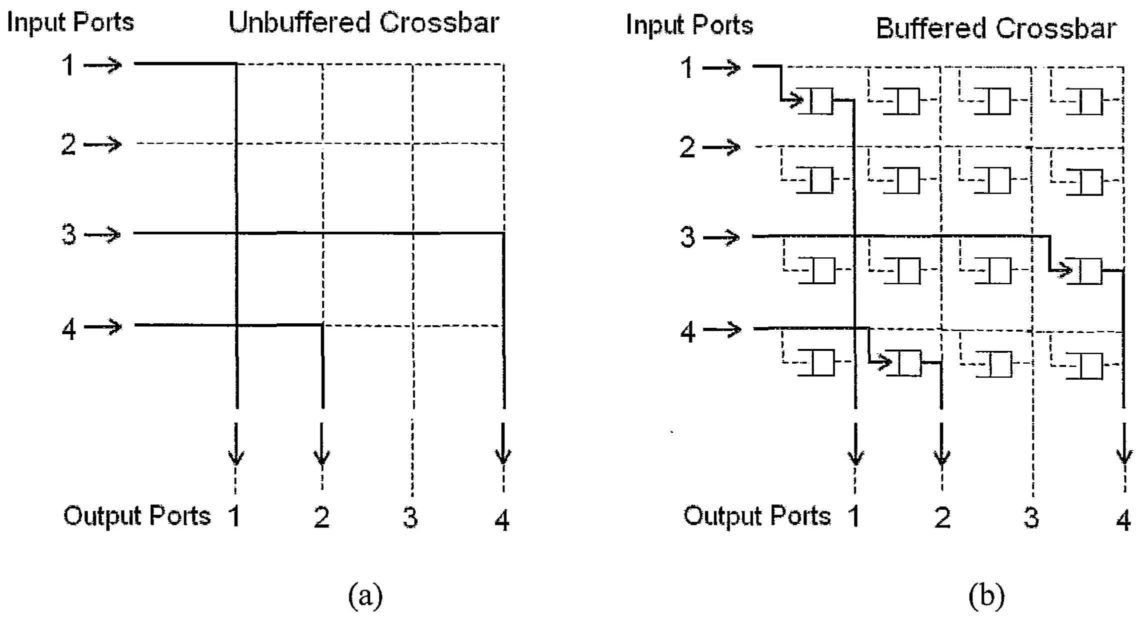


FIGURE 2.5: Transmission paths in (a) *IQ* switches and (b) *CIXQ* switches

2. Variable-size packets are supported [33, 35]. In an *IQ* switches, the variable-size packets must be first segmented into fixed-size cells at the input side, and then reassembled at the output side. But the *Segmentation and Reassembly* (SAR)

circuits increase the design difficulty and switching delay. The *CIXQ* switch can forward the variable-size packets without SAR processing at the expense of long queues required at the N^2 crosspoints.

For the disadvantages of having a FIFO in each crosspoint, they can be listed as following:

1. The design complexity is increased. *CIXQ* switches can simplify the scheduling at the expense of maintaining N^2 crosspoint queues in the crossbar. The memory required in the crosspoint queues is proportional to the square of the number of ports (O(N^2)). Therefore, it is expensive and difficult to build a large *CIXQ* switch.

2. The *CIXQ* switch must maintain a large crosspoint queue to support the variable-size packets. While *CIXQ* operates directly on the variable-sized packets, the crosspoint queue needs to be large enough to store the largest size packets on the networks.

2.6 Power Analysis

Power Consumption posses a major problem in VLSI design. This section provides a survey about the power estimation techniques and CAD tools for the CMOS VLSI designs as well as a review of power study for crossbar switches.

2.6.1 Power Consumption in CMOS circuits

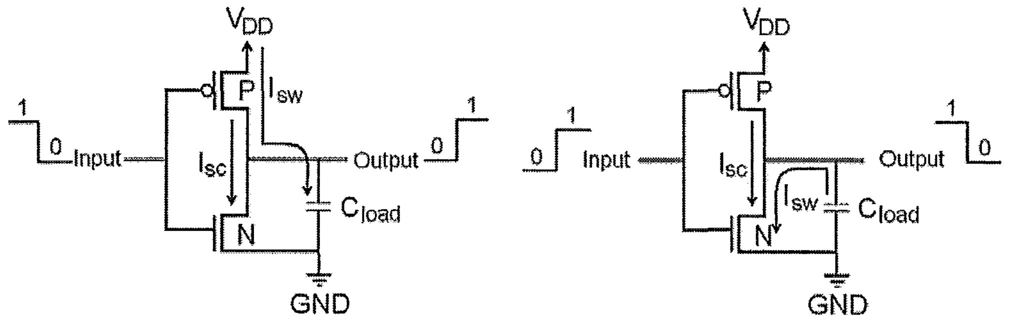
It is important to understand how CMOS circuits consume power when performing a power analysis. The total power dissipation in a CMOS circuit is generally comprised of three major components:

(1) Static power: the power due to leakage current when the circuit is idle. Thus, it is sometimes called leakage power. The amount of leakage current varies with die size, junction temperature, and process variations [36].

(2) Dynamic power: the power required for charging and discharging the load capacitance during signal transitions (when the circuit is switching). The main factors affecting dynamic power are load capacitance, supply voltage, and clock frequency [36].

(3) Short-circuit power: the power due to short-circuit current during switching of transistors.

The power dissipation in a CMOS circuit is illustrated in FIGURE 2.6 [37] through the example of a CMOS inverter during a signal transition. The load capacitor C_{load} represents the sum of parasitic capacitance of the nMOS and pMOS transistors, the capacitance associated with internal and external wires of the inverter cell, and the input capacitance of the circuits driven by the inverter. Assuming the circuit is initially in a steady state, when an input signal changes from high to low, i.e. a zero voltage is applied at the input, the P-type transistor turns on and the N-type transistor turns off. The load capacitor is charged up due to the switching current (I_{SW}) flowing from the P-type transistor to ground, as it is depicted in FIGURE 2.6 (a). During this charging phase, the energy drawn from power supply is $C_{load} \cdot V_{DD}^2$ and the output makes a transition from low to high.



(a) Charging phase (b) Discharging phase

FIGURE 2.6: Power dissipation in an inverter

As shown in FIGURE 2.6 (b), when an input changes from low to high, only the N-type transistor is active. The energy stored in the load capacitor is discharged through the N-type transistor. The discharging phase gives an output transition from high to low. For a particular time in the transition, both P- and N- type transistors are switched on simultaneously and the short circuit current (I_{SC}) can flow from power supply to ground, causing the dissipation of the short-circuit power.

Therefore, the dynamic power consumed by a CMOS inverter over a time interval [0, T] can be expressed by (from [37])

$$P_{dynamic} = C_{load} \cdot V_{DD}^2 \cdot N_{0\to 1} \cdot \frac{1}{T} \quad (2.1)$$

where $N_{0\rightarrow1}$ is the number of times that the output changes from 0 to 1, i.e., the number of times that C_{load} is charged over the period [0, T]. If the inverter operates at a clock frequency f_{clk}, Eq. 2.1 can be rewritten as (from [37])

$$P_{dynamic} = C_{load} \cdot V_{DD}^2 \cdot N_{trans} \cdot f_{clk} \quad (2.2)$$

where N_{trans} is the average number of transitions per clock cycle at the output, referred to as switching activity.

In summary, the power dissipation in CMOS circuits comes from three major parts: dynamic power caused by the switching activity, static power due to the leakage currents and short-circuit power due to the short-circuit current. The dynamic power is the dominant component. For CMOS technologies up to $0.35\mu m$, the dynamic power is about 80% of the total power [37]. As the technology scales further down, the contribution of dynamic power increases because the functionality requirements and clock frequencies are increased [38]. As a result, the majority of existing power estimation techniques focuses on the dynamic power dissipation in CMOS circuits.

2.6.2 Power Analysis Techniques

Power analysis is defined as the problem of evaluating the average power dissipation in a digital circuit [39]. Power analysis tools are required to help with the power management tasks. More importantly, they can help the designers to determine how well a design meets the power specifications in early stages of the design process. In general, power analysis techniques fall into two main categories based on the levels of design abstraction: *low-level* and *high-level techniques*, as it is summarized in FIGURE 2.7.

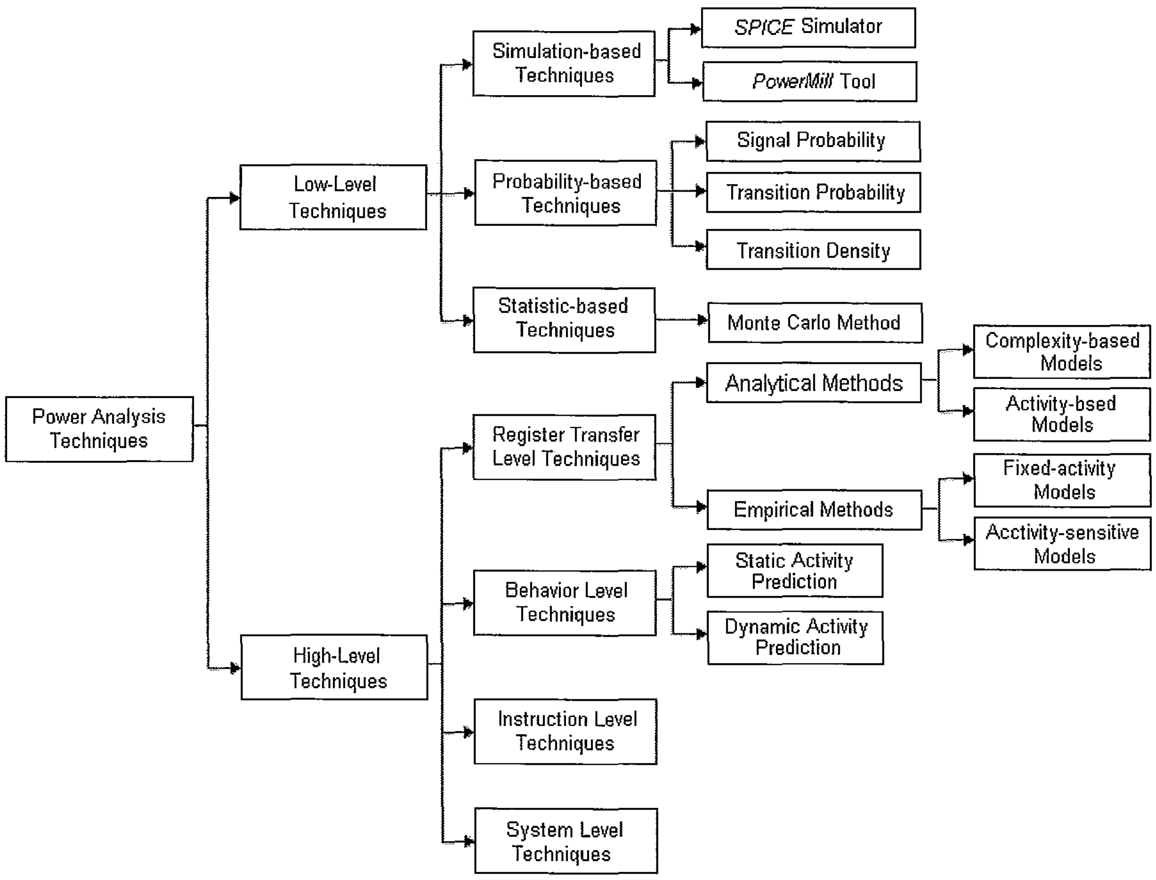

Power Analysis Techniques
Low-Level Techniques
High-Level Techniques
Simulation-based Techniques
Probability-based Techniques
Statistic-based Techniques
Register Transfer Level Techniques
Behavior Level Techniques
Instruction Level Techniques
System Level Techniques
SPICE Simulator
PowerMill Tool
Signal Probability
Transition Probability
Transition Density
Monte Carlo Method
Analytical Methods
Empirical Methods
Static Activity Prediction
Dynamic Activity Prediction
Complexity-based Models
Activity-bsed Models
Fixed-activity Models
Acctivity-sensitive Models

FIGURE 2.7: An overview of power analysis techniques

2.6.2.1 Low-Level Techniques

The low-level techniques estimate the power from an analysis of switching activities of a design. They are also called gate-level techniques because the design is typically at the gate level. Refer to FIGURE 2.7, the low-level techniques can be categorized into three groups:

1. Simulation-based techniques: these approaches estimate the power through a computer simulation. The circuit is first simulated with a large number of input signals. During the simulations, all possible switching activities inside the circuit are captured. The power is then measured based on the simulation results. Even though the simulation based techniques are simple and straight-forward, the accuracy of the estimated power directly depends on the specific input signals used to drive the simulator. So they are considered strongly pattern-dependents. *SPICE* and *PowerMill* are two examples of power estimation tools using circuit simulation.
2. Probability-based techniques: these approaches analyze the power using transition probability to overcome the pattern-dependent problem exists in the simulation based techniques. The transition probability describes all possible logic signal transitions in the design. Instead of simulating the whole circuit with specific input signals in the simulation-based techniques, the probability-based approaches first compute the probability that a signal transition occurs at internal nodes. Then they propagate the probabilistic information into specialized circuit models to evaluate the power drawn by the circuit. The probability-based approaches determine the signal and transition probabilities of the internal nodes of the circuit by calculating (1) the signal probability that logic signals at specific nodes are logic high, or (2) the average transition probability from the probability waveforms, which describes the probability that logic signals make low-to-high transition at specific time points [39], or (3) the transition density, which is

defined as the average number of transitions per second at a node in the circuit [40]. (as it is depicted in FIGURE 2.7)

3. Statistic-based techniques: these approaches analyze the power with use of statistical estimation strategies. The statistic-based techniques employ traditional simulation models to test the circuit for a limited number of randomly generated input signals. The power consumption is monitored during the repeated simulations, and the process is only stopped if the measured power satisfies a specified error bound. For example, R. Burch *et al.* [41] adopt a Monte Carlo method to compute the power of ASIC designs.

The low-level techniques generally provide high estimation accuracy at the cost of long computation time. Furthermore, they require a complete circuit design at the gate level. In order to avoid costly redesign process, other techniques are needed to analyze the power of a design at higher levels of abstraction than the gate level.

2.6.2.2 High-Level Techniques

A number of high-level techniques have been proposed to estimate the power at high levels of abstraction design. As shown in FIGURE 2.7, the high-level techniques can be divided into four categories: register-transfer level (RTL) techniques, behaviour level techniques, instruction level techniques and system level techniques.

1. Register-transfer level (RTL) techniques: these approaches calculate the power at the architecture, or register-transfer level. At this stage, the primitives are

functional blocks such as adders, multipliers, controller, and SRAM. The RTL techniques can be subdivided into two classes: (1) analytical methods and (2) empirical methods, and they are descried as follows.

(1) The analytical methods evaluate the power of a design in RTL description by indentifying the factors that affect the power dissipation in the circuits, such as the physical capacitance and activity information. These factors can be checked based on two different schemes: the complexity-based schemes and activity-based schemes. The complexity-based schemes compute the total power by summing up the power consumption of basic logic components required in the design. An example of these approaches is the Chip Estimation System (CES) discussed in [42]. The activity-based schemes first measure the design's average activity using the concept of entropy from information theory. The activity information is then applied to evaluate the power. These approaches are described in [43, 44]. The analytical methods which include the complexity-based schemes and activity-based schemes require very little information for the power calculation, such as technology parameters, memory sizes, and the number of logic components. However, they normally results in a low accuracy.

(2) The empirical methods are other RTL estimation techniques that employ a macro-modeling scheme to obtain higher accuracy than the analytical methods. These methods produce an analytical model from the power measurement of the basic building blocks. The empirical methods can further

be subdivided into fixed-activity schemes, which ignore the power caused by the data activities, and activity-sensitive schemes, which account for the power of data activity, see FIGURE 2.7. The Power Factor Approximation (PFA) tool [45] is one example of fixed-activity schemes. The power estimation tool called ESP [46] and SPA [47] are examples of activity-sensitive schemes.

2. Behaviour-level techniques: these approaches predict the signal activity of the design, such as the access frequencies for different hardware resource, and use that information to analyze the power. Two methods are suggested to predict signal activity: (1) static activity prediction and (2) dynamic activity prediction, as it is described in FIGURE 2.7. The static activity prediction method determines the access frequencies from an analysis of the desired behaviour of the design. For example, the power model proposed by R. Mehra [48] was based on the study of a control-data flow graph (CDFG) for a high-level synthesis system. In contrast, dynamic activity prediction method analyzes the access frequencies by gathering the activity statistics from a design simulation. One example of this method is the Profile-Driven Synthesis System (PDSS) [49].
3. Instruction-level techniques: these approaches compute the power by realizing a given behaviour in software on a programmable instruction set processor. The model for embedded general-purpose and DSP process [50] is one example of these techniques.

4. System-level techniques: these approaches get a rough power accounting for all the components in a system at the earliest stage of design specification, for instance, the web power estimation tool *PowerPlay* [51].

In conclusion, the power analysis techniques are divided into two categories for the VLSI designs: the low-level techniques, which estimate the power of a digital circuit in the gate-level abstraction, and the high-level techniques, which compute the power in a level higher than gate-level, as it is illustrated in FIGURE 2.7. Compared to the low-level techniques, the high-level methods obtain a lower accuracy, but they require a shorter operation time for the power estimation. In the literature, a lot of power estimation models and tools for the crossbar switches are derived from one of the techniques discussed in this section. The previous work which is proposed to estimate the power dissipation in crossbar switches is presented in Section 2.6.4.

2.6.3 Commercial Power Analysis Tools

With the rapid growth of power-sensitive and high-density programmable devices, such as the portable electronics, power dissipation has become an important factor. Power analysis tools are essential for designers to check the power consumption of circuits during the design process. Today, semiconductor vendors have two different approaches to investigate the power in the stages from early concept to late design implementation: spreadsheets [52, 53, 54, 55], and CAD tools [56, 57, 58].

2.6.3.1 The Spreadsheets

The designers need to provide the activity information for the power estimation using the spreadsheets. These spreadsheets analyze the power in three parts: static power, dynamic power and the maximum allowed power. The static power of each component is given in the device family datasheet. Designers can determine the total static power based on the number of specific components used in the design. To calculate the dynamic power, the average switching frequencies for the logic blocks or each module are defined by the designers, while the other coefficients or parameters are usually available in the device datasheet. The total power is measured as a summation of the static and dynamic power. Furthermore, the maximum allowed power in the device is computed to prevent any violation of power budget. If the estimated power is less than the maximum allowed power, the design is considered valid. Otherwise, the designers have to go back to solve the high power problem.

The power estimation flow used in spreadsheets can be summarized in following steps [53, 59]:

1) To identify the components used in the design and determine the total static power from the static power of specific components. The static power is described as

$$P_{static} = \sum_{all\ \text{components}} P_{static\_per\_component} \quad (2.3)$$

where $P_{static\_per\_component}$ is the static power dissipated by each component in the design, and it is given in the device datasheet.

2) To find out the switching activity of the logic blocks or each module in the design, and the dynamic power can be calculated by following equation

$$P_{dynamic} = \sum_{all\ \text{modules}} Kf_i(\max)S_iC_iV_{swing}V_{\text{supply}} \quad (2.4)$$

where K is the power coefficient provided in the datasheet; $f_i(\max)$ and S_i are the maximum frequency and average switching activity; C_i reflects the capacitance related to the particular logic block or module; V_{swing} and V_{supply} represent the swing voltage and supply voltage.

3) The total estimate power can be expressed as:

$$P_{total} = P_{static} + P_{dynamic} \quad (2.5)$$

4) The maximum allowed power (P_{MAX}) for a device is determined by

$$P_{MAX} = \frac{T_J - T_A}{\theta_{JA}} = \frac{T_J - T_C}{\theta_{JC}} \quad (2.6)$$

where T_J is the maximum allowed junction temperature of the device; T_A and T_C represent the ambient temperature and the case temperature of device respectively; θ_{JA} and θ_{JC} stand for the thermal resistance and the junction-to-case thermal resistance. The values of T_J, θ_{JA} and θ_{JC} are both specified in the device family datasheet, while designers give the values of T_A and T_C.

5) To ensure $P_{total} < P_{MAX}$. If $P_{total} > P_{MAX}$, the design is considered invalid, and a redesign process is required.

Spreadsheets can be used in the early design phases, such as the board design and layout phase, to develop proper power management. They are simple to use, but results normally have low estimation accuracy.

2.6.3.2 The CAD Tools

Compared to spreadsheets, CAD tools capture the switching activities of the designs based on the simulations instead of the switching information defined from the designers. Therefore, the estimated results from the CAD tools should have a higher accuracy. However, the CAD tools typically require a complete circuit design before a power analysis can be performed. Xilinx's *Xpower* [58] and Altera's *PowerPlay* [57] tools are two well known commercial tools for power estimation.

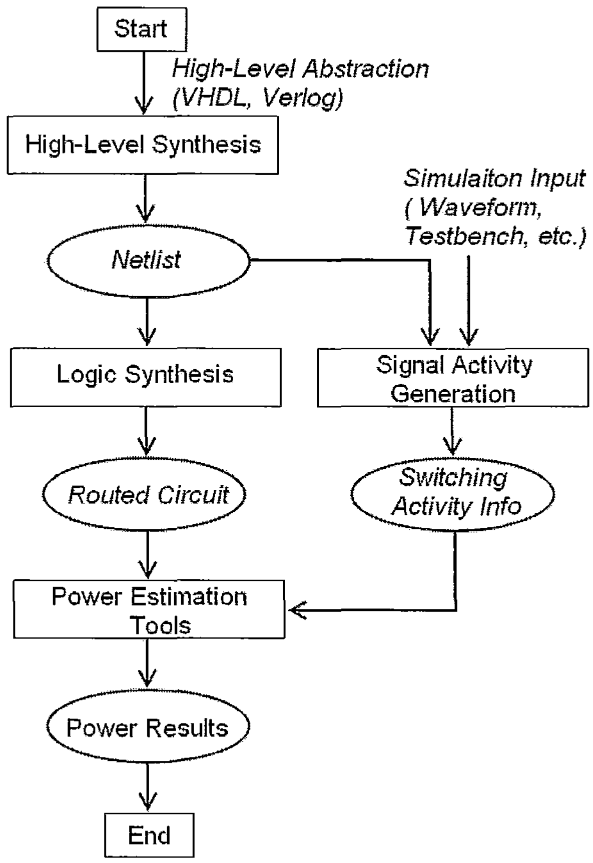


FIGURE 2.8: Power estimation flow in CAD tools

The approach for the CAD tools to estimate the power consumption is illustrated in FIGURE 2.8 (from [60]). The complete design specified in a high-level description language, like VHDL or Verilog, is first synthesized to a *netlist* during synthesis. The designers run circuit simulations with test input signals represented as waveforms or testbenches. In the simulations, the signal activity generation software collects the switching activity information inside the circuit. Finally, the estimator concludes the power from the routing information in *Logic Synthesis* and the switching activity information in simulations.

In short, there are two popular design tools in the industry to estimate the power consumption of a VLSI design: the spreadsheets and the CAD tools. FIGURE 2.9 compares these tools regarding the design process and the accuracy of estimation. A power measurement in later stage of the design process causes a higher accuracy. Therefore, depending on which stage the design is in and the required accuracy, the designers can use either the spreadsheets or the CAD tools.

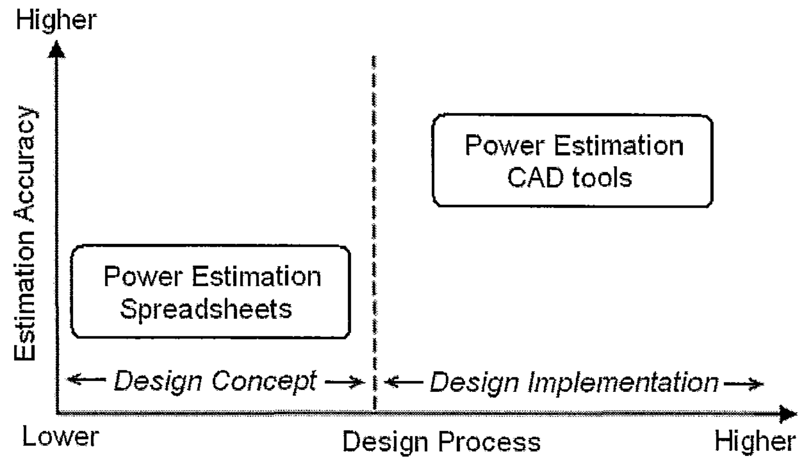

Higher
Estimation Accuracy
Power Estimation CAD tools
Power Estimation Spreadsheets
← Design Concept →
← Design Implementation →
Lower
Design Process
Higher

FIGURE 2.9: Power estimation tools

2.6.4 Power Estimation of Crossbar Switches

Many techniques have been suggested to analyze the power for general CMOS VLSI designs. The analysis can be performed at various abstraction levels as described in Section 2.6.2. So far, only a handful of studies have been done to analyze the power dissipation in crossbar switches.

In early 1997, a simple analytic model [61] was introduced to evaluate the power dissipation of multiprocessor interconnection networks under a fixed power constraint. This simple model could not perform power analysis of other interconnect designs at higher abstraction levels. Moreover, it is not sufficiently detailed because the analysis is just based on a counting of transistors required in the designs, but ignoring the wires. Recently, many models have been created to address the power problem of crossbar interconnects at various levels. H. Zhang *et al.* [62] derived an analytical model for the energy consumption of a crossbar interconnect network on the transistor level. Their study pointed out that a hierarchical interconnect network had the best energy efficiency.

With the development of SoC technology, a large number of device components like microprocessors, memory, and interfaces can be integrated into a single microchip. Hence, the power consumption of on-chip interconnects becomes an important design issue. A gate-level model was proposed in [63] for the power of the crossbar switches in standard cell ASIC processes. With a double metal 0.6 micron CMOS technology, the experimental results showed the power model had an average error less than 10%. A. G. Wassal *et al.* [64] proposed a system-level approach to compute the power consumption of VLSI packet switching fabric. By taking the traffic patterns into account, a traffic model was introduced to generate the switching activity inside the circuit. This approach delivers a high accuracy compared to other methods that did not include traffic models and statistics. The power estimated from this approach had an error of about 14%. Another power model for crossbar interconnects was also presented in [65] at architectural level. The model was shown to be quite accurate with an average error of 5.8%, but it is not suitable for crossbar switches bigger than *16x16*.

There were several power models proposed in the past for network routers [66, 67] and switches [68, 69]. In [66], the authors introduced a framework at the architecture level to estimate the power of four widely-used switch fabric architectures in network routers: crossbar, fully connected, Banyan and Batcher-Banyan. This framework improved the accuracy of estimation to bit level by introducing *"Bit Energy"*, which describes the energy consumed by a single bit travelling through the switch fabrics. Their results showed that a *32x32* bufferless crossbar switch fabric consumed approximately 300 mW, under uniform traffic conditions and load almost 60%. Another architectural-level power model were developed in [67] for the network routers. The authors broke down a router into three building blocks: the memories, crossbars and arbiters, and proposed a power model for each of them. The power models were verified by two commercial routers, the *Alpha 21364* router and the *IBM InfiniBand 8-port 12X* router. The experimental results shows the estimation results were very close to the values estimated by designers of these two products with an error of less than 15%.

A detailed analysis of multiplexer-based crossbar switches is presented in [68, 69]. T. Wu *et al.* [68] estimated the power consumption of a *256x256* crossbar switch implemented using TSMC 0.25 micron CMOS technology with a pipelined MUX. The design was shown to require about 2.5 W at a clock rate of 1 GHz. In [69], the power dissipated by both unpipelined and pipelined optoelectronic crossbar switches using 0.18 micron CMOS technology was analyzed. The optoelectronic switches are constructed from optical I/O and electrical crossbar switching elements. The authors not only modelled the power of basic components of switches, but also proposed an approach to minimize the power dissipation. According to their power models, the designers can estimate the switch power based on the complexity of design architectures, allowing for an exploration of architectural tradeoffs. Furthermore, their models can be used in any size of CMOS technologies from different standard cell library vendors like American Semiconductor and TSMC. In this thesis, we follow the same methodology as described in [69] to model the power dissipation in *CIXQ* switches (see Chapter 5).

The previous approaches do not directly focus on the power consumption of *CIXQ* switches. To our best knowledge, the first work to measure the power consumption of *CIXQ* switches was presented by D. G. Simos [70]. The purpose of this work was to evaluate the design feasibility of a *CIXQ* switch in 0.13 micron and 0.09 micron CMOS technologies. A *32x32 CIXQ* switch was implemented, and the power and area required by such design were observed. The analysis results indicated that the power and area required by the switch is about 3.2 *W* and 200 mm^2 in 0.13 micron CMOS with aggregate incoming throughput of300 Gbps . As a conclusion, a *32x32 CIXQ* switch with 300 Gbps aggregated line rate is feasible since a medium-priced packaging technology can meet the area requirement and power dissipation will not cause any problem in current CMOS technologies. Also, the power breakdown showed that about 62% of the power is consumed by data wires driving the inputs and outputs, while memories and other components are minority consumers. The discussions in this paper are only valid for a *CIXQ* switch with size *32x32* and no general power model was provided. Therefore, our work is considered the first approach to model the power dissipation for *CIXQ* switches.

2.7 Summary

CIXQ switches have gained considerable research attention in the past decade. Thus far, researchers are interested in three areas of *CIXQ* switches development: the switching performance, implementation and power consumption. FIGURE 2.10 shows the distribution of previous work in these areas. Most attention has been paid to the performance of *CIXQ* switches under uniform and non-uniform traffic. Numerous theoretical and experimental studies establish an agreement that *CIXQ* switches can achieve high performance with a less complex scheduling than *IQ* switches. However, the implementation is only feasible when the CMOS technology allows the integration of a large amount of memory inside a single chip. Since then, researchers have seriously considered the development and deployment of *CIXQ* switches in high-speed networks. Most recently, the power dissipation of *CIXQ* switches have become a critical concern as the CMOS feature size is scaled down to the nanometer. To date, only a limited amount of work has been done to address the power problem, and no power model has been presented for *CIXQ* switches.

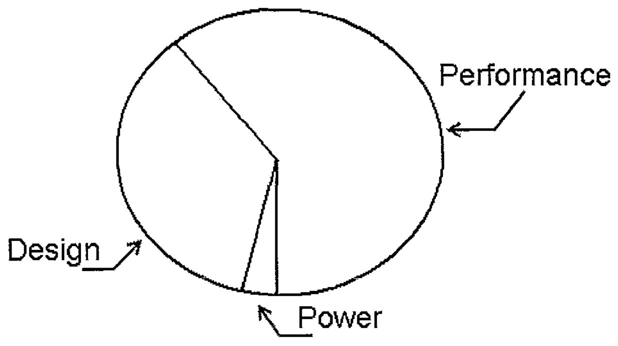


FIGURE 2.10: The research distribution about *CIXQ* switches

According to the research focus of *CIXQ* switches discussed in this chapter, the following chapters will first investigate their switching performance using MATLAB, then present a VHDL implementation of a *CIXQ* switch in FPGA, and finally develop power models to study the power characteristics of the switches.

Chapter 3 Performance Evaluation

3.1 Introduction

CIXQ switches have been considered as an alternative to traditional *IQ* switches to simplify the scheduling. As the major advantage, *CIXQ* switches can simplify the scheduling because the input and output arbitration can be performed independently and in parallel. Many algorithms have been proposed for scheduling the packets at the inputs and outputs of *CIXQ* switches in the last decade, such as *Round-Robin* (RR) [20], *Dual Round-Robin* (DRR) [23], *Longest Queued First* (LQF) [24], and *Oldest Cell First* (OCF) [24]. The studies in [18, 19, 20, 21, 22, 23, 24] have established that these algorithms can obtain high throughput with reasonable delay under uniform and non-uniform traffic.

In this chapter, we evaluate the performance of *CIXQ* switches under uniform and non-uniform traffic with Bernoulli arrivals. In order to observe the performance, a *32x32* *CIXQ* switch is simulated using MATLAB. An *IQ* switch is also implemented for comparison purpose. Both switches do not use an internal speedup. RR arbitration with single iteration is used at the inputs and outputs of the *CIXQ* switch while RR arbitration with four iterations, denoted as *iSLIP* [7], is employed at the inputs of the *IQ* switch. Although a lot of existing algorithms can achieve better performance than RR scheme and iSLIP for packet switching systems, the purpose of this chapter is to show that the *CIXQ* switch can obtain high performance using a simple scheduling algorithm, which is described as the major advantage of *CIXQ* switch in Section 2.5. Therefore, it is sufficient to have a performance comparison between the simple RR scheme and a more complicated version of RR, iSLIP, to meet this purpose.

The rest of chapter is organized as follows. We first describe the switch model for the *CIXQ* switch. Next, we introduce the scheduling algorithm of RR arbitration and iSLIP. The simulation methodology and traffic models are also discussed. Then we present the simulation results and switch size effect. Finally, a conclusion is draw for the performance properties of the *CIXQ* switch.

3.2 Switch Model

In this section, we describe the *CIXQ-k* switch model, where *k* is the crosspoint queue size. The variable-size packets are usually segmented into small fixed size cells at the input side of the switch, and then are re-assembled at the output sides. The transmissions of cells occur between input ports and output ports. The transmission time is defined as one cell-time or one time slot.

An *NxN CIXQ-k* switchwith RR arbitration at inputs and outputs and without speedup is described in FIGURE 3.1, and it is denoted as a RR/RR *CIXQ-k* switch. A RR/RR *CIXQ-k* switch has the following major parts:

- *Input Queue (IQ) with VOQ.* There are *N* VOQs at each input port. A VOQ at input port *i* that stores cells for output port *j* is denoted as *VOQ(i,j)*.
- *Crosspoint Queue (XQ).* There is a FIFO queue in each crosspoint, i.e. N^2 crosspoint queues locate inside the crossbar. A XQ in the crosspoint between input *i* and output *j* is represented as *XQ(i,j)*.

- *Flow Control Mechanism.* A credit-based flow-control mechanism is used to avoid the overflow of the XQ. Each VOQ contains a credit counter to indicate whether the corresponding XQ has a space for next incoming cell. If the credit counter is less than the maximum size of the XQ, the VOQ is considered eligible for input arbitration.
- *Input and output arbitration.* A simple round-robin arbitration is employed separately at the input and output ports. In round-robin fashion, the input arbiter selects an eligible VOQ to move cells from inputs to XQs while the output arbiter transmits cells from the last serviced XQ to the output port.

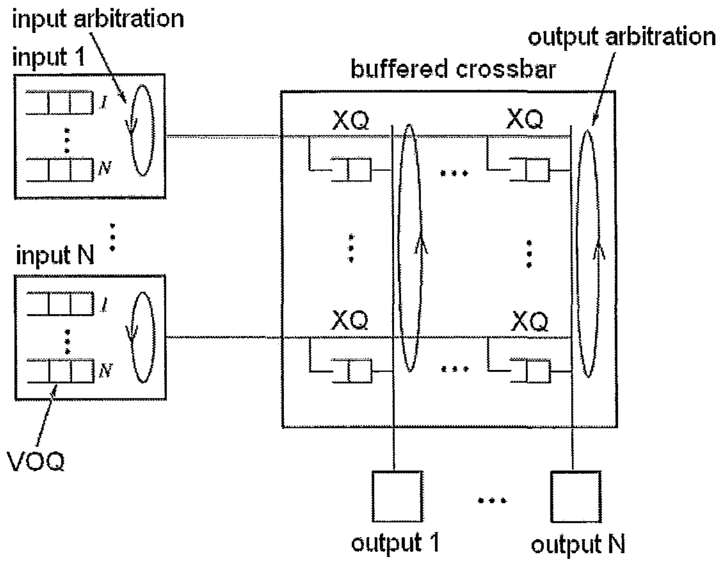


FIGURE 3.1: A RR/RR *CIXQ-k* switch

In summary, when a cell arrives at a *CIXQ* switch, it is stored in the appropriate *VOQ(i,j)* at the input side. Once *VOQ(i,j)* becomes eligible for input arbitration, it waits for a grant to transmit the cell to *XQ(i,j)*. At the output side, the arbiter moves the cell from *XQ(i,j)* to its final destination, output *j*. The input and output arbitration follow a round-robin scheme.

3.3 Scheduling Algorithm: RR vs. iSLIP

In this section, we describe and compare the Round Robin arbitration and iSLIP scheduler in details.

3.3.1 Round-Robin

Round-robin scheme is one of the simplest scheduling algorithms, which can provide reasonably fair service for equal weight traffic. Compared to *Parallel Iterative Matching* (PIM) [9], RR scheduling also has three steps for the arbitration process: *Request, Grant and Accept* (RGA), as shown below. However, it applies a round robin scheme instead of a random policy in the steps of Grant and Accept. A RGA with RR scheduling for an *IQ* switch at the inputs and outputs is described as follows:

Step 1: Request. Each unmatched input port sends a request to every output port for which it has one or more queued cells.

Step 2: Grant. If unmatched output receives one or more than one requests, it selects one and only one request based on the round robin order. Therefore, a request that appears next from the current position of the round robin pointer at the output arbiter is granted. The output notifies each input whether or not its request is granted. Then the pointer is increased to the location beyond the granted input (modulo *N*, where *N* is the number of ports in the switch).

Step 3: Accept. If an unmatched input receives more than one grant, it chooses one and only one grant based on the round robin order. So, the input accepts the grant that appears next from the current positions of round robin pointer at input arbiter. Then the pointer is increased to the location beyond the accepted output (modulo *N*).

As the major advantage of *CIXQ* switches, the input and output scheduling are independent of each other. When RR scheduling is employed at the inputs and outputs of a *CIXQ* switch, there is no flow control information exchange between the schedulers. Therefore, the input and output arbitration is reduced to two steps without *Accept* process when used in a *CIXQ* switch. For instance, the input scheduling can be described as:

Step 1: Request. Each VOQ at the input port sends a request to a crosspoint queue for which it has one or more queued cells.

Step 2: Grant. If the crosspoint queue receives one or more requests, it grants a request that is in the next location of the round robin pointer. Then the pointer is incremented to one location beyond the granted crosspoint queue (modulo *N*). The crosspoint queue also notifies each VOQ whether or not its request is granted. Finally, the granted VOQ moves the data to the crosspoint queue.

3.3.2 iSLIP

iSLIP proposed by McKeown [7] is a variation of the RGA scheduler with round robin arbitration, where the *Grant* process is modified as:

Step 2: Grant. If an unmatched output receives more than one request, it selects at most one request that appears next in a round robin schedule. The output notifies each input whether or not its request is granted. Then the pointer is increased to the location beyond the granted input (modulo *N*) *if and only if* the grant is accepted in Step 3.

Other than that, iSLIP follows the same procedure in Step 1 and Step 3 of the RGA with RR scheduling. This small change leads to a large improvement in performance by removing the synchronization problem (i.e. contention) at the output arbiter for random uniform traffic [7]. The performance is further improved as the number of iterations increase up to about $\log_2 N$ for an *NxN* switch. For convenience, an IQ switch with n iterations of iSLIP is represented as IQ-iSLIP-n in this thesis. When multiple iterations are applied, iSLIP can be described as follows:

Step 1: Request. Each unmatched input port sends a request to every output for which it has one or more queued cells being destined to.

Step 2: Grant. If unmatched output receives one or more requests, it selects only one request that appears next in a RR schedule. The output notifies each input whether or not its request is granted. Then the pointer is increased to the location beyond the granted input (modulo *N*) *if and only if* the grant is accepted in Step 3 of the first iteration.

Step 3: Accept. If an unmatched input receives one or more grants, it chooses at most one grant that appears next from the current positions of RR pointer at input arbiter. Then the pointer is increased to the location beyond the accepted output (modulo *N*).

3.3.3 Summary

iSLIP is an iterative round-robin scheme based on the three-phase RGA process. More iterations lead to a larger improvement in performance. The basic RR scheduling can be simplified into two-phase RG process when used in *CIXQ* switches. Furthermore, no iterations are required in the arbitration process. Therefore, the arbitration time of RR scheduling in *CIXQ* switches is much faster than iSLIP with multiple iterations in *IQ* switches.

3.4 Traffic Model

Similar to [21, 22], two common traffic patterns are considered for the performance evaluation of *IQ* and *CIXQ* switches: the uniform traffic and unbalanced traffic. In addition, a more difficult traffic pattern, the log-diagonal traffic [11], is also discussed. We do not address the issue of correlated traffic in this thesis since our primary motivation is to develop an analytic power model, and the topic of correlated traffic associated schedulers is beyond the scope of this thesis.

3.4.1 Bernoulli Arrivals

We assume that the arrival cells of the switch have a Bernoulli distribution with probability of p per time slot. The Bernoulli distribution is a discrete distribution having two possible outcomes labelled by $x = 0$ and $x = 1$. In these events, x = *1* ("success") occurs with probability p, and $x = 0$ ("failure") occurs with probability 1-p, where $0 \le p \le 1$. Therefore, it has the probability function as below [71]

$$f(x) = \begin{cases} p & \textit{if } x = 1 \\ 1 - p & \textit{if } x = 0 \\ 0 & \textit{otherwise} \end{cases} \qquad (3.3)$$

3.4.2 Uniform Traffic

Uniform traffic is the simplest pattern to simulate while evaluating the performance of switches. Let p_{ij} be the probability of a cell at input i destined for output j, where $1 \le i \le N$ and $1 \le j \le N$, and it is evenly distributed among all the output ports (N) for a uniform traffic [72]:

$$p_{ij} = p / N \quad \forall i, j \quad (3.4)$$

where p is the total traffic load at any input and output port of the switch, which is normally assumed to be not more than one ($p \le 1$) and N is the number of ports of switch.

3.4.3 Unbalanced Traffic

For unbalanced traffic, the traffic load for each input and output pair is determined by the unbalanced probability *u*, where *0 <= u <= 1,* and it can be expressed as [21],

$$p_{ij} = \begin{cases} p(u + \frac{1-u}{N}) & \textit{if } \text{i=j} \\ p\frac{1-u}{N} & \text{otherwise} \end{cases} \quad (3.5)$$

When *u* is zero, it is the uniform traffic since the traffic load is distributed to all traffic flows evenly, i.e. $p_{ij} = p / N$. When *u* is one, the traffic is completely unbalanced because the traffic load is only assigned to the connections between input *i* and output *j*, where *i* equals j.

3.4.4 Log-diagonal Traffic

Log-diagonal traffic is more difficult to schedule than uniform and unbalanced traffic. The traffic load from input *i* to output *j* obeys the condition [11]:

$$p_{ij} = 2p_{i|j+1|}, \ where \ |j+1| = (j+1) \bmod N \quad (3.6)$$

For the total traffic load at each input and output port of the switch, it has the property that $\sum_{i=1}^{N} p_{ij} = p$ and $\sum_{j=1}^{N} p_{ij} = p$.

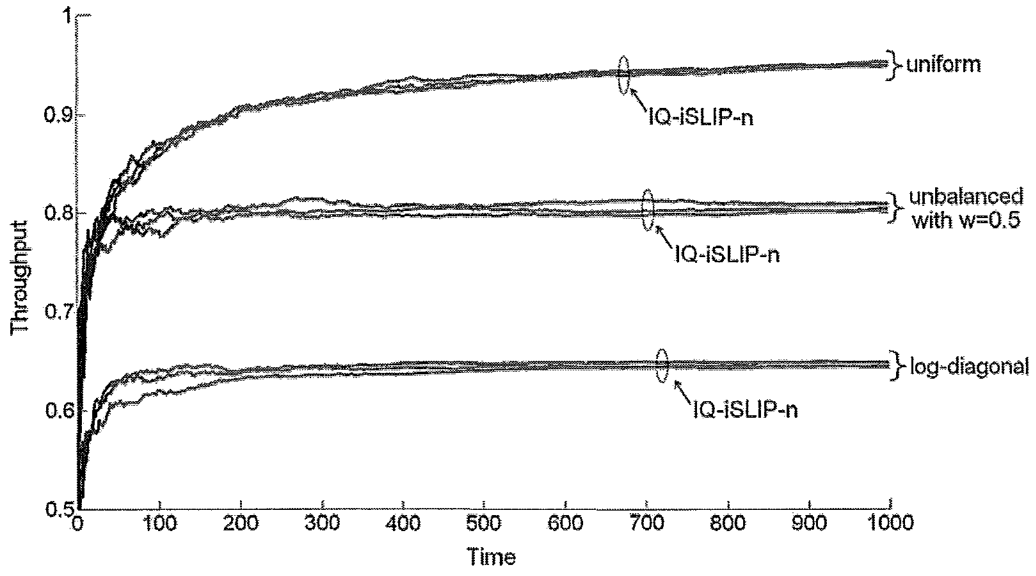


FIGURE 3.2: Transient throughput performance of *IQ* switch with n iterations of iSLIP under various traffic patterns, where n = 4, 8, and 16

3.5 Simulation Methodology

A *32x32* RR/RR *CIXQ*-k switch and *IQ* switch with 4 iterations of iSLIP without speedup are implemented in MATLAB. We choose four iterations of iSLIP since more iterations do not yield much improvement according to our simulations, as it is depicted in FIGURE 3.2.

As described in Section 3.2, the cells are transmitted throughput the *CIXQ* switch in two independent steps: cells at the input side are moved from VOQs to XQs; and cells are transmitted from XQs to output ports at the output side. The *CIXQ* switch can be simply designed to perform these two steps for each time slot. The MATLAB code for each transmission step is described in FIGURE 3.3 and FIGURE 3.4. The discrete-time is used as the simulation time.

```
VOQ_XQ = zeros(N,N); %arrival cells at XQ(i,j) from VOQ(i,j)
for ip = 1:N
    xq = LSVOQ(ip); %recall the last service VOQ
    for jj = 1: N
       % we found an eligible VOQ
       if (VOQ_COUNTER(ip, xq) < k) %k = crosspoint buffer size
            % an arrival cell
            if (VOQ(ip,xq) > 0)
                %move it to the XQ
                VOQ_XQ(ip,xq) = 1;
                %update the last serviced VOQ
                LSVOQ(ip) = xq + 1;
                %update the VOQ counter
                VOQ_COUNTER(ip, xq) = VOQ_COUNTER(ip, xq) + 1;
                if(LSVOQ(ip) > N)
                    LSVOQ(ip) = 1; %wrap around
                end;
                break;
            end;
        end;
        %check next XQ if no cell arrives at current one
        xq = xq + 1;
        if (xq > N)
          xq = 1; %wrap around
        end;
    end;
end;
  % update the VOQ once the cells are removed to XQ
  VOQ = VOQ - VOQ_XQ;
```

FIGURE 3.3: MATLAB Code for moving cells from input to *XQ*

```
XQ_OP = zeros(N,N); %arrival cells at OP(j) from XQ(i,j)
    for op = 1:N
        %recored the last serviced crosspoint buffer
        xq = LSXQ(op);
        for jj = 1:N
            %select nonempty crosspoint buffer
            if (VOQ_COUNTER(xq, op) > 0)
                % forward the cell to OP
                XQ_OP(xq, op) = 1; %permutation matrix
                %update the VOQ counter
                VOQ_COUNTER(xq, op) = VOQ_COUNTER(xq, op) - 1;
                %record the last service OP
                LSXQ(op) = xq + 1;
                if(LSXQ(op) > N)
                    LSXQ(op) = 1; %wrap around
                end;
                break;
            end;
            %check next XQ if no cell arrives at current one
            xq = xq + 1;
            if(xq > N)
                xq = 1; %wrap around
            end;
        end;
    end;
```

FIGURE 3.4: MATLAB Code for moving cells from *XQ* to output

3.6 Simulation Results

It has been shown that if the crosspoint queues are large enough that every traffic flow receives fair service [15]. The performance of CIXQ switch is observed in terms of throughput and delay. Throughput is usually defined as the probability that an input or output port is busy per time slot. In our measurements, the throughput is approximated as the ratio between the number of cells arriving at the switch and the number of cells departing from the switch in a time slot. In a switching system, a 100% throughput means the number of cells entering the switch is equal to the one leaving the switch. Delay is defined as the time that an arriving cell incurs to arrive at the output port.

The throughput and delay of the switches are measured through the simulations. This section provides the simulation results for RR/RR *CIXQ*-k switch and *IQ* switch with four iterations of iSLIP (short for *IQ*-iSLIP-4) under different traffic patterns. For each experiment, a method of long-run is used to detect the steady state of the switches [73].

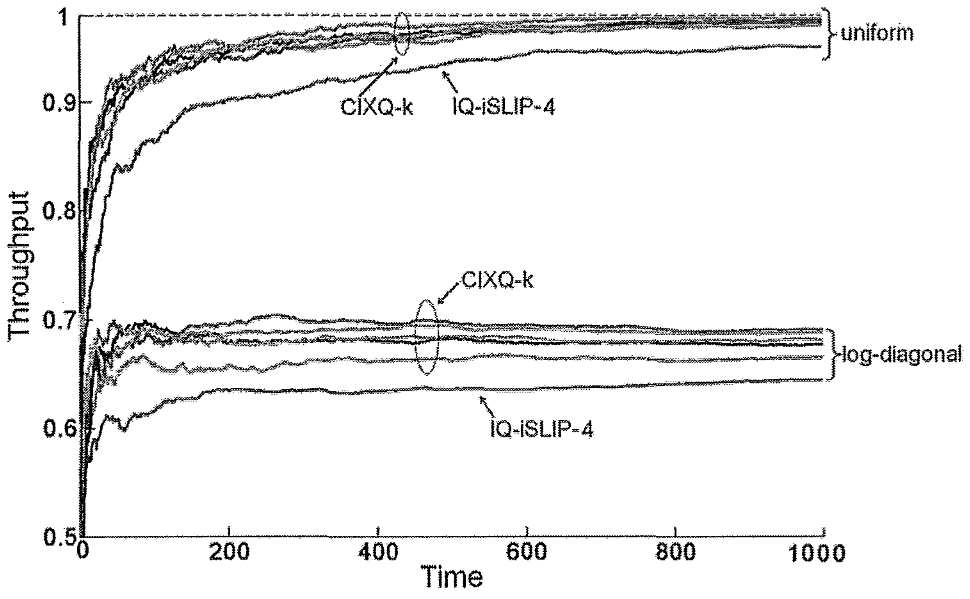


FIGURE 3.5: Throughput performance of a 32x32 *CIXQ* switch under uniform and log-diagonal traffic, where k = 1, 4, 8, 16 and 32

3.6.1 Uniform Traffic and Log-diagonal Traffic

The throughput performance for *IQ*-iSLIP-4 and *CIXQ*-k switch is illustrated in FIGURE 3.5 under uniform traffic and log-diagonal traffic with 100% input load. For uniform traffic, Roberto Rojas-Cessa *et. al.* [22] proved that *CIXQ* switch can reach 100% throughput even with one-cell crosspoint queue. The simulation results in FIGURE 3.4 have the same agreement because the *CIXQ*-k switch, where k = 1, 4, 8, 16 and 32, all achieve 100% throughput for the uniform traffic. In the steady state, it is well know that the *IQ* switch with iSLIP can provide 100% throughput under uniform traffic [7].

However, the throughput described in FIGURE 3.5 does not present this property well because the simulation time is not long enough. In fact, we do observe *IQ* and *CIXQ* switch both reach 100% with a longer running time, about 10,000 time slots. From the observation, we can see that the RR/RR *CIXQ*-k *CIXQ* switch obtains a 100% throughput faster than the *IQ*-4iSLIP switch.

For log-diagonal traffic, the *IQ*-iSLIP-4 switch saturates at about 63% throughput while the *CIXQ* switch has about 68% with a smallest crosspoint queue ($k = 1$). Moreover, the throughput is improved by increasing k from 1 to 32. When k equals the switch size ($k = N = 32$), the *CIXQ*-k switch can reach up to 70%. These properties are also presented in the delay performance plotted in FIGURE 3.6. When the input load is 63%, the delay becomes infinite for *IQ*-iSLIP-4 switch. This phenomenon happens to *CIXQ*-k switch at 68% when k is one.

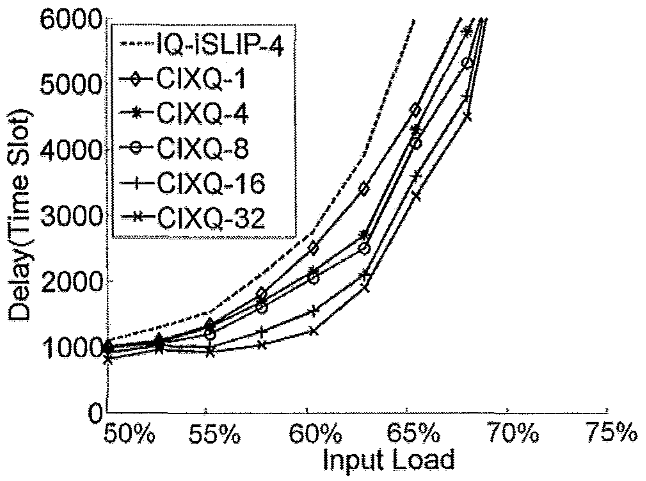


FIGURE 3.6: Delay performance under log-diagonal traffic

3.6.2 Unbalance Traffic

The throughput of *IQ*-iSLIP-4 and *CIXQ*-k switch is illustrated in FIGURE 3.7 under the unbalanced traffic with $u = 0.5$ for value of k from 1 to 32. The throughput in the steady state is about 84% for *CIXQ*-1 and 80% for *IQ*-iSLIP-4 switch when $u = 0.5$. By changing the unbalanced probability from 0 to 1 with an increment of 0.05, the steady-state throughput is described in FIGURE 3.8. The simulation results form U-shape curves. The lowest points are located at $u = 0.5$. If u is zero (uniform traffic), *CIXQ*-k and *IQ-iSLIP-4* achieve 100% throughput, which is consistent with our previous discussion. If u is one, the throughput of the completely unbalanced traffic is also 100%.

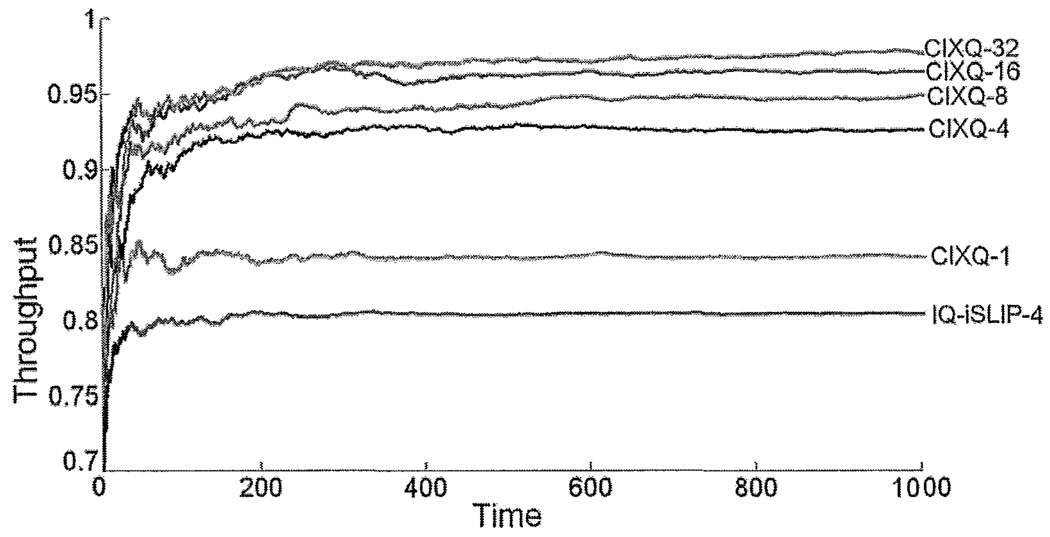


FIGURE 3.7: Throughput performance under unbalanced traffic when $u = 0.5$

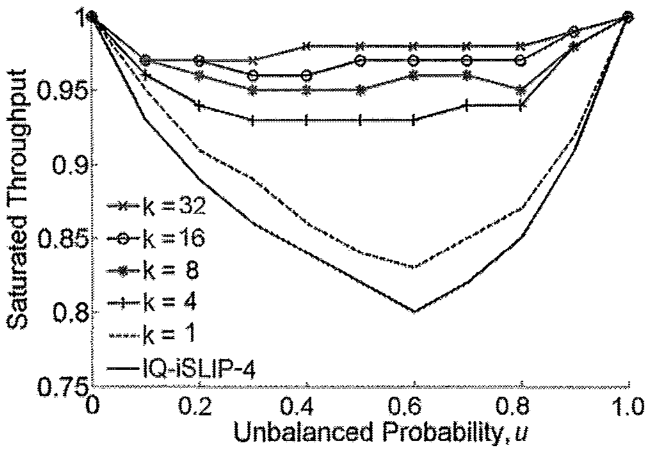


FIGURE 3.8: Throughput of *CIXQ*-k under unbalanced traffic, $N=32$

From FIGURE 3.7 and 3.8, it is obvious that a larger k leads to an improvement in the throughput of *CIXQ*-k switches, and that *IQ-iSLIP*-4 has the lowest throughput performance in all cases. The throughput of the *CIXQ*-k switch almost reaches 98% when $k = N = 32$, i.e. when the switch has $O(N^3)$ buffers. It suggests that 100% throughput is achievable with large enough crosspoint queues although the cost may be intractable.

3.7 Switch Size Effect

The effect of different switch sizes on the performance of *CIXQ* switch is studied in this section using a simplest case of RR/RR *CIXQ*-k switch, i.e. $k = 1$. The throughput of *CIXQ*-1 switch with N = 8, 16, 32 and 64 under unbalanced traffic is provided in FIGURE 3.8. The simulation results show the throughput is not affected by the switch size. For example, all *CIXQ*-1 switches have the same throughput of approximately 85% at $u = 0.5$. Therefore, the performance of *CIXQ* switch appears to be independent of the switch size, which appears to indicate that *CIXQ* switch is scalable.

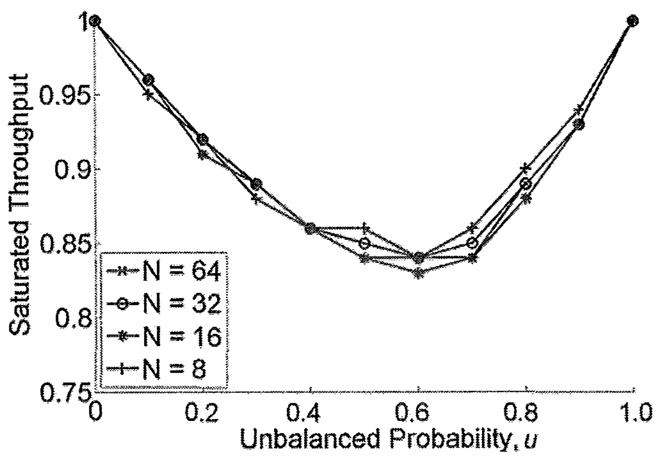


FIGURE 3.9: Throughput of *CIXQ*-1 under unbalanced traffic

3.8 Conclusions

The performance of *CIXQ* switches under different traffic patterns has been studied throughput MATLAB simulations. A *CIXQ*-k switch with Round-robin arbitration and an *IQ* switch with 4-iteration iSLIP are implemented. The simulation results indicate that the RR/RR *CIXQ-k* switch achieves 100% throughput under uniform traffic. As a significant advantage over the *IQ* switch, *CIXQ* switches can schedule the packets at the inputs and outputs separately and independently, therefore simplifying the scheduling with a fast arbitration time. As a significant disadvantage over the *IQ* switch, the *CIXQ* switch requires N^2 FIFO queues in the crossbar, which significantly increase the cost and hardware complexity of the switch. The comparison between the RR/RR *CIXQ-k* and *IQ-iSLIP* switch shows the RR/RR *CIXQ-k* switch has a somewhat better performance even though it uses a less complex scheduling algorithm, at the expense of increased hardware complexity. In the future, scheduling efficiency may become a key issue for switch design as the line rates increase in capacities from Gigabits to Terabits per second. A determination of which switch (*CIXQ*-k or *IQ*) is built would require a thorough study of scheduling algorithm, which is beyond this thesis.

The simulation results also suggest that the throughput of *CIXQ* is proportional to the crosspoint queue size, but independent of the switch size. The simulations draw the same conclusion as presented in [19, 21, 22, 24].

Chapter 4 Design and Implementation

4.1 Introduction

The implementation of *CIXQ* switches has become more realistic with the recent advances in CMOS technology. Many researchers [28, 29, 30, 31, 32] have considered the development and deployment of *CIXQ* switches in high-speed networks. In this chapter, we describe the VHDL implementation of a *CIXQ* switch. The designs are based on two different architectures: *Broadcast-and-Select (B&S)* and *Demultiplexer-Multiplexer (DEMUX-MUX)*. The rest of the chapter is organized as follows: we first introduce the features and describe the overview of the switch designs. The implementation of input port, crosspoint queue and arbiter are described next. Then we show the integration of the whole switch. We also provide a Time Division Multiplex (TDM) mechanism to achieve 100% throughput in the design while a FIFO queue is employed at each input port. Finally, we present the simulation results after the designs are placed and routed to the Altera Cyclone II FPGA.

4.2 Design Features

Unlike the previous approaches in [28, 29, 30, 31, 32], we implement the input/output ports, scheduler and switch fabric into a single FPGA device. The *CIXQ* switches created in the VHDL language have following features:

(1) The switch is designed to handle 64 byte long data cells/packets and variable datapath widths: $w = 4$, 8 and 16 bits.

(2) The switch is built using a modular design that allows us to scale up or down with minor changes.

(3) The round robin (RR) arbitration is used at the output ports. In an *N x N CIXQ* switch, *N* identical RR arbiters are implemented to schedule cell transmission between crosspoint queues and output ports.

(4) Since the primary purpose of this thesis is to develop power models for the *CIXQ* switch, a single FIFO queue is realized at each input port in order to keep the design process simple.

As shown in FIGURE 4.1 (a), the major components of a crossbar switch include: (a) *the input ports*, which stores the arrival packets; (b) *the arbiter*, which identifies and resolves the contention during data transmission; (c) *the switch fabric*, which provides the connection between the input and output of a switch; and (d) *the output ports*. For the implementation of the switch fabric, we consider two different designs: the traditional *broadcast-and-select (B&S)* and *Demultiplexer-Multiplexer (DEMUX-MUX)* designs. An *NxN CIXQ* switch in the B&S design is formed by *N* broadcast busses, N^2 crosspoint queues and *N* multiplexers (see FIGURE 4.1 (b)). A data bus at each input port is used to connect the input queue (*IQ*) to crosspoint queues (*XQs*), while a multiplexer (*MUX*) at each output port is employed to connect *XQ*s to the output port. At the input side, the cells queued in the IQs are broadcasted to *XQ*s through the shared bus. An address filter is used at each XQ to determine which *XQ* will receive the broadcast data. At the output side, the *XQ*s forwarded the cells to the destination output port through a multiplexer based on the scheduling policy.

Instead of using a broadcast bus, the DEMUX-MUX design has *N* demultiplexers (*DEMUX*s) at the input side as displayed in FIGURE 4.1 (c). As a result, the data is forwarded from the input port to the destined *XQ* according to the operation of the DEMUX, other than blindly broadcasting to all *XQs* in the B&S. At the output side, the DEMUX-MUX design follows the same method as the B&S to transmit data to the output ports.

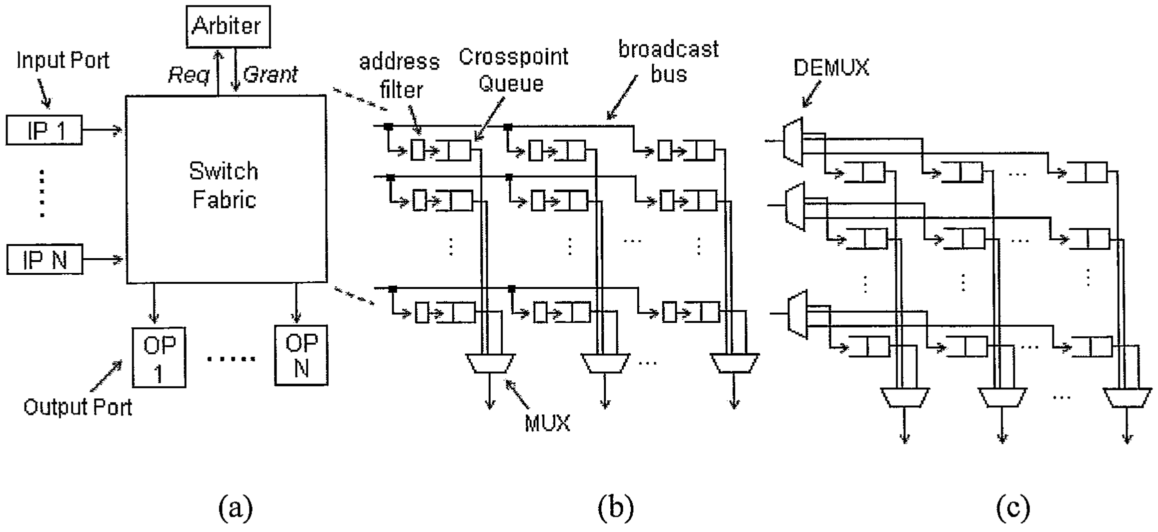


FIGURE 4.1: (a) Switch architecture overview; (b) broadcast-and-select design; and (c) DEMUX-MUX design

4.3 Overview of Switch Designs

A *4x4 CIXQ* switch in the B&S and DEMUX-MUX designs are described in FIGURE 4.2 and 4.3. These designs have the same input and output ports. The inputs are the *w*-bit input data *(data_in1 to data_in4)* and the starting point signals of the incoming packet *(sp1 to sp4)*. The outputs are the output data *(output_port1 to output_port4)*. In addition, the output arbiter is composed of four individual round robin arbiters, each for one output port. Two high level schematics of a *4x4 CIXQ* switch in both designs are provided in Appendix A. Note that the global control signals (*global_clock, global_clear and global_reset*) are not shown in FIGURE 4.2 and FIGURE 4.3 for simplicity.

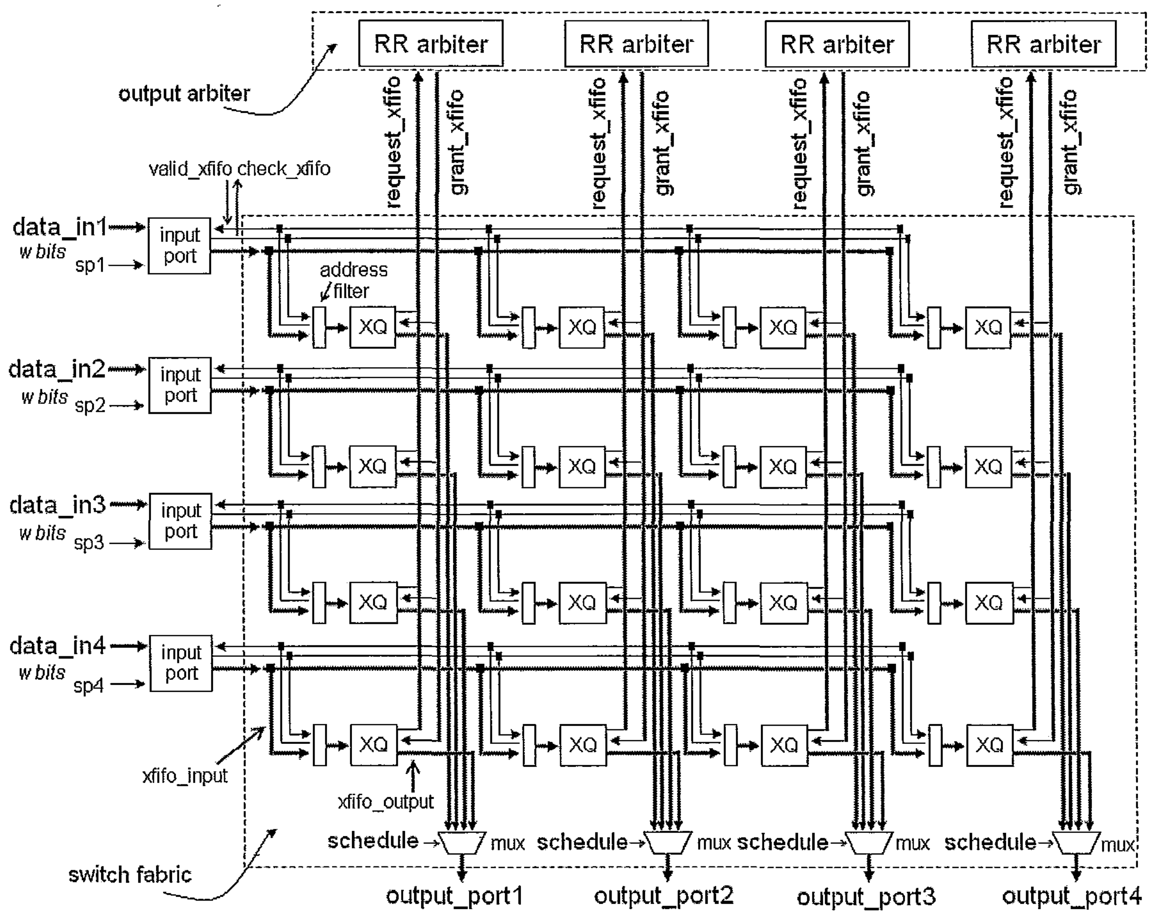


FIGURE 4.2: Block diagram of a 4x4 *CIXQ* switch in B&S design

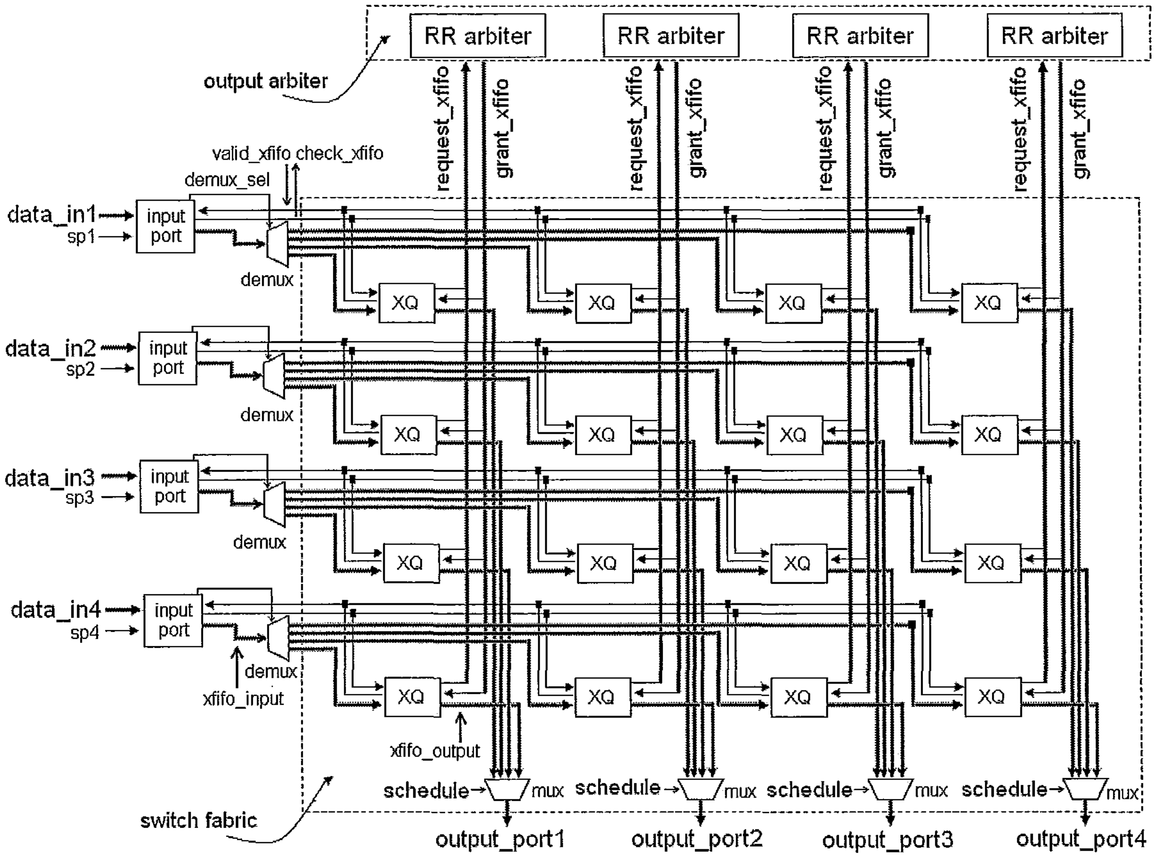


FIGURE 4.3: Block diagram of a 4x4 *CIXQ* switch in DEMUX-MUX design

Referring to FIGURE 4.2 and FIGURE 4.3, the major internal signals in the switch designs are described as follows.

- ***Eligibility query*** *(check_xfifo):* A query signal from the input port to the XQs in a row. This signal is $1+\log_2(N)$ bits wide and indicates which output port the cell is destined to. The input port will broadcast this signal to the *XQ*s to query whether or not the destined *XQ* is eligible for the incoming cell.
- ***Query result*** *(valid_xfifo):* The response of an eligibility query from the *XQ*s to the input port. This signal controls the data transmission from the input port to the

*XQ*s since the input port is only allowed to transfer cells to *XQ*s if *valid_xfifo* is asserted high.

- ***Requests*** *(request_xfifo):* The request signal from the XQs to the output arbiter (RR arbiter). This signal shows that one XQ requests a connection to the output port.
- ***Grants*** *(grant_xfifo):* The grant signal from the output arbiter to the XQs. This signal indicates which XQ is granted for the output connection.
- ***MUX selection*** *(schedule):* The control signal from the output arbiter to the MUX. This signal is in $\log_2(N)$ bits wide and controls the operation of the MUX.
- ***DEMUX selection*** *(demux_sel):* The schedule signal from the input port to the DEMUX. This signal is $\log_2(N)$ bits wide and controls the operation of the DEMUX in the DEMUX-MUX design.

The overall functionality of the switches in both designs can be described as follows. The 64-byte data cells first are queued at each input port based on the order of arrival. The input port then broadcasts an eligibility query to notify the destined *XQ* that a cell is waiting to move in. If the *XQ* is eligible, the input port sends the cell to the destined *XQ* through a broadcast bus in the B&S design or through a DEMUX in the DEMUX-MUX design. The *XQ* then issues a request to the arbiter for an output port. The arbiter grants the incoming requests based on a round-robin order that ensures fair service to all *XQ*s. If the requesting *XQ* is granted, it sends the cell to the destination output port through a MUX.

4.4 Input Port Design

In order to keep the design process simple, each input port employs a single FIFO queue to store the cells that wait for the transmission through the switch fabric. FIGURE 4.4 shows the block diagram of an input port, which contains five sub-blocks: input queue (*IQ*), port controller, enqueue controller, dequeue controller and routing table. A detailed schematic diagram of an input port is available in Appendix B.

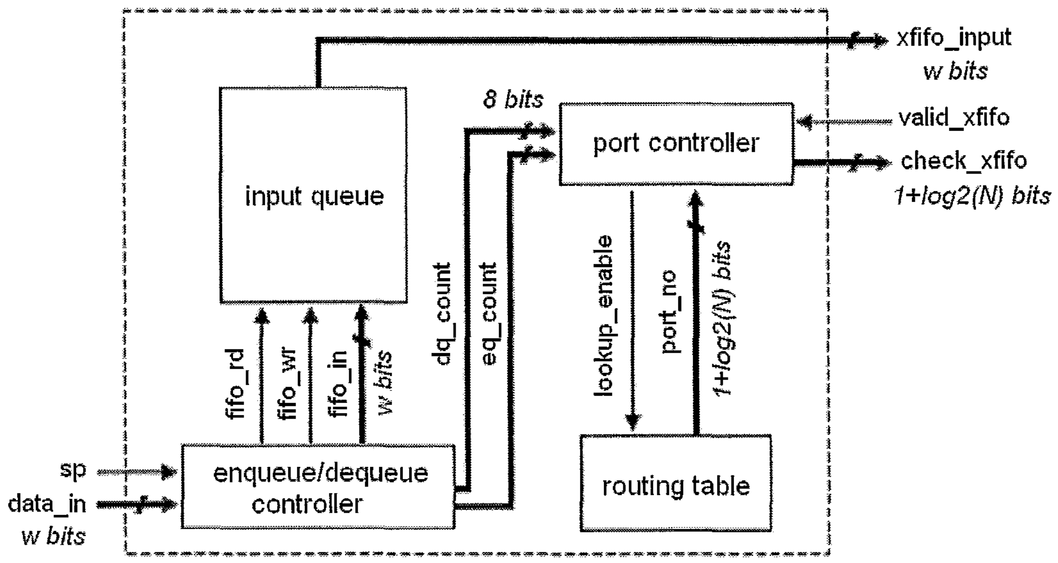


FIGURE 4.4: Input port architecture for an *NxN CIXQ* switch

The inputs of the input port are listed in details as follows,

- *sp:* the starting point of an arrival cell
- *data_in: w-bit* input data
- *valid_xfifo*: the query result signal sent from the XQ to the input port. It indicates the destined *XQ* is eligible for the incoming cell, i.e. the cell is allowed to move from the input port to the *XQ* when *valid_xfifo* is asserted high.

The outputs are described as follows:

- x*fifo_input:* the data being sent from the input port to the XQ.
- *check_xfifo*: a query signal to check the eligibility of the destined XQ.

In addition, as shown in FIGURE 4.4, the input port includes the following internal signals:

- *fifo_in:* the *w-bit* input data from the enqueue controller to the *IQ*.
- *fifo_wr:* a write signal from the enqueue controller to the *IQ*.
- *fifo_rd:* a read signal from the dequeue controller to the *IQ*.
- *eq_count:* an 8-bit counter which counts the number of the words of a cell that have currently been stored in the *IQ*.
- *dq_count:* an 8-bit counter which counts the number of the words of a cell that have currently been moved from the *IQ* to the XQ.
- *lookup_enable:* This signal is sent from the port controller to the routing table. It is initially logic low. If a cell is completely saved into the *IQ*, it is asserted high to active the routing table. After the destination port of the cell is obtained, the signal is reset to its initial value.
- *port_no:* This signal is sent from the routing table to the port controller and is $1+\log_2(N)$ bits wide, where *N* is the size of switch. It is the destination port of an arrival cell. The most significant bit shows the validity of this signal. If it is high, *port_no* is valid to use. Otherwise, it is ignored. For instance, in a 4*x4* switch, if

port_no is "110", it indicates the information is valid since the left-most bit is '1' and the destination port is "10", i.e. the third output port.

In the design of the input port, there are two counters used in the enqueue process and dequeue process: *eq_count* and *dq_count* (see Appendix B). Both counters are set to 8 bits wide to handle various datapath widths (*w = 4, 8, 16 bits*).

- Enqueue counter (*eq_count*): It counts the number of bits of an arrival cell that are currently saved into the *IQ*. It only starts counting from zero if a new cell arrives (*sp='1'*) and stops once a cell is completely stored into the *IQ*.
- Dequeue counter (*dq_count*): It counts the number of bits of a cell that are currently dequeued from *IQ* to *XQ*. The dequeue counter starts counting up once the cell is being moved to the XQ. If a whole cell has been forwarded to the XQ, the dequeue counter is reset and disabled.

The overall functionality of an input port can be described as follows. The *w-bit* word of data is first moved into the *IQ* once *fifo_wr* is asserted high (*fifo_wr = '1'*). If a cell is completely enqueued, the routing table is enabled to look up the destination (*lookup_enable = '1'*). If the destined *XQ* is eligible for the incoming cell, the data is moved from the *IQ* to the *XQ* by setting *fifo_rd = '1'*. After a cell is completely moved to the XQ, the internal signals are reset to their initial values.

Referring to the Appendix B, the operation of the input port is mainly controlled by three controller state machines, and the detailed algorithms for each state machine are given in the following subsections.

4.4.1. Enqueue controller state machine

4.4.2. Port controller state machine

4.4.3. Dequeue controller state machine

4.4.1 Enqueue controller state machine

The enqueue controller state machine shown in FIGURE 4.4 controls the enqueue process. It keeps track of the starting point of an arrival cell and counts the number of data bits of a cell that have been enqueued. The enqueue process is only enabled if the following conditions are true: (1) a new cell arrives and (2) the input queue is eligible for the incoming cell. In addition, an input queue is determined eligible for the incoming cell if the following conditions are true: (1) the input queue is not full; (2) the input queue has free space for at least one cell; and (3) the enqueue process is not active.

The enqueue controller state machine is responsible for the following tasks while not in RESET mode and the input queue is eligible for the incoming cell:

1. Wait until a new packet arrives, i.e. *sp* = *'1'*, start the enqueue counter.
2. Starting writing data into input queue by setting *fifo_wr* = *'1'*.
3. If a cell is completely saved, disable the enqueue process by setting *fifo_wr* = *'0'* and reset enqueue counter. Go to Step 1.

   (A state diagram for the enqueue process is given in Appendix E.1)

4.4.2 Port controller state machine

If the *IQ* receives a cell, the port controller needs to determine which output port the cell travels to. It accesses the routing table for the destination port and then sends out a query to check the status of corresponding *XQ*. When the destined *XQ* is eligible for the incoming cell, the port controller notifies the dequeue controller to forward the cell from *IQ* to *XQ*. The port controller will be not reset until a cell is completely moved to *XQ*.

The port controller state machine is responsible for the following tasks while not in RESET mode and the input queue is not empty:

1. If a cell is completely enqueued into the input queue, go to Step 2. Otherwise, the destination output port is set to 0 indicating no any further action is taken by the controller. Go to Step 1.
2. Enable the routing table to find the destination port.
3. Wait until the cell is completely forwarded from the *IQ* to the *XQ*, reset the signals and go to Step 1.

   (A state diagram for the port controller is given in Appendix E.2)

4.4.3 Dequeue controller state machine

The dequeue controller state machine shown in FIGURE 4.4 controls the process for moving the cell from the input queue to the *XQ*. The dequeue process will only be enabled if the destined *XQ* is eligible for the incoming cell. After a cell is completely transferred to the *XQ*, the dequeue controller is reset.

The dequeue controller state machine is responsible for the following tasks while not in RESET mode and the input queue in not empty:

1. If the *XQ* is eligible for a cell, i.e. *valid_xfifo* = *'1'*, start moving the cell from the *IQ* to the XQ and enable the dequeue counter, go to Step 2. Otherwise, go to Step 1.
2. If the entire cell is moved to XQ completely , then reset the state machine and go to Step 1.

(A state diagram for the dequeue process is given in Appendix E.3)

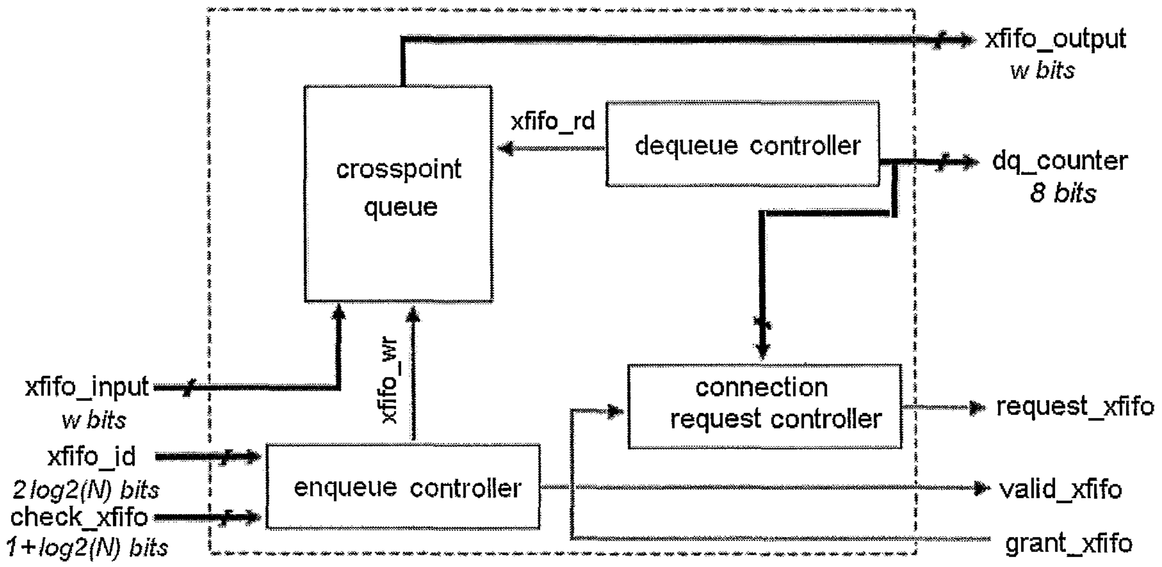


FIGURE 4.5: The inputs and outputs of a crosspoint queue design

4.5 Crosspoint Queue Design

In a *CIXQ* switch, the crosspoint queue is designed to receive and forward the data from the input ports to the output ports. FIGURE 4.5 shows the block diagram of a crosspoint queue design containing four sub-blocks: crosspoint queue (XQ), enqueue controller, dequeue controller and connection request controller. A detailed schematic diagram of a crosspoit queue design is available in Appendix C. The global control signals which include *global_clock*, *global_reset* and *global_clear* are not displayed for simplicity in FIGURE 4.5.

The inputs of the crosspoint queue design are described as below.

- *xfifo_in*put: the *w-bit* input data moving from the input port.
- *check_xfifo:* a query signal from the input port to check the eligibility of a *XQ*, which is $1+\log_2(N)$ bits wide.
- *grant_xfifo:* the grant signal from the output arbiter. It indicates that the output request is granted when it is logic high, i.e. the *XQ* is allowed to forward data to the output port.

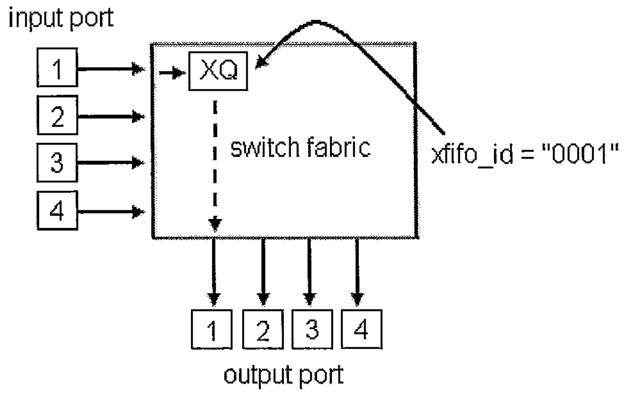


FIGURE 4.6: A *XQ* in *4x4 CIXQ* switch

- *xfifo_id*: the identifier (ID) of a *XQ* inside the switching fabric. It is $2*\log_2(N)$ bits wide, where *N* is the switch size. The leftmost $\log_2(N)$ bits specify the input port number while the rightmost $\log_2(N)$ bits represent the output port number. For example, a *XQ* with the identifier of *"0001"* in a *4x4* switch holds the connection between the input port *0* and output port *1*, i.e. the first input and the second output, as shown in FIGURE 4.6.

The outputs of the crosspoint queue can be interpreted as follows.

- *request_xfifo:* an output request, which is sent to the arbiter for output connection.
- *valid_xfifo*: the status of the current *XQ*, which is sent to the input port in response to the query signal. When it is logic high, data can be moved from the input port to the *XQ*. Otherwise, no data can be enqueued into the *XQ*.
- *xfifo_output:* the *w-bit* output data of the *XQ* being sent to the output ports.
- *dq_counter:* the number of data bits of a cell being sent to the output. The output arbiter only grants a *XQ*'s request if the dequeue process is disabled. Otherwise, the request will be ignored. Therefore, it is sent to the output arbiter for the purpose of monitoring the dequeue process at the XQ

As shown in FIGURE 4.5, the crosspoint queue includes the following internal signals:

- *xfifo_wr*: a write signal from the enqueue counter to the crosspoint queue.
- *xfifo_rd*: a read signal from the dequeue counter to the crosspoint queue.

As illustrated in Appendix C, the *XQ* has only one counter to incorporate with the dequeue process, while the enqueue process is controlled by the eligibility query *(check_xpb)*. The operation of enqueueing and dequeueing at the XQ is descried as follows:

- The enqueue process: the enqueue controller first checks if the output port number representing in a valid query *(check_xfifo)* is the same as the one in its own ID *(xfifo_id)*. If they match, the current *XQ* is considered to be the destination of the incoming cell, and the *XQ* starts receiving the cells. Otherwise, the XQ stops receiving the cells.
- The dequeue process: if output arbiter grants the request (*grant_xfifo = '1'*), the XQ moves the cell to the output and enable the 8-bit dequeue counter (*dq_count*). Once an entire cell is moved out completely, the dequeue process and the dequeue counter are disabled.

The overall functionality of the crosspoint queue design can be described as follows. If the *XQ* is eligible for incoming data, i.e. *valid_xfifo = '1'*, and the query signal matches its own ID, it starts receiving the cell from the input port. Otherwise, it stops receiving any data. A non-empty XQ sends a request to the arbiter for an output connection by setting *request_xfifo = '1'*. The arbiter grants the incoming requests based on the round-robin order. A requesting XQ does not move the cell to the output port until it is granted by the arbiter. After a cell has completely moved out of the *XQ*, the grant signal is reset to zero, and the *XQ* stops moving data until its next request is granted.

Refer to Appendix C, the operation of the *XQ* is mainly controlled by three state machines. The detailed algorithms for each state machine are given in the following subsections.

4.5.1. Enqueue controller state machine

4.5.2. Output request controller state machine

4.5.3. Dequeue controller state machine

4.5.1 Enqueue controller state machine

The enqueue controller state machine shown in FIGURE 4.5 controls the process to move cells from the input port to an eligible *XQ*. The *XQ* is eligible for the incoming cell if the following conditions are true: (1) the *XQ* is not full; (2) it has free space for at least one cell; (3) the enqueue process is not enabled.

The enqueue controller state machine is responsible for the following tasks while not in RESET mode and the *XQ* is not full:

1. If the query signal *(check_xpb)* matches the XQ's ID *(xpb_id)*, go to Step 2. Otherwise, go to Step 1.
2. If *XQ* is eligible, set *valid_xfifo = '1'* and sent it to the input port.
3. Wait for 2 clock cycles for the input port to receive the signal in Step 2. If *check_xfifo* remain unchanged, then start receiving data from the input port. Otherwise, reset the *valid_xfifo = '0*.
4. Reset the state machine and go to Step 1.

   (A state diagram for the enqueue process is given in Appendix E.4)

4.5.2 Output request controller state machine

If the *XQ* has received a newly arrived cell, the output request controller in FIGURE 4.5 issues an output request to the arbiter. Once the arbiter grants the request, the controller withdraws its request. The controller will not be reset until a cell has been moved to the output port.

The request controller state machine is responsible for the following tasks while not in RESET mode and the *XQ* is not empty:

1. If the *XQ*'s request is not granted currently, make a request to the arbiter by setting *request_xfifo = '1'* and go to Step 2. Otherwise, go to Step 1.
2. Wait until the request is granted, i.e. *grant_xfifo = '1'*, then withdraw the request by setting *request_xfifo = '0'*.
3. Wait until a packet is completely dequeued to output port, go to Step 1

(A state diagram for the request process is given in Appendix E.5)

4.5.3 Dequeue controller state machine

The dequeue controller state machine shown in FIGURE 4.5 controls the process to move the cells from *XQ* to output port. If a request is granted by the arbiter, the controller forwards the data to output ports. The dequeue controller is not reset or disabled until a cell is completely dequeued.

This state machine is responsible for the following tasks while not in RESET mode and the *XQ* is not empty:

1. If the request is granted, i.e. *grant_xfifo* = *'1'*, change the status to ready for reading data and enable the dequeue counter, then go to Step 2. Otherwise, go to Step 1.
2. Clear the dequeue counter to zero and start moving data to the output port.
3. Wait until an entire cell is moved to the output port, then disable the dequeue counter and go to Step 1

(A state diagram for the dequeue process is given in Appendix E.6)

4.6 Arbiter Design

Round-robin (RR) scheduling is used to resolve the conflicting requests of the *XQ*s at the output side. In each time slot, each *XQ* sends at most one request and receives at most one grant. Therefore, the arbiter need to select one of N^2 requests based on the round-robin service discipline, and notifies each requesting XQ whether or not its request is granted per time slot.

Normally, the output arbiter in a *CIXQ* is designed as a single component with at least N^2 inputs and outputs to deal with the incoming request and grant signals. However, the hardware implementation becomes too complicated and difficult for large switches. To reduce the design complexity, the arbiter in our design is composed of N individually identified round-robin arbiters, and each of them only handles N requests from the *XQ*s at each output port, as shown in FIGURE 4.2 and FIGURE 4.3. More importantly, this design is easy to scale up for large *CIXQ* switch in the future implementation.

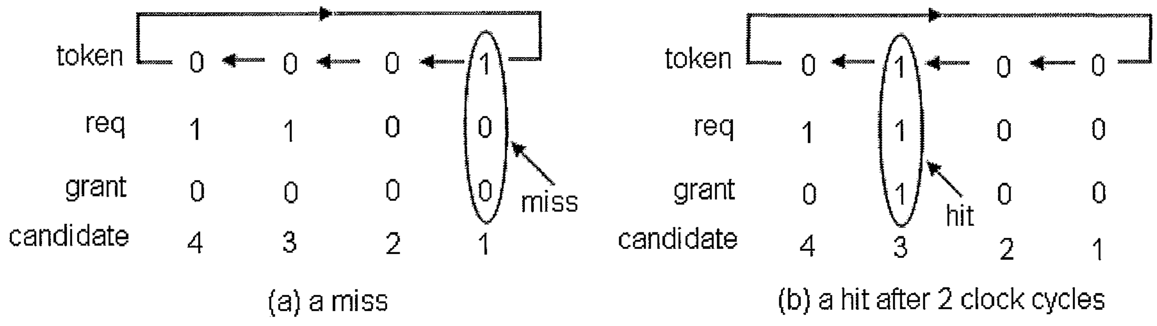


FIGURE 4.7: A hit and a miss in a round-robin arbiter

In our approach, the grant process of the arbiter is viewed as an exclusive token granting request in a round robin pattern [74]. If the request is in the same position as the token, it is called a hit, otherwise it is a miss. FIGURE 4.7 describes an example of a hit and a miss for an arbiter managing 4 candidates, where the third and fourth candidates are requesting an output connection. The token is shifting to the left in each cycle. FIGURE 4.7 (a) presents a case of a miss because the token is in the position of the first candidate, who is not sending a request at the moment. A miss also occurs when the token moves to the position of second candidate. In next cycle, a hit happens because the request of the third candidate is in the same position as of the token, as illustrated in FIGURE 4.7 (b).

This design is simple and in low cost, but the grant process is inefficient. Although an improvement has been proposed in [74], the current design is acceptable for small N. For large N, the ring delay will become unacceptable, and a better round-robin arbiter must be designed.

The block diagram of a RR arbiter is shown in FIGURE 4.8, which contains four sub-blocks: ring counter, hit detector, token controller and output controller. A detailed schematic diagram of the RR arbiter in a 4x4 *CIXQ* switch is available in Appendix D. The inputs of the arbiter are as follows:

- Requests signals (*request_fifo1 ... request_fifoN*): They are from the XQs and indicate which XQ is requesting an output connection.
- Deuqueue counters (*dq_count1 ... dq_countN*): They are from the XQs and indicate the status of dequeue process at the XQs.

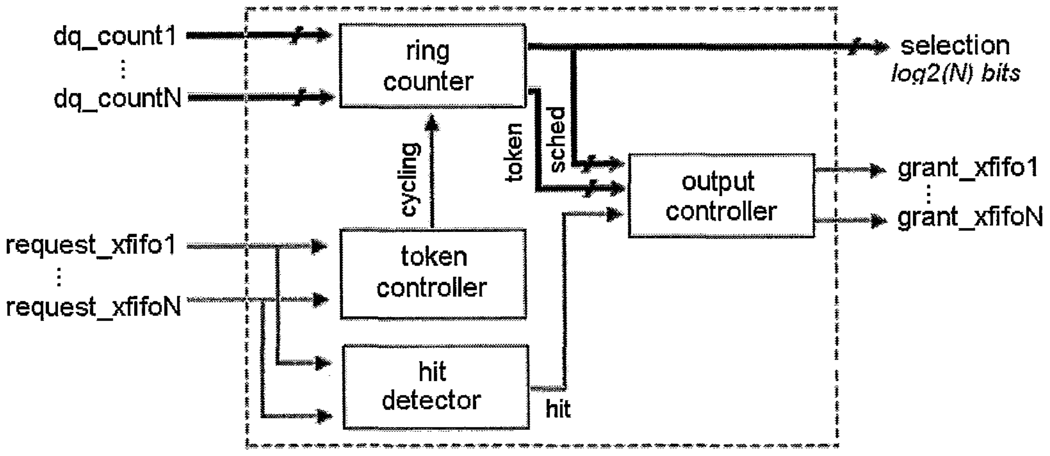


FIGURE 4.8: Block diagram of a round-robin arbiter

The outputs of the arbiter include:

- Grants signals (*grant_xfifo1 ... grant_xfifoN*): they are sent to the XQs and indicate which requesting XQ is granted.
- Selection signal (*selection*): they are sent to the MUX in $\log_2(N)$ bits wide and inform the MUX which XQ is selected as the output.

The operation of the arbiter is mainly controlled by three processes and one state machine: a hit detector process, token controller process, a ring counter process, and output controller state machine (demonstrated inside the dash line box of Appendix D).

Taking the example of a RR arbiter dealing with four candidates, the detailed algorithms for each process are given in the following subsections.

4.6.1 A hit detector process

This process detects the occurrence of a hit in the current cycle (see FIGURE 4.8). By definition, a hit occurs if the request and the cycling token are in the same position. A hit detector simply compares each single bit of the request and token signals. If the comparison result is logic high, a hit is confirmed. Otherwise, a miss is considered. With the use of *AND* and *OR* gates, the behaviour of a hit can be represented as below,

```
hit <=  (token(0) AND req0) OR (token (1) AND req1) OR
        (token(2) AND req2) OR (token(3) AND req3);
```

4.6.2 Token controller process

This process controls the movement of the token (see FIGURE 4.8). If a miss happens, the token needs to move to next bit position. The token controller is used to move the token to next position based on the following conditions:

(1) At least one XQ is requesting for an output connection, i.e. *lmask* = *'1'*. The internal signal *lmask* represents which XQ is sending a request, and it is coded as,

```
lmask <= req0 OR req1 OR req2 or req3;
```

As long as *lmask* is asserted high, it indicates that at least one XQ is requesting an output connection.

(2) None of the existing requests is granted, i.e. *lgnt= "0000"*. The internal signal *lgnt* indicates which requesting *XQ* is granted. If any bit of *lgnt* is changed to one, it means the corresponding *XQ* is granted. Note that only one bit in *lgnt* can be changed from 0 to 1 since at most one grant can be made for each cycle.

4.6.3 A ring counter process

Refer to FIGURE 4.8, a 2-stage Johnson ring counter is used to generate a ring counter with the repeating pattern: *00, 10, 11* and *11* [75]. Besides, the ring counter is also responsible for transforming the repeating pattern, *Q*, into two control signals: *token* and *sched*, which are in turn used in the output controller. Table 4.1 shows the truth table of a 2-stage Johnson counter and the corresponding values of *token* and *sched.*

Table 4. 1: Truth table of a Johnson counter

Clock Pulse	Q1	Q0	token	sched
0	0	0	0001	00
1	1	0	0010	01
2	1	1	0100	10
3	0	1	1000	11

4.6.2.4 Output controller state machine

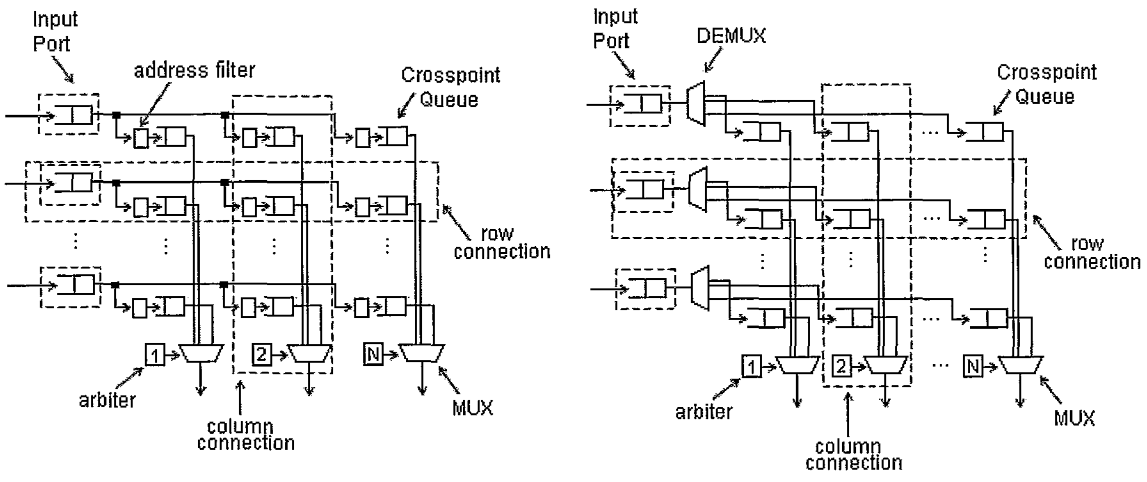
The output controller state machine is used to drive the actual scheduling signals: *lgnt* and *schedule*. *lgnt* in *N* bits wide indicates which XQ is granted and *schedule* in $\log_2(N)$ bits wide is sent to the multiplexer for an output selection. This state machine is responsible for the following tasks while not in RESET:

1. Wait until a hit occurs, i.e. *hit* = *'1'*, send the grant signals to the *XQs* and the selection signals to multiplexer. Then go to Step 2. Otherwise, go to Step 1.
2. Wait 2 clock cycles for the XQs and multiplexer to receive the signals.
3. Wait until the hit signal is reset, i.e. *hit* = *'0'* and no data is being dequeued to output port, i.e. *dq_count* = *0*, then reset the state machine and go to Step 1.

   (A state diagram for output controller state machine is given in Appendix E.7)

4.7 Integration of the Switch Components

A full body of *CIXQ* switch can be integrated from basic components: input port, crosspoint queue, and arbiter, which are built as individual blocks in VHDL. In a *CIXQ* switch, the arrival cells stored at the input ports move through the switch fabric following two independent data paths: one is to forward the packets from input ports to *XQs* based on their destination, referred to as a row connection of the switch, and the other is to move the packets from *XQs* to the output ports based on the round-robin service discipline, referred to as a column connection of the switch. An *NxN CIXQ* switch can be viewed as a structure with *N* row connections and *N* column connections. FIGURE 4.9 shows the row connection and the column connection of a *CIXQ* switch in the broadcast-and-select and DEMUX-MUX designs.



(a) (b)

FIGURE 4.9: A row connection and a column connection of the *CIXQ* switch in (a) broadcast-and-select design; and (b) DEMUX-MUX design.

The row connection for B&S design is formed from an input port and *N* crosspoint queues, while the row connection for DEMUX-MUX design requires an input port, *N* crosspoint queues. Both designs have the same architecture of a column connection, which needs an arbiter and MUX. In the following subsections, a *4x4 CIXQ* switch is presented as an example for the integration of the full switch body.

4.7.1 A Row Connection

For a *4x4 CIXQ* switch, the B&S design requires one input port and four crosspoint queues in a row connection (see FIGURE 4.10 (a)). The DEMUX-MUX design needs the same components but a DEMUX is also needed between the input port and crosspoint queues (see FIGURE 4.10 (b)). Under the modular design concept, the row connection is built as an individual component.

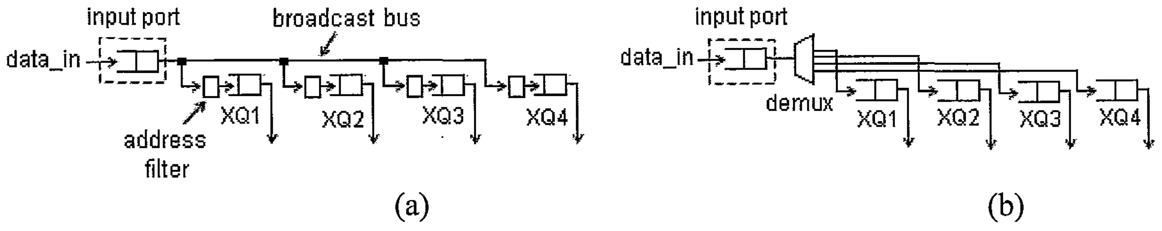


FIGURE 4.10: Block diagram of a row connection in
(a) broadcast-and-select design;and (b) DEMUX-MUX design.

4.7.2 A Column Connection

A column connection in both B&S and DEMUX-MUX design is formed by the same components: an output arbiter and a MUX, as shown in FIGURE 4.11. The column connection is designed as an individual building block in the principle of modular design.

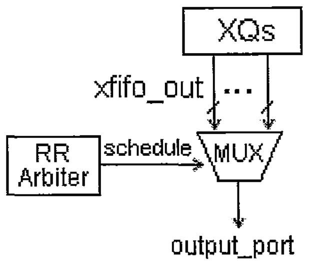


FIGURE 4.11: A column connection in a *4x4 CIXQ* switch

4.7.3 Integration of the Switch

A *CIXQ* switch with four input and output ports can be integrated from four row connections and four column connections. The overall structure of a *4x4 CIXQ* switch is described in FIGURE 4.12 for B&S design and FIGURE 4.13 for DEMUX-MUX design.

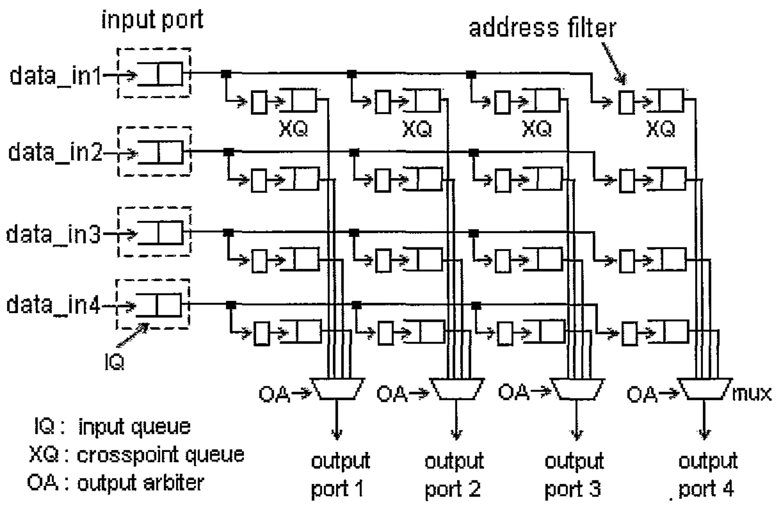


FIGURE 4.12: A full body of *4x4 CIXQ* switch in the broadcast-and-select design

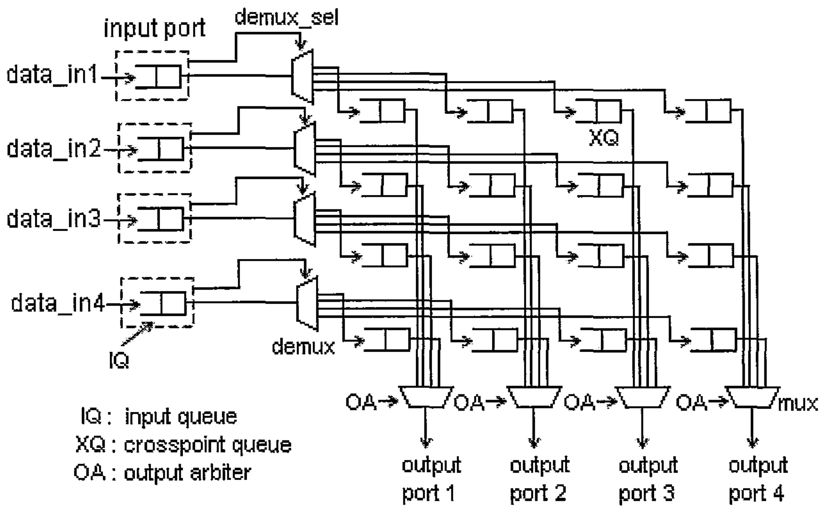


FIGURE 4.13: A full body of *4x4 CIXQ* switch in the DEMUX-MUX design

4.8 Achieving 100% throughput

In attempt to analyze the power consumption in next chapter, the *CIXQ* switches without VOQs need to operate at 100% throughput. But their throughput is limited to a value less than 100% due to the Head-of-Line (HOL) blocking. In this section, we will introduce a deterministic TDM scheme to achieve 100% throughput for a given input pattern, just as iSLIP can achieve a deterministic TDM scheme with 100% throughput for a given input pattern.

4.8.1 The existence of HOL blocking

In a *CIXQ* switch with FIFO queues at the inputs, the switch fabric can only move the packets at the head of the queue per cycle. If an arrival packet at the input port is destined to a non-eligible XQ, which has no free space for at least one packet, this packet must wait until the XQ becomes eligible before forwarding. As a result, the HOL blocking arises because the rest of the packets in that queue are blocked by the HOL packet, even if there is no contention at the destination XQs for those packets.

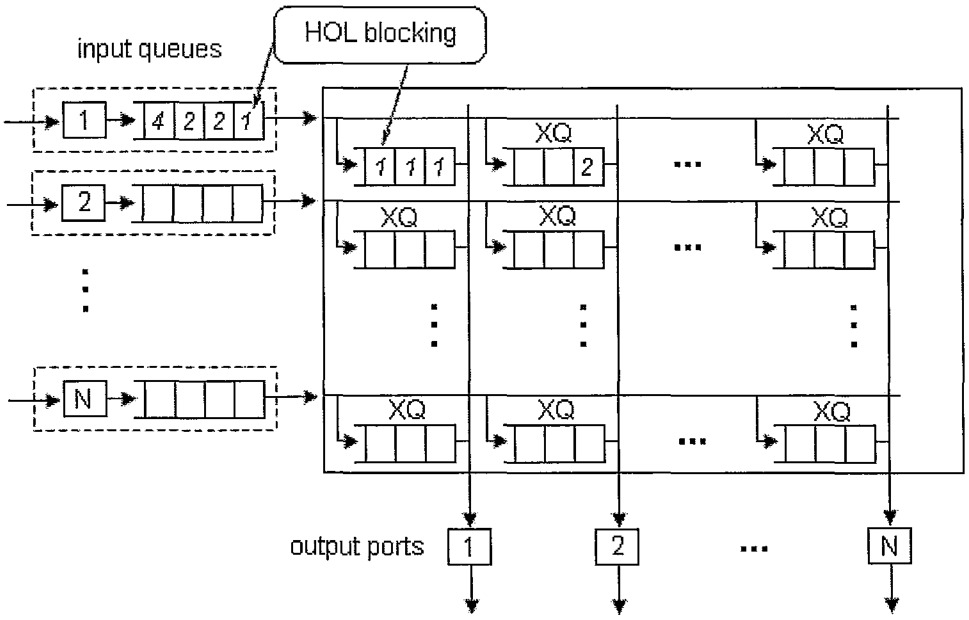


FIGURE 4.14: HOL blocking in a *CIXQ* switch without VOQs

An example of a HOL blocking that exists in our design is described in FIGURE 4.14. The HOL packet at input port *1* cannot be moved out of the queue because the destined XQ has not enough space for a packet. The other packets can only wait in the queue until the HOL packet is removed although their destined XQs may have free space.

4.8.2 A TDM Scheme for 100% throughput

For power analysis, we assume the cells are moved through the switch with perfect scheduling without any internal blocking, such as the HOL blocking. A simple and straightforward method to achieve this goal is to increase the size of the XQs to have sufficient space to temporally store the packets while any blocking occurs. However, this method requires plenty of memory available in the device. Moreover, we have no way to know how large the XQs should be to remove the internal blocking. So, this method is not practical. Another effective method is to schedule the cells from the input ports to the XQs in a pre-computed TDM order, i.e. in a circular order (round-robin order). Although this routing scheme is not realistic, it is sufficient for the power analysis by providing 100% throughput.

4.8.3 Implementation of the TDM scheme

The routing table can be designed as a ring counter, shifting to the left in each cycle. A detail schematic of the routing table in a *CIXQ* switch is shown in FIGURE 4.15. The inputs of the routing table are described as follows:

- *dq_count*: it is the dequeue counter from the crosspoint queue and indicates the number of data bits of a cell is currently moved from the input queue to XQ.
- *lookup_en*: a signal from input port and controls the operation of routing table. The routing table operates only if *flow_en* is asserted high.

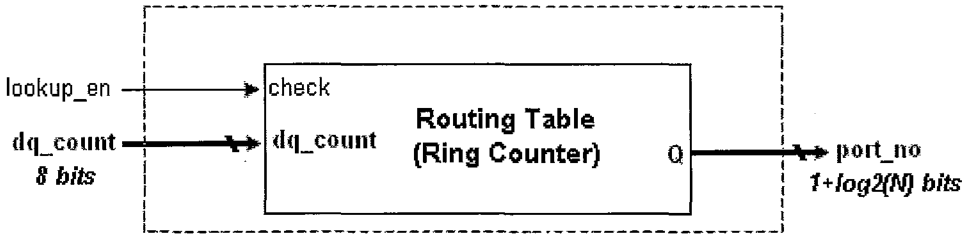


FIGURE 4.15: Detail schematic of a routing table

The output of the routing table is *port_no* in $1+\log_2 N$ bits wide, where the left-most bit is the validity of this information and the rest of bits represent the output port number (destined XQ).

The operation of the routing table can be described as follows. When *lookup_en* is asserted high (*lookup_en = '1'*) and the dequeue process at the XQ is idle (*dq_count = 0*), the sequence of the ring counter is sent out as *port_no* to schedule the arrival cells from the input port to the output port.

4.9 Simulation Results

4.9.1 Simulator

Altera Quartus II simulator [76] is used to simulate and test the designs in the experiments. FIGURE 4.16 illustrates the simulation flow of Quartus II simulator. Tow types of simulation are supported: functional and timing simulations. To perform functional simulations, a functional simulation netlist must be first generated. A functional netlist file is a flattened netlist extracted from the design files that does not contain timing information. For timing simulations, place-and-route and static timing analysis must be perform first to generate timing simulation netlist, which includes timing delays of each logic block and the routing delays. The most common input stimulus for the Quartus II simulator is Vector Waveform Files (*.vwf* files).

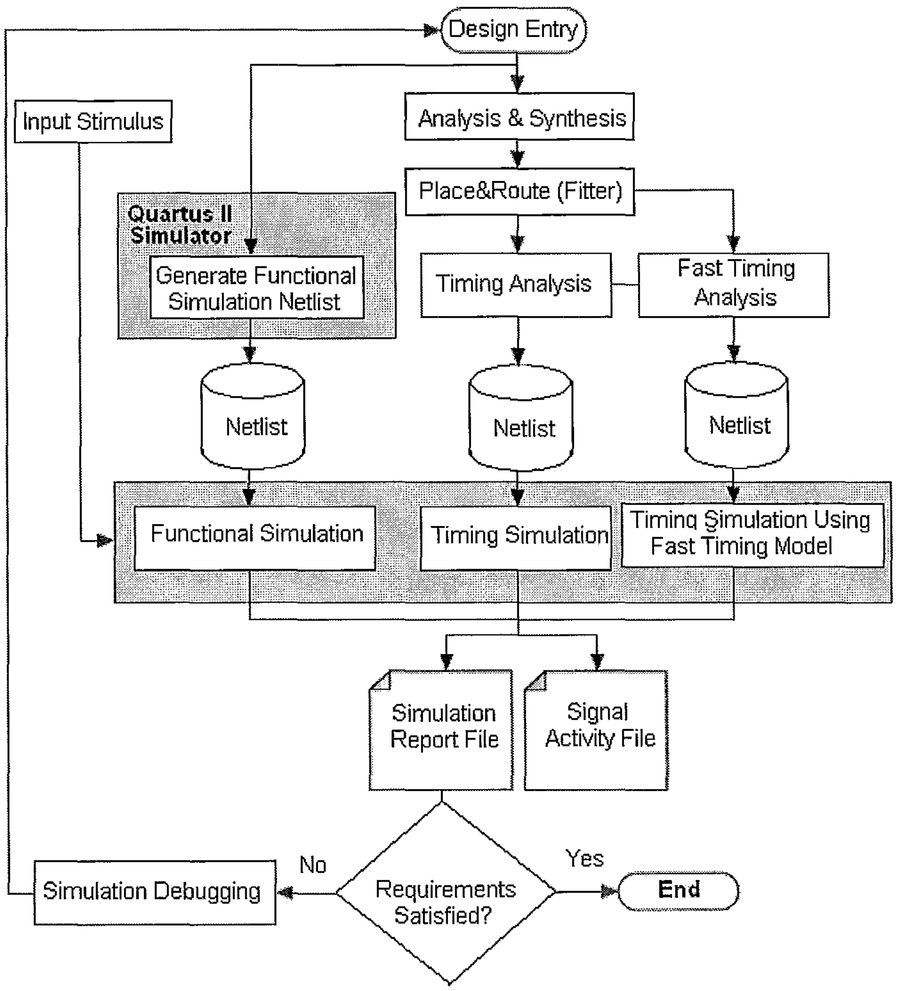


FIGURE 4.16: Quartus II simulation flow

4.9.2 Device

Most commercial FPGAs implement logic function using look-up-tables (LUTs). As depicted in Figure 2, several LUTs, flip flops, and multiplexers are grouped together into a logic element (LE). Logic elements are then grouped together into a bigger logic structure, logic block (LB), in a hierarchical architecture. The Altera Cyclone II FPGA family offering up to 68,416 LEs is used to simulate and test the components and the designs. The device resources available in Cyclone II FPGAs are listed in Table 4.2 [77]. The summary of resource usage for each design with 4-bit datapath is given in Table 4.3.

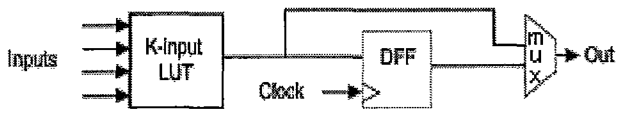


FIGURE 4.17: A logic element in FPGA

Table 4.2: Device Resources of Cyclone II FPGAs

Device	Total Logic Elements (LEs)	RAM blocks M4K	Total PLLs	Embedded Multipliers 9-bit Elements	Total pins	Total memory bits
EP2C35	33,216	105	4	70	475	483,840
EP2C50	50,528	129	4	172	450	594,432
EP2C70	68,416	250	4	300	622	1,152,000

Table 4.3 Summary of resource usage for *CIXQ* switch with $w = 4$ bits

Resource	*4x4* switch Resource Usage	*8x8* switch Resource Usage	*16x16* switch Resource Usage
Logic Elements (LEs)	1,718	6,563	23,999
Registers	1,078	4,065	14,769
I/O Pins	35	67	131
Virtual Pins	0	0	0
Memory Bits	294,912	393,216	574,464
Embedded Multiplier 9-bit Elements	0	0	0
PLLs	0	0	0

4.9.3 Discussion

Using Vector Waveform Files as the stimulus inputs, the overall switch designs in the B&S and DEMUX-MUX architectures with sizes of *4x4*, *8x8* and *16x16* are simulated and tested on Cyclone II EP2C70 device by Quartus II simulator. Timing simulations are performed. The simulation results show the switch designs are functional properly as described in this chapter.

Appendix G presents the simulation result of a *4x4 CIXQ*-4 switch with 8-bit datapath and a clock frequency of 50MHz in $200\mu s$. At the beginning, the switch was cleared and reset. The data is input to every input port of the switch simultaneously. The input frame pulse signals (*sp1 to sp4*) mark the beginning of the incoming cells. According to the deterministic arriving order of the cells and the pre-computed TDM scheduling algorithm, the cells are sent through the switch fabric to the output lines.

Chapter 5 Power Models

5.1 Introduction

Power has become a critical design concern for VLSI designs, especially when the CMOS feature sizes are decreased to nanometre scale. Designers must consider the power in the early design stage when they make architectural trade-offs. This raises a problem because accurate power estimation is usually made after a chip layout is performed. At this stage, it may be too late or too expensive to fix the high power problem. Therefore, a power estimation or analysis tool is needed for the designer to explore the power efficiency of devices early in the design flow. A lot of power models have been presented for various crossbar switch architectures, such as the network routers [66, 67] and switches [68, 69]. In a previous work, D. G. Simos [70] is the first attempt to measure and report the power consumption of *CIXQ* switches. To our best knowledge, no power model is developed for *CIXQ* switches thus far.

The power dissipation in CMOS VLSI circuits has two major components: static power and dynamic power. The dynamic power of a VLSI standard cell can be modeled by

$$P = \frac{1}{2} f_{clk} f_{duty} (C_{\text{int}} + C_{load}) V_{dd}^2 \quad (5.1)$$

where f_{clk} is the clock or operating frequency, f_{duty} is the probability of a signal transition at the output port, also called duty cycle, V_{dd} is the core voltage, C_{int} is the intrinsic capacitance of the cell switched during an output transition, and C_{load} is the capacitive load of the wires being driven by the cell, including the input capacitance of the next cell(s) being driven.

In this chapter, we propose analytical models for the dynamic power of *CIXQ* switches built in an FPGA. As the major contribution of this thesis, we are the first to propose power models for CXIQ switches. The rest of the chapter is organized as follows. We first described the methodology to model the switch power. Then we present the analytical power models for the basic components of the switches as well as the power models for the *IQ* and *CIXQ* switches. Finally, we verify and discuss the switch power models based on the measurements from the experiments.

5.2 Experimental Methodology

In order to model the switch power, we only consider the power dissipation inside the switch fabric because the input ports and output ports can be implemented separately [29, 30]. As described in chapter 4, a *CIXQ* switch fabric circuit consists of four basic components: (1) demultiplexers (DEMUX); (2) multiplexers (MUX); (3) broadcast busses (BUS); and (4) crosspoint queues (XQ). Referring to FIGURE 4.1, the power dissipation of an *NxN CIXQ* switch in the DEMUX-MUX and Broadcast-and-Select (B&S) designs can be estimated by following equations:

$$P_{DEMUX-MUX} = N \cdot P_{DEMUX} + N^2 \cdot P_{XQ} + N \cdot P_{MUX} \quad (5.2)$$

$$P_{B\&S} = N \cdot P_{BUS} + N^2 \cdot P_{XQ} + N \cdot P_{MUX} \quad (5.3)$$

In our approaches, the power models for each component are first developed, and then the switch power models are proposed based on Equation (5.2) and (5.3). In following subsections, the design tools that are used in the experiments are first discussed. The assumptions and settings for the experiments are presented next. Lastly, the modelling method is introduced.

5.2.1 Design Tools

In the experiments, two design tools are used:

(1) **Quartus II simulator:** a simulation tool to verify the correctness and behaviour of the designs. More importantly, it generates the signal activity data for the power analysis. (See FIGURE 4.16 for more details).

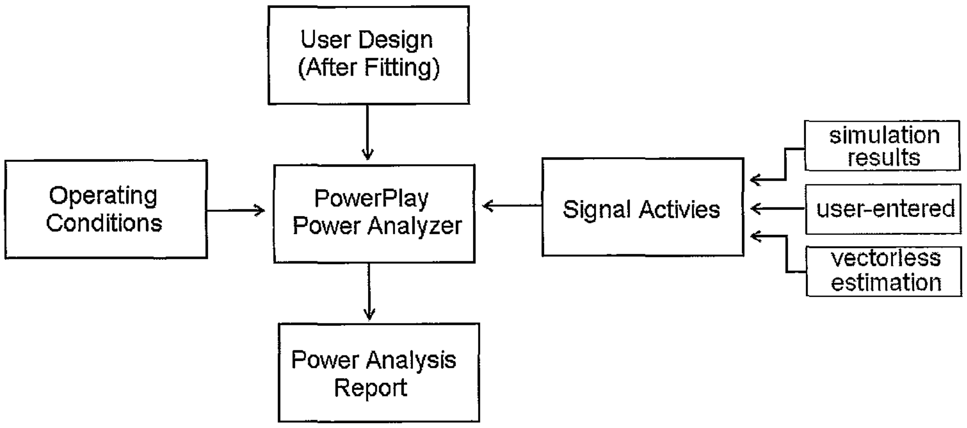


FIGURE 5.1 [57]

FIGURE 5.1: *PowerPlay* power analysis flow

(2) ***PowerPlay* Power Analyzer**: A commercial power analysis tool which was introduced in Section 2.6.3. The power analysis flow of the PowerPlay power

analyzer tool is given in FIGURE 5.1 and its power analysis result is accurate (to within 20%) of actual device power consumption under a typical design operation [57]. There are three ways to provide the information about the signal activities for the power estimation:

- Simulation results. During the simulations, the signal activity information is captured and saved as Signal Activity Files (*.saf* files). The Power Analyzer reads the simulation results generated by different simulators, such as Quartus II simulator, ModelSim, NC-VHDL and VCS. It provides the most accurate results when reading signal activities from simulations.
- User-entered. The designers can define the signal activities for some nodes given the lack of the activity information from the simulation results. For example, they can assign specific toggle rates and static probabilities to some nodes and entities in the design.
- Vectorless Estimation. When the simulation or user-entered signal-activity data is not available, the Power Analyzer estimates the signal activity automatically.

5.2.2 Assumptions

During switching, different packets travel on different data paths concurrently, and the traffic load on each data path may change dramatically from time to time. To estimate the dynamic power of the designs, the assumptions used in the experiments are:

(1) The incoming data is deterministic and uses a deterministic TDM scheduling.

(2) The switch operates with 100% throughput, i.e. every row in the switch receives one packet in every cell-time-slot on average, and every column in the switch removes one packet every cell-time-slot on average.

5.2.3 Experimental Setup

To gather experimental power results, the basic components created in VHDL are first synthesized and simulated using the Altera Quartus II simulator and Cycone II EP2C70F72C6 FPGA. Then, the Altera *PowerPlay* Power Analyzer is used to measure the power consumed by the designs based on the simulation results. The test input data is generated from the MATLAB. The measurement procedures are summarized in FIGURE 5.2.

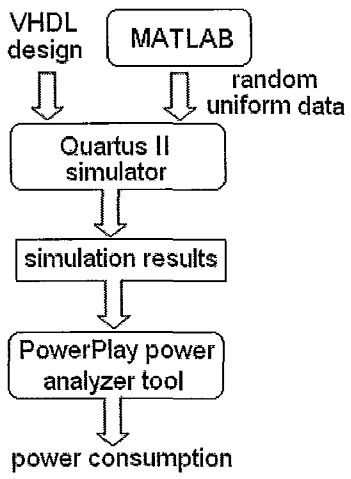


FIGURE 5.2: Power measurement flow

The settings of the simulator and Power Analyzer tool for the experiments are:

(1) Device: ***Cyclone II EP2C70F672C6*** in 90-nm process, 1.2V core voltage

(2) Analysis/Synthesis settings:
- Optimization Technique: Balanced (Speed/Area)

- *PowerPlay* power optimization: Normal compilation
- Fitter Effort: Standard Fit (highest effort)
- Simulation Mode: Timing

(3) *PowerPlay* Analyzer settings
- Generate Signal Activity file for power analysis
- Device power characteristics: Typical
- Operating Conditions:
- Ambient temperature: $25\,^0C$
- Use cooling solution: 23 mm heat sink with 200 LFpM airflow
- Junction-to-case: 2.2 $^0C/W$
- Case-to-heat sink: 0.10 $^0C/W$
- Heat sink-to-ambient: 2.80 $^0C/W$
- Board thermal model: Typical
- Junction-to-board: 3.80 $^0C/W$
- Board temperature: 25 ^0C
- Default toggle rate for unspecified signals: 12.5%

5.2.5 Modelling Method

The factors affecting the power consumption of the basic components are listed in Table 5.1. In the experiments, one of power factors will change and the other factors will be kept constant. The power caused by every change is measured by the *PowerPlay* tool after the simulation. The analytical models are derived based on the measurement results.

Table 5.1: Power Factors for the basic components

Components	Power Factors
DEMUX	number of outputs, datapath width (w), f_{clk}
MUX	number of inputs, w, f_{clk}
XQ	queue size, w, f_{clk}
BUS	switch size, w, f_{clk}

Note: the core voltage is fixed for devices. In the Cyclone II FPGAs, V_{dd} is typically $1.2V$.

5.3 Power Models for Basic Components

In this section, we present the power models for the basic components of the *CIXQ* switches following the experimental methodology in Section 5.2.

5.3.1 Demultiplexer Logic Block

A large *1-to-N* demultiplexer block can be constructed from multiple smaller *1-to-m* demultiplexers arranged in a radix-m tree topology with $\log_m(N)$ levels and $\frac{(N-1)}{(m-1)}$ basic cells. The power consumed by all standard cells of a *1-to-N* demultiplexer in FPGA can be described in a similar form suggested in [69]:

$$P(DEMUX1N) = \frac{(N-1)}{(m-1)} \cdot S_{DEMUX1m} \cdot K_{DEMUX1m} \cdot f_{clk} \cdot f_{duty} \cdot w \quad (5.4)$$

where w is the datapath width and $K_{DEMUX1m}$ is the power constant that reflects the average intrinsic and load capacitances switched for a *1-to-m* logic cell. In general, the power constant K increases with the complexity of the logical function being realized. As we described in Chapter 4, every logic function in an FPGA is configured through a series of look-up-tables (LUTs), programmable gates and interconnections within the "Logic Elements" (LEs). The number of FPGA LEs used in the *1-to-N* DEMUX block will depend upon the number of LEs require for each *1-to-m* cell after synthesis and optimization. The constant $S_{DEMUX1m}$ indicates the synthesis efficiency of the basic logic function being generated, i.e. $S_{DEMUX1m}$ is the average number of LEs used in the FPGA for the basic *1-to-m* logic cell after synthesis and optimization.

The large demultiplexer used in our designs is formed from *1-to-2* demultiplexer cells arranged in a binary tree (see FIGURE 5.3). Therefore, the power consumed by a large *1-to-N* demultiplexer based on a *1-to-2* logic cell, i.e. $m = 2$, can be described as

$$P(DEMUX1N) = \frac{(N-1)}{(m-1)} \cdot S_{DEMUX12} \cdot K_{DEMUX12} \cdot f_{clk} \cdot f_{duty} \cdot w \quad (5.5)$$

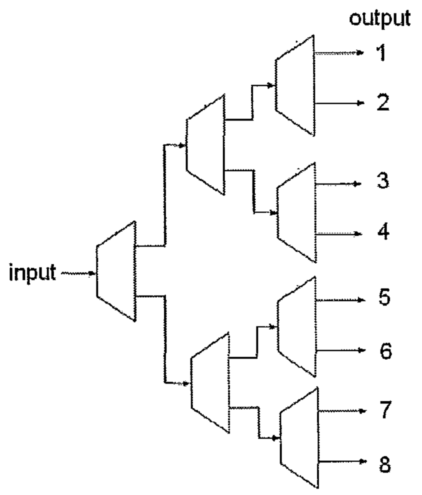


FIGURE 5.3: An example of *1-to-8* DEMUX

The number of LEs used in *1-to-N* DEMUX block can be estimated from the following equation:

$$N_{LE-DEMUX} = \frac{(N-1)}{(m-1)} \cdot S_{DEMUX12} \quad (5.6)$$

where $S_{DEMUX12}$ represents the synthesis efficiency of a *1-to-2* logic cell after synthesis and optimization, and $K_{DEMUX12}$ is the power constant for the *1-to-2* logic cell.

In our experiments, we measured the number of LEs and the power consumption needed for various DEMUXs using Altera Quartus simulator and the *PowerPlay*

Analyzer at a fixed clock rate of 50 MHz. The measurements are performed by changing the numbers of outputs of the DEMUXs from 4 to 512 and the datapath widths from 4 to 16 bits. The testing circuit is illustrated in FIGURE 5.4. According to the measurement results, $K_{DEMUX12} = 3.333 \times 10^{-5}$ mW/(MHz*LE), and $S_{DEMUX12} = 1$, i.e., each *1-to-2* DEMUX cell requires only one LE in the Altera Cyclone II FPGA family.

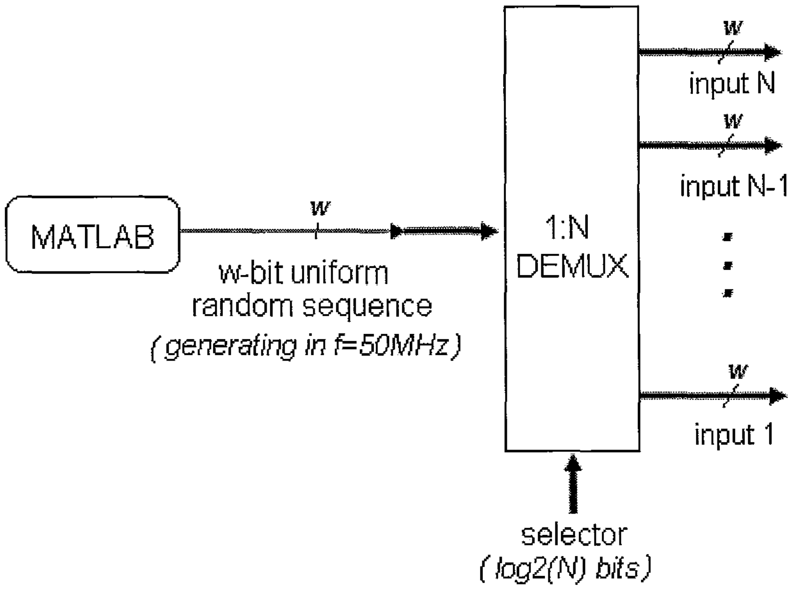


FIGURE 5.4: Testing circuit for DEMUX power

The number of LEs used in the synthesis of large *1-to-N* demultiplexer is shown in FIGURE 5.5 with three datapath widths: w= 4, 8 and 16 bits. The analytic results estimated from Equation (5.5) are shown by the solid lines. The experimental results for the number of LEs used reported by the Altera Quartus simulator are represented by the dashed lines, which lie under the solid lines and are not visible. The analytic results for the number of LEs required in a large logic block are essentially identical with the experimental results.

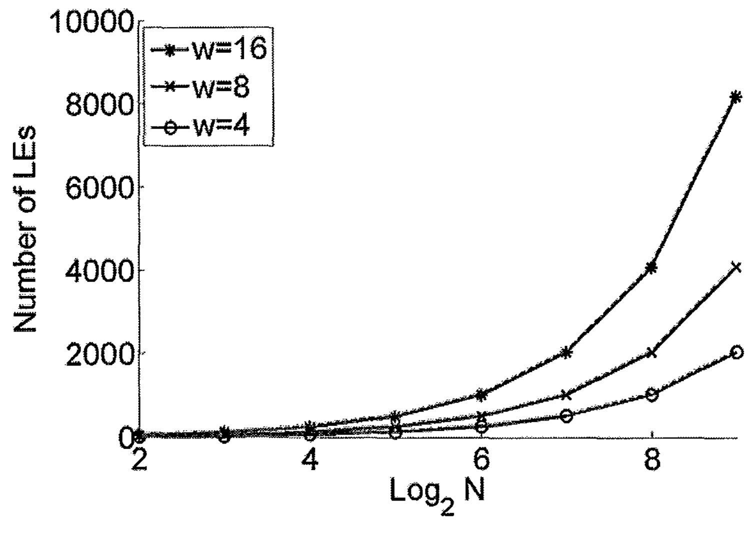


FIGURE 5.5: The number of LEs used in *1-to-N* DEMUX

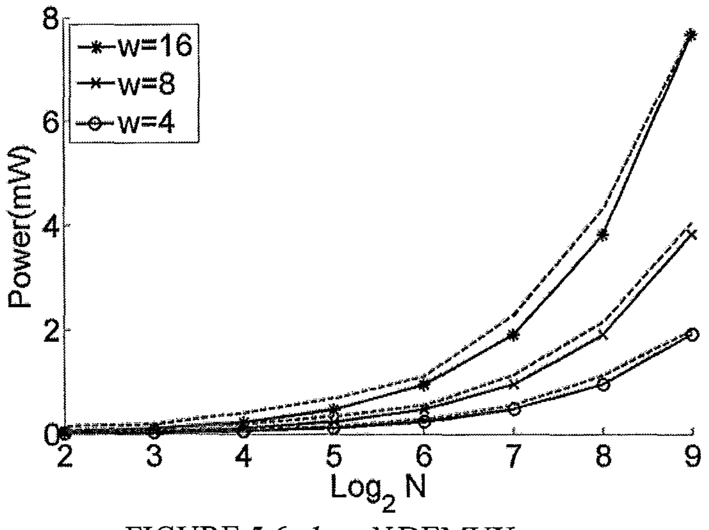


FIGURE 5.6: *1-to-N* DEMUX power

The power required by a *1-to-N* demultiplexer for datapath widths: w=4, 8 and 16 bits at a clock rate of 50 MHz is illustrated in FIGURE 5.6. The dotted lines are the experimental results reported by the *PowerPlay* tool, and the solid lines demonstrate the analytic results estimated from the model. The errors between the experimental results and analytic results are provided in Table 5.2, which shows that they have good agreement with average error less than 15%. The estimation error is defined as below.

$$Error = \frac{abs(estimated\_value - actual\_value)}{estimated\_value} \times 100\% \quad (5.7)$$

Table 5.2: Estimation Error in *1-to-N DEMUX* Power

logN	*w* = 4	*w* = 8	*w* = 16
2	7.35%	23.53%	15.81%
3	14.19%	5.61%	18.48%
4	23.70%	9.40%	4.63%
5	20.50%	9.90%	9.90%
6	17.53%	19.01%	19.01%
7	9.98%	7.53%	6.71%
8	2.78%	2.23%	2.00%
9	9.07%	10.99%	5.64%

5.3.2 Multiplexer Logic Block

With the same strategy, the large *N-to-1* multiplexer logic blocks in the designs are implemented by a binary tree of *2-to-1* multiplexer cells as illustrated in FIGURE 5.7.

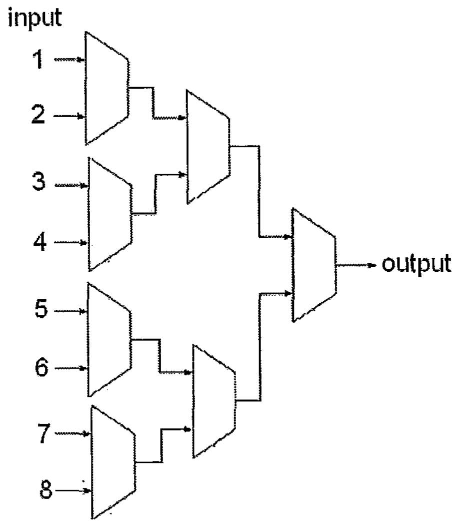


FIGURE 5.7: An example of *8-to-1* MUX

The power dissipated by a large *N-to-1* multiplexer logic block is evaluated in two cases: (a) *full power mode*, where all MUX cells are switched on at the same time during the operation and (b) *low power mode*, where the power required can be reduced considerably by disabling unnecessary switching, as established in [69]. In a crossbar switch application, each input port forwards at most one packet to one crosspoint in its row per time-slot. Therefore, in the low power mode, the switch only requires $\log_2(N)$ *2-to-1* multiplexer cells to be active during a time-slot. The circuit must be designed so that only the active input port of the large multiplexer undergoes a signal change, i.e. the data signals at each inactive input port must be held constant. Compared to the full power MUX, the low power MUX does not toggle or change the states of the unselected inputs so that it generates low power consumption. For example, given a *4-to-1* MUX, where Input 1 is selected as the output, then all logic cells are on in full power mode as shown in FIGURE 5.8 (a), and only two cells are active in low power mode as shown in FIGURE 5.8 (b).

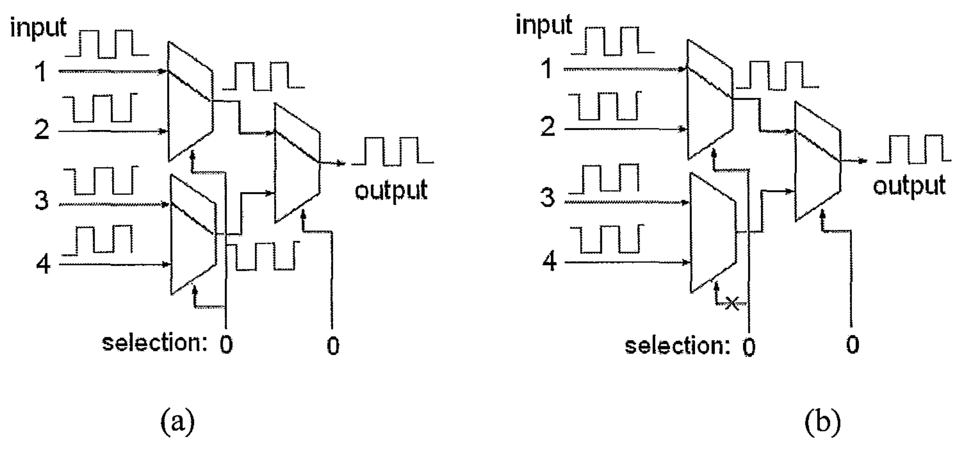


FIGURE 5.8: Power in *4-to-1* MUX: (a) full power mode; (b) low power mode.

In theory, the power of a large MUX is proportional to the number of active cells during the operation. A large MUX has $\frac{(N-1)}{(m-1)}$ cells in total when constructed from *m-to-1* cells. Therefore, in *the full power mode* the total power consumed by a multiplexer based on *2-to-1* cells, i.e. $m = 2$, is expressed as

$$P(MUXN1) = \frac{(N-1)}{(m-1)} \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \quad (5.8)$$

The number of LEs required in the MUX block can be described as

$$N_{LE-MUX} = \frac{(N-1)}{(m-1)} \cdot S_{MUX21} \quad (5.9)$$

where S_{MUX21} is the average number of LEs required in the synthesis of a *2-to-1* multiplexer cell, and K_{MUX21} represents the average intrinsic and load capacitances being switched for a *2-to-1* multiplexer after synthesis and optimization.

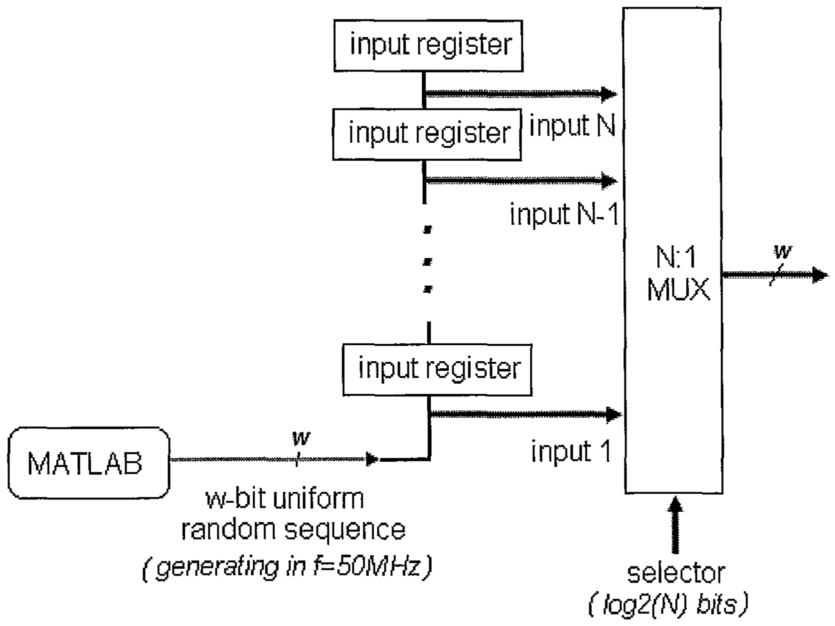


FIGURE 5.9: Testing circuit for MUX power

By changing the number of inputs in the multiplexer from 4 to 512 and datapath widths from 4 to 16 bits, the number of LEs and the power consumption at a clock rate of 50 MHz are measured using the testing circuit shown in FIGURE 5.9. According to the measurement results, we have $S_{MUX21} = 0.65863$ and $K_{MUX21} = 1.6631 \times 10^{-4}$ *mW/(MHz.LE)*.

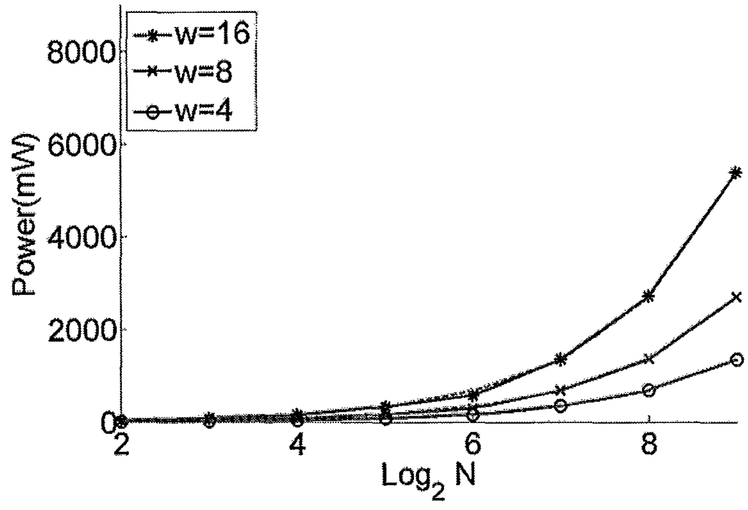


FIGURE 5.10: The number of LEs used in *N-to-1* MUX

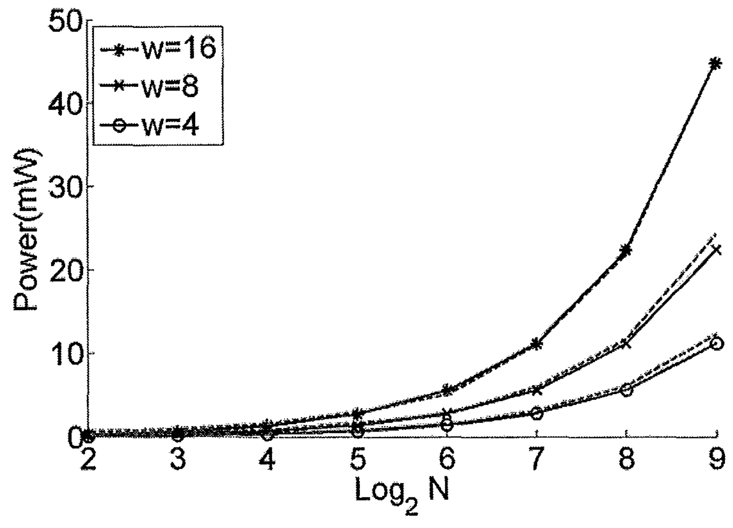


FIGURE 5.11: *N-to-1* MUX power in full power model

The number of LEs used in large *N-to-1* multiplexers is illustrated in FIGURE 5.10. The analytic results from Equation (5.9) represented by the solid lines are almost the same as the experimental results shown by the dash lines. The power consumption of an *N-to-1* multiplexer at a 50 MHz clock rate is given in FIGURE 5.11, where the analytic results are shown by the solid lines and the experimental results are shown by the dotted lines. They are very close to each other with an average error less than 0.5%.

In *low power mode*, a MUX has only $\log_2(N)$ cells active for the connection (see FIGURE 5.8 (b)), and the power dissipated by an *N-to-1* multiplexer is estimated by

$$P(MUXN1) = \log_2 N \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \quad (5.10)$$

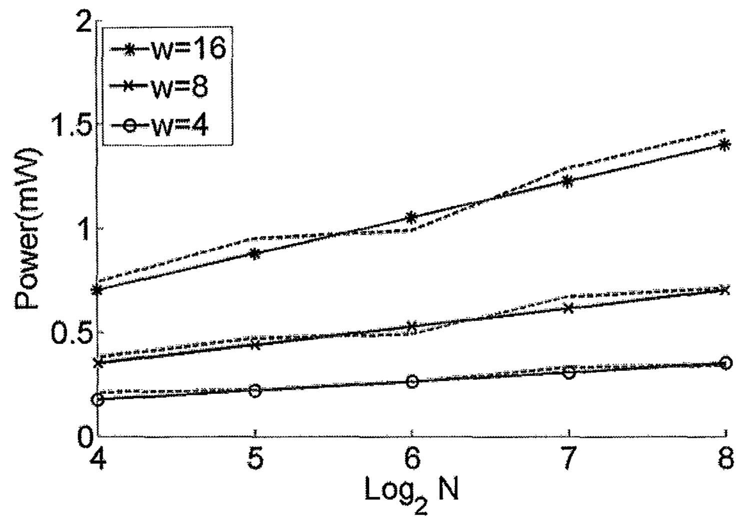


FIGURE 5.12: *N-to-1* MUX in low power model

The analytic power results for several large multiplexer blocks clocked at 50 MHz using the low-power design technique are depicted in FIGURE 5.12. The experimental results reported by the *PowerPlay* tool are presented in the dotted lines. The analytic results denoted by dash lines have a very good agreement to the experimental results. The average estimation error presented in Table 5.3 is about 10%. The low-power technique results in significant power reductions: for a large *256-to-1* multiplexer block using 16-bit datapath width, the full power version requires 21.90 *mW*, while the low-power version requires 1.47 *mW*, a reduction in power by a factor of about 15. Note that the number of LEs used in a large MUX is same in both the full power mode and low power mode.

Table 5.3: Estimation Error of *N-to-1 MUX* Power in Low Power Mode

N	logN	w=4	w=8	w=16
4	2	12.45%	12.45%	14.01%
8	3	2.48%	2.48%	1.79%
16	4	2.90%	6.90%	9.35%
32	5	7.21%	0.89%	0.17%
64	6	5.62%	11.06%	10.16%
128	7	3.11%	4.67%	0.76%
256	8	8.57%	4.53%	1.17%
512	9	4.16%	8.79%	15.15%

5.3.3 Crosspoint Queue Logic Blocks

A *CIXQ* switch has N^2 FIFO queues in the switch. The power dissipated by a FIFO logic block with a capacity of N w-bit words implemented in an FPGA technology can be represented by

$$P(FIFONw) = \frac{1}{2} f_{clk} \cdot f_{duty} \cdot (C_{\text{int}} + C_{load}) \cdot (V_{dd})^2 \quad (5.11)$$

where f_{clk} is the clock rate, f_{duty} is the probability of signal transitions at the output port, C_{int} reflects the intrinsic capacitance of one FIFO logic block in the FPGA, and C_{load} is the capacitive load of the wires being driven by the logic block.

The intrinsic capacitance of the FIFO block depends upon the state of the block. Four distinct states have been identified: *(IDLE = I, READ = R, WRITE = W, READ-WRITE = RW)*. In an *NxN CIXQ* switch, the long-term average power of each FIFO block will be a weighted average of the power dissipated in each state of the FIFO block. To develop an analytic power model, the long-term state probabilities must be defined, and these probabilities will depend upon the traffic routed through the switch. We assume the switch operates with 100% throughput with a random and uniformly distributed traffic pattern, i.e. every row in the switch receives one cell in every cell-time-slot on average, and every column in the switch removes one cell every cell-time-slot on average. Equivalently, *N* packets move through the switch per cell -time-slot, achieving 100% throughput. To achieve 100% for our simulations, a FIFO queue size of 5 cells at each input port was used for all switches.

The steady-state rate of cell insertions or deletions through any one FIFO queue is therefore *1/N* cell per cell-time-slot. In the steady state, the probabilities of each state of a FIFO queue (*I,R,W,RW*) are given by: *P(I) = (1-1/N)*(1-1/N), P(R) = (1/N)*(1-1/N), P(W) = (1/N)*(1-1/N), and P(RW)=(1/N)*(1/N)*. For example, for a *4x4* switch with random uniform traffic, the state probabilities are given by {0.5625, 0.1875, 0.1875, and 0.0625}.

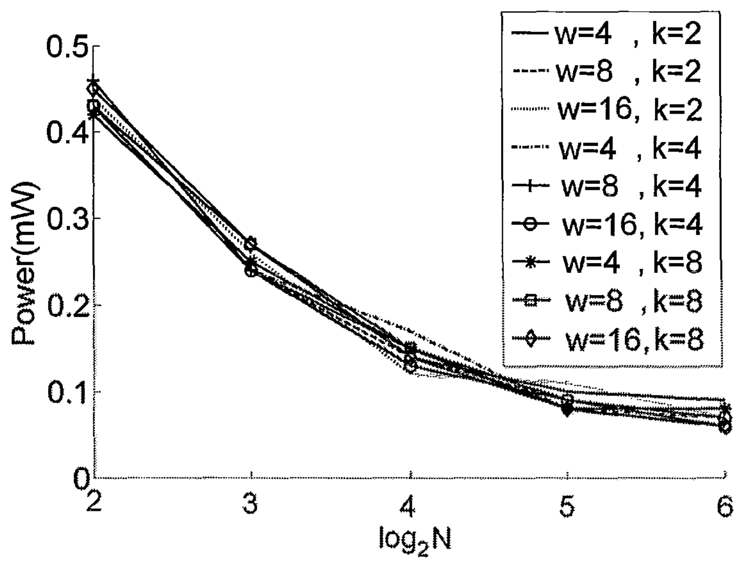


FIGURE 5.13: Power consumption of a single crosspoint queue in *NxN CIXQ*-k switches with various datapath widths

The state of each FIFO queue may also depend upon the queue capacity and datapath width. As described previously, in the steady state of a switch with 100% throughput, each FIFO queue can perform one of the four possible operations per time slot: (a) write at most one cell; (b) read at most one cell; (c) write and read one cell at the same time; and (d) remain idle. Our models consider the dynamic power during switching. The FIFO queue dissipates dynamic power only in writing or reading operation. The amount of power drawn by these operations depends on the number of cells being moved in to or out from the queue. In conclusion, at most one cell with a fixed size of 64 bytes can be inserted to or deleted from the FIFO queue per time slot. We have observed that the FIFO queue consumes the same amount of power during switching, independent of queue size and datapath width. However, a larger queue increases the static power of the design. The experimental results shown in FIGURE 5.13 have demonstrated that the dynamic power of FIFO queues is relatively independent of queue capacity (*k*), or datapath width (*w*).

Given the deterministic TDM scheme, the power consumed by a FIFO queue can be accurately modelled based on its duty cycle. At a 50 MHz clock rate, each FIFO at 100% and 0% duty cycle consumes about 1.72 *mW* and 0.05 *mW* respectively. For intermediate duty cycles, the FIFO power is proportional to the duty cycle. In our *CIXQ* switch applications, the load or duty cycle equals *1/N* since each FIFO is active on average roughly one cell-time-slot out of *N*. Thus, the analytic power model for a single FIFO queue inside the switch fabric of CXIQ switches can de described by

$$P(FIFO) = K_{FIFO} \cdot f_{clk} \cdot (f_{duty} = 1/N) \quad (5.12)$$

where K_{FIFO} reflects the intrinsic and load capacitances switched by the FIFO during a signal transition, and the measurements indicate that $K_{FIFO} = 39.85 \times 10^{-3} mW / MHz$.

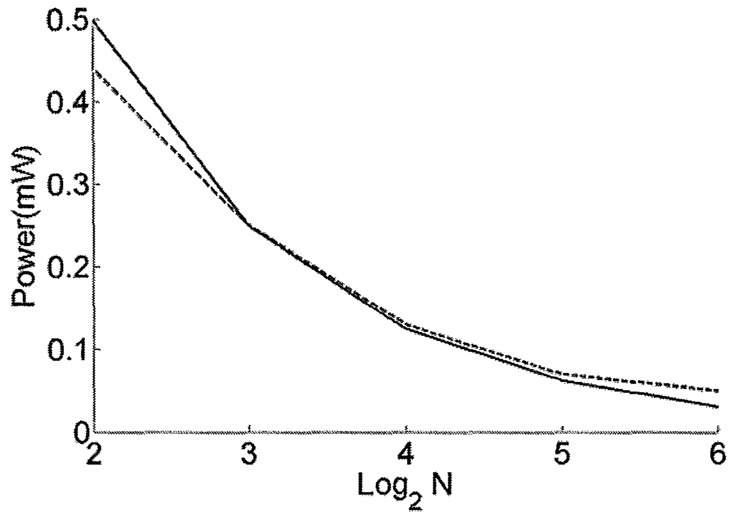


FIGURE 5.14: Power consumption of a single crosspoint queue in *NxN* switches

FIGURE 5.14 plots the average power for one FIFO logic block, for various switch sizes *N*. Regardless of the FIFO queue capacity and datapath widths, the analytic results represented in a solid lines have a good agreement with the experimental results shown by the dash lines. From the power plot, we can see as the switch size *N* grows, each FIFO block power drops because the duty cycle is decreased. Therefore, the FIFO queue power is inversely proportional to the switch size. As shown in Table 5.4, the average error between the experimental results and analytic results is less than 10%.

Table 5.4: Estimation Error of a *FIFO* Power in *NxN* Switches

N	logN	Error
4	2	11.24%
8	3	2.22%
16	4	7.93%
32	5	9.10%
64	6	17.59%

5.3.4 Broadcast Busses

A broadcast-and-select design uses *N* busses to connect the input ports and crosspoint queues instead of the demultiplexers used in the DEMUX-MUX design. To model the bus power, the *CIXQ* switches based on the broadcast-and-select design with sizes of *4x4*, *8x8* and *16x16* are first implemented in FPGA, and then tested and simulated using Quartus II simulator. After the switches are proved to be functional, the total power of the switches is measured using the *PowerPlay* tool. Then we estimate the power consumption of the MUX and crosspoint queue from the power models developed in previous sections. According to the broadcast-and-select architecture, the broadcast bus power in *NxN CIXQ* switch can be calculated by the following equation:

$$P(BUS) = \frac{P(XQ) - N^2 \cdot P(FIFO) - N \cdot P(MUXN1)}{N} \quad (5.13)$$

Based on the measurement results, the power for a bus with *w*-bit wide spanning N columns can be described as

$$P(Bus1N) = N \log_2 N \cdot K_{Bus1N} \cdot f_{clk} \cdot f_{duty} \cdot w \quad (5.14)$$

where K_{Bus1N} reflects the intrinsic and load capacitances switched by the bus during a signal transition, and the power measurements indicate that $K_{Bus1N} = 2.7350 \times 10^{-4}$ mW/MHz. The bus power model is consistent with theory as it increases logarithmically with the fan-out of the bus. FIGURE 5.15 shows the bus power consumed in a *CIXQ* switch at 50 MHz clock rate with various datapath widths. The analytic power results have an excellent agreement with the experimental results for *4x4, 8x8* and *16x16* switches shown as the solid points.

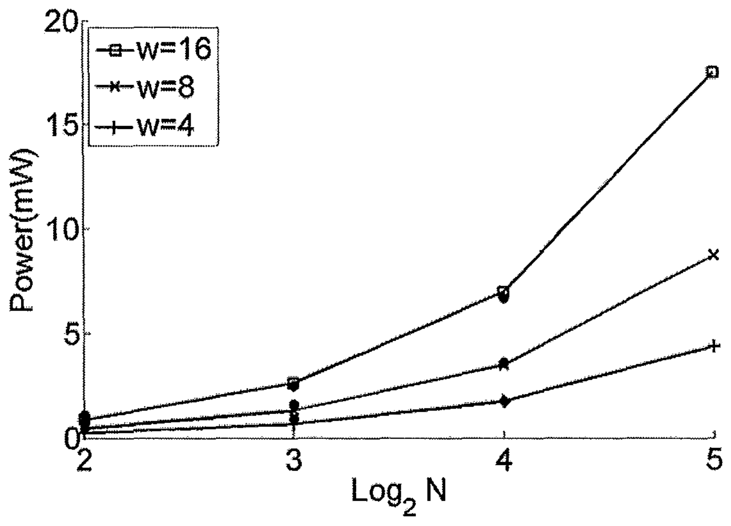


FIGURE 5.15: A broadcast bus power in *NxN* switches

5.4 Power Models for *IQ* and *CIXQ* Switches

Finally, the total power dissipated by *CIXQ* switches can be estimated from the power models developed in section 5.3 according to the relationships described in Equation (5.2) and (5.3). For comparison purpose, we also propose the power models for *IQ* switches, which simply subtract the power consumed by the crosspoint queues. The I/O power is not considered in the power models. But the I/O power is easily computed given the number of I/O pins used and data rate per port.

5.4.1 Broadcast-and-Select Design

In the broadcast-and-select design, the basic components of an *NxN CIXQ* switch are N broadcast busses, N^2 crosspoint queues and N multiplexers. Regarding the power minimization techniques used in the multiplexers, the power of *IQ* and *CIXQ* switches in the full power mode can be described as

$$\begin{aligned} P(IQ) &= N \cdot P(Bus1N) + N \cdot P(MUXN1) \\ &= N^2 \log_2 N \cdot K_{Bus1N} \cdot f_{clk} \cdot f_{duty} \cdot w \\ &\quad + N \cdot \frac{(N-1)}{(m-1)} \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \end{aligned} \qquad (5.15)$$

$$\begin{aligned} P(XQ) &= N \cdot P(Bus1N) + N^2 \cdot P(FIFO) + N \cdot P(MUXN1) \\ &= N^2 \log_2 N \cdot K_{Bus1N} \cdot f_{clk} \cdot f_{duty} \cdot w + N^2 \cdot K_{FIFO} \cdot f_{clk} \cdot (f_{duty} = 1/N) \\ &\quad + N \cdot \frac{(N-1)}{(m-1)} \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \end{aligned} \qquad (5.16)$$

Using the low-power multiplexer design, the power consumed by the switches can be expressed as

$$\begin{aligned} P(IQ) &= N \cdot P(Bus1N) + N \cdot P(MUXN1) \\ &= N^2 \log_2 N \cdot K_{Bus1N} \cdot f_{clk} \cdot f_{duty} \cdot w \\ &\quad + N \cdot \log_2(N) \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \end{aligned} \qquad (5.17)$$

$$\begin{aligned} P(XQ) &= N \cdot P(Bus1N) + N^2 \cdot P(FIFO) + N \cdot P(MUXN1) \\ &= N^2 \log_2 N \cdot K_{Bus1N} \cdot f_{clk} \cdot f_{duty} \cdot w + N^2 \cdot K_{FIFO} \cdot f_{clk} \cdot (f_{duty} = 1/N) \\ &\quad + N \cdot \log_2(N) \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \end{aligned} \qquad (5.18)$$

5.4.2 DEMUX-MUX Design

In DEMUX-MUX design, an *NxN CIXQ* switch is formed from *N* demultiplexers, N^2 crosspoint queues and *N* multiplexers. Without using the low power design technique in the multiplexers, the power of *IQ* and *CIXQ* switches can be described as

$$\begin{aligned} P(IQ) &= N \cdot P(DEMUX1N) + N \cdot P(MUXN1) \\ &= N \cdot \frac{(N-1)}{(m-1)} \cdot S_{DEMUX12} \cdot K_{DEMUX12} \cdot f_{clk} \cdot f_{duty} \cdot w \\ &\quad + N \cdot \frac{(N-1)}{(m-1)} \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \end{aligned} \qquad (5.19)$$

$$\begin{aligned} P(XQ) &= N \cdot P(DEMUX1N) + N^2 \cdot P(FIFO) + N \cdot P(MUXN1) \\ &= N \cdot \frac{(N-1)}{(m-1)} \cdot S_{DEMUX12} \cdot K_{DEMUX12} \cdot f_{clk} \cdot f_{duty} \cdot w + N^2 \cdot K_{FIFO} \cdot f_{clk} \cdot (f_{duty} = 1/N) \\ &\quad + N \cdot \frac{(N-1)}{(m-1)} \cdot S_{MUX21} \cdot K_{MUX21} \cdot f_{clk} \cdot f_{duty} \cdot w \end{aligned} \qquad (5.20)$$

The power consumed by the switches with low-power multiplexers can be expressed as

$$\begin{aligned}P(XQ) &= N\cdot P(DEMUX1N)+N\cdot P(MUXN1)\\ &= N\cdot\frac{(N-1)}{(m-1)}\cdot S_{DEMUX12}\cdot K_{DEMUX12}\cdot f_{clk}\cdot f_{duty}\cdot w\\ &\quad +N\cdot\log_2(N)\cdot S_{MUX21}\cdot K_{MUX21}\cdot f_{clk}\cdot f_{duty}\cdot w \qquad (5.21)\end{aligned}$$

$$\begin{aligned}P(XQ) &= N\cdot P(DEMUX1N)+N^2\cdot P(FIFO)+N\cdot P(MUXN1)\\ &= N\cdot\frac{(N-1)}{(m-1)}\cdot S_{DEMUX12}\cdot K_{DEMUX12}\cdot f_{clk}\cdot f_{duty}\cdot w+N^2\cdot K_{FIFO}\cdot f_{clk}\cdot(f_{duty}=1/N)\\ &\quad +N\cdot\log_2(N)\cdot S_{MUX21}\cdot K_{MUX21}\cdot f_{clk}\cdot f_{duty}\cdot w \qquad (5.22)\end{aligned}$$

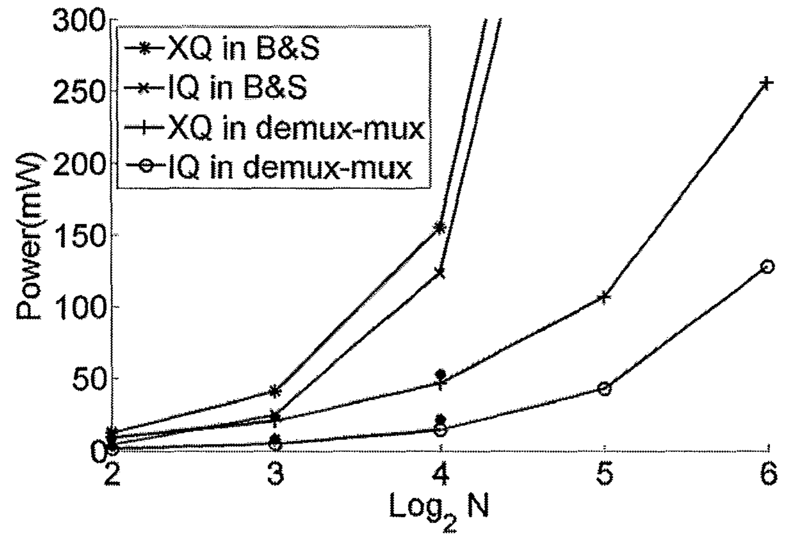


FIGURE 5.16: Power consumption of *IQ* and *CIXQ* switches

5.5 Verification and Discussion

In order to validate the power models, we implement the *IQ* and *CIXQ* switches with sizes of *4x4, 8x8* and *16x16* in VHDL, following the design procedures in Chapter 4. The switches are synthesized, optimized and fully tested by the Altera Quartus II simulator with 100% throughput, which is achieved using the routing mechanism proposed in Section 4.8. The simulations are performed with random uniform traffic that is generated by MATLAB until a steady-state is reached, and the power is reported by *PowerPlay* Analyze*r*.

The experimental results and the analytic results of power consumed by *IQ* and *CIXQ* switches with the low power multiplexers are shown in FIGURE 5.16 and Table 5.5, assuming a 50 MHz clock rate and a datapath width of 16 bits, i.e., *w=16 bits*. In FIGURE 5.16, the experimental power results for the *IQ* and *CIXQ* switches using the DEMUX-MUX design are shown as the bold bots, and the solid lines represent the analytic power results. The experimental results show excellent agreement with the analytic results as the bold dots are nearly superimposed upon the analytic results.

Table 5.5: Power comparison between *IQ* and *CIXQ* switches

Switch Design	Switch Size (*N*)	Switch Type	Analytic Result (*mW*)	Experiment Result (*mW*)	Error (%)
Broadcast and Select	4	*IQ*	4.90	5.42	9.59
		CIXQ	12.87	12.17	5.75
	8	*IQ*	25.21	25.11	0.40
		CIXQ	41.15	37.97	8.38
	16	*IQ*	123.24	114.7	7.45
		CIXQ	155.12	168.22	7.79
DEMUX and MUX	4	*IQ*	2.93	3.38	13.38
		CIXQ	9.24	9.27	0.32
	8	*IQ*	7.44	8.37	11.16
		CIXQ	23.30	23.69	1.66
	16	*IQ*	19.53	21.82	10.47
		CIXQ	59.39	52.74	12.60

As listed in Table 5.5, the analytic models are shown to be quite accurate as the average error between the analytic power and actual power is less than 10%. In a broadcast-and-select design, the broadcast busses dominate the power dissipation with $O(N^2 \log_2(N))$ power. In a DEMUX-MUX design, the crosspoint queues dominate the power dissipation with $O(N^2)$ power. The use of crosspoint queues has a huge impact on the power consumption of the *CIXQ* switches. The power required by the *CIXQ* switch is 3 times more than that consumed by the *IQ* switch for these switch size of *4x4, 8x8 and 16x16*. As shown in Table 5.1, for the switch sizes of *4x4, 8x8* and *16x16*, the power consumption of *CIXQ* switches is 9.3, 23.7 and 52.7 mW while the *IQ* switch requires 3.4, 8.4 and 21.8 mW, respectively. Furthermore, DEMUX-MUX architecture is a better design choice for switch architectures because it has higher power efficiency compare to the broad-and-select design.

Chapter 6 Conclusions

The Combined Input and Crosspoint Queued (*CIXQ*) switches with a FIFO queue in each crosspoint have recently been proposed as an alternative to Input Queued (*IQ*) switches. Although the *CIXQ* switches can simplify the scheduling, the employment of N^2 FIFO queues in the crossbar increases the design cost and hardware complexity. This thesis has studied the capability and flexibility of the *CIXQ* switches using current design methods in terms of performance, design implementation and power consumption. The primary objective is to develop a general analytic methodology for the power analysis of the *CIXQ* switches.

This chapter summarizes the results of this thesis and provides a conclusion regarding the current research areas of the *CIXQ* switches as well as the future research directions. The contributions, as described in the Introduction (Chapter 1), are repeated as follows.

(1) A preliminary performance comparison between the *CIXQ* and *IQ* switches is made in MATLAB.

Chapter 3 evaluates the switching performance of the *CIXQ* switch, where a simple Round Robin arbitration is used at the inputs and outputs. An *IQ* switch using the iSLIP scheduler is also implemented for comparison purpose. These two schedulers represent the simplest schedulers for each switch. The experimental results in Chapter 3 show that a *32x32* *CIXQ* switch with RR/RR scheduling has marginally better throughput than an *IQ* switch with iSLIP scheduling (with 4 iterations) under 2 non-uniform traffic models (unbalanced and log-diagonal). However, we cannot draw any conclusions on which switch has better performance without analyzing with more traffic patterns, other scheduling algorithms and other switch sizes. It is known that both switches can achieve 100% throughput, so peak performance will be comparable in both switches.

(2) *CIXQ* switches based upon various structures are designed and implemented in an FPGA technology.

Chapter 4 describes the implementation of the CX*IQ* switch based on the broadcast-and-select structure and the DEMUX-MUX structure. The switches are created in VHDL with a modular design, and the basic modules are the input ports, arbiters and crosspoint queues. Each module and the complete switch designs are synthesized and simulated using Altera design tools for Cyclone II FPGA devices. The simulation results indicate that the switch is functioning properly.

(3) As the major contribution, the first power models for *CIXQ* switches are developed.

Chapter 5 proposes a number of power models for the *CIXQ* and *IQ* switches. The power models are accurate with an average error of about 10%. As power becomes a first-class architectural design constraint [3], the power models can help the designers to explore the power-performance trade-offs in the early stages of the design process, thus avoiding the costly redesign process to fix the high power problem.

6.1 Discussion

This section presents a discussion regarding the simulation and experimental results about the switching performance, design implementation and power consumption of *CIXQ* switches.

6.1.1 Performance

The performance study illustrates that the *CIXQ*-k (where k is the crosspoint queue size) switch with RR arbitration marginally outperforms the *IQ* switch with iSLIP scheduling under non-uniform traffic patterns. Compared to the *IQ* switch, the *CIXQ* switch with one-cell crosspoint queue (*CIXQ*-1) increases the throughput from 80% to 84% for the unbalanced traffic and improves the throughput from 63% to 68% for the log-diagonal traffic. Furthermore, a larger crosspoint queue leads to a larger improvement in the throughput. When the crosspoint queue is the same as the switch size ($N = k$), the throughput can be improved to 98% for unbalanced traffic and about 70% for the log-diagonal traffic (see FIGURE 3.5), however this has a cost of $O(N^3)$ cell buffers. In general, the *CIXQ* switch requires $O(N^2)$ FIFO queues, which complicates the hardware design and increases power consumption. Although the performance comparison indicates that the *CIXQ* switch achieves marginally higher performance with comparable-complexity scheduling compared to the *IQ* switch given two simple scheduling algorithms (RR and iSLIP), there exist other practical algorithms which can improve the throughput of both *IQ* and *CIXQ* switch to about 100%. If the complexity of 100% throughput schedulers is comparable, the *CIXQ* switch would offer no improvement compared to the *IQ* switch and would require more complex hardware and power. Therefore, a determination of which switch (*CIXQ*-k or *IQ* switch) achieves better performance would require a thorough study of scheduling algorithms, which is beyond the scope of this thesis.

6.1.2 Implementation

A *CIXQ* switch with a FIFO queue at each input port is designed and implemented in a Cyclone II FPGA device using the VHDL language. Two design architectures are employed in the implementations of the *CIXQ* switch: the DEMUX-MUX and broadcast-and-select structures. Quartus II design tools are used to synthesize and simulate the designs, and the simulation results indicate they function correctly as described in this thesis.

In the industry, the input and output ports of a switch are typically implemented on several line-cards and the switching crossbar is realized on a separate VLSI substrate (refer to Section 2.4). With this design method, if we consider the implementation of a buffered crossbar in Cyclone II FPGAs with up to 1,152,000 bits of on-chip memory and up to 622 I/O pins (see Table 4.2), it is feasible to make a *CIXQ* switch up to 32 ports with a maximum crosspoint queue of 2 cells, assuming 8-bit datapath, 50MHz clock rate and fixed-size cell of 64 bytes. In other words, a *32x32 CIXQ-2* switch with *12.8Gbps* aggregate line rate is supported in Cyclone II FPGAs. Table 6.1 illustrates the possible implementations of *NxN CIXQ*-k switch in Cylone II FPGAs using above design method.

With the decreasing feature size of CMOS technology, larger amounts of I/O pins and memory resources should be available for future FPGA architectures. For example, Altera's Stratix III FPGAs provide the embedded memory of up to several Megabits and thousands of I/O pins, as it is described in Table 6.2 [78]. Using the same design features as above, Stratix III FPGAs have sufficient memory and I/O pins for a *CIXQ* switch with up to 128 ports with a maximum crosspoint queue of one cell (see Table 6.3).

Table 6.1: Possible implementations of *NxN CIXQ*-k switch in Cyclone II FPGAs

N	k_{max}
4	140
8	35
16	8
32	2

Table 6.2: Device Resources of Stratix III FPGAs

Device	Total Logic Elements (LEs)	Embedded Multipliers 9-bit Elements	Total pins	Total memory Kbits
EP3SL200	198,900	1,250	864	9,396
EP3SE260	254,400	3,180	960	14,688
EP3SL340	338,000	4,225	1,104	16,272

Table 6.3: Possible implementations of *NxN CIXQ*-k switch in Stratix III FPGAs

N	k_{max}
4	1986
8	496
16	124
32	31
64	7
128	1

6.1.3 Power Dissipation

As power has become a critical design concern, designers must consider the power in the early design stage when they make architectural trade-offs. Therefore, analytical power models are proposed for the *IQ* and *CIXQ* switch in this thesis. The verification shows the power models are accurate with an average error of about 10%. The power analysis indicates that the *CIXQ* switch with N^2 crosspoint queues requires about three times as much power as the *IQ* switch for modest switch sizes (*32x32*) in an FPGA enviroment. Furthermore, the DEMUX-MUX design is a better architecture choice because of the higher power efficiency compared to the broad-and-select design.

The power models do not account for the I/O power. However, this power is linearly related to the number of I/O pins and the data rate per port. A power evaluation of the *CIXQ*-k switch using the power models is descried through an example, in which the implementation of the *CIXQ*-k switch is realized using the Stratix III EP3SL340 FPGA. Table 6.4 lists the DC and switching characteristics of the device [79].

Table 6.4: DC and switching characteristic of Statrix III EP3SL340

Symbol	Description	Value
θ_{JA}	Thermal resistance	8.1 $^oC/W$
T_J	Maximum junction temperature	125 ^oC
T_A	Ambient temperature	40 ^oC
I_{OH}	Current strength at High transition	4 mA
I_{OL}	Current strength at Low transition	2 mA
V_{CC}	Core voltage	1.2 V
$R_{\max}$	Maximum line rate at each port	1.25 $Gpbs$

The Stratix III EP3SL340 device provides up to 132 full duplex $1.25Gpbs$ Low-Voltage Differential Signalling (LVDS) channels (132 TX + 132 RX), so it is possible to build a *16x16 CIXQ*-k switch with 8-bit datapath and a clock rate of 250 MHz ($20Gpbs$ as the aggregated line rate, R_{total}). The DEMUX-MUX design is used for the sample calculation. The steps to estimate the power, as described in Section 2.6.3, are shown as follows. The calculation results indicate that it is feasible to implement a *16x16 CIXQ-124* switch with the $20Gpbs$ aggregate line rate in the Stratix III EP3SL340 device.

Table 6.5: Sample Power Evaluation for an implementation of 16x16 *CIXQ*-k switch in Statrix III EP3SL340 device

Internal Power Calculation (P_{INT})	***CIXQ* switch**	***IQ* switch**
From power models: Eq. 5.20	$P_{INT} = 148.475\ mW$	$P_{INT} = 48.825\ mW$
I/O Power Calculation		
Average duty cycle (f_{duty}) (typically 0.50)	$f_{duty} = 0.50$	
Average current at each input or output port (I_{per_IO})	$I_{port} = 3.0\ mA$	
$I_{port} = (I_{OH} + I_{OL}) \cdot f_{duty}$		
Average power at each port per Gbps (P_{port})		
$P_{port} = (I_{port} \cdot V_{CC}) / R_{\max}$	$P_{port} = 2.88\ mW / Gpbs$	
Total I/O power (P_{IO})		
$P_{IO} = 2 \cdot P_{port} \cdot R_{total}$	$P_{IO} = 115.2\ mW$	$P_{IO} = 115.2\ mW$
Total power calculation		
Estimated total power (P_{EST})		
$P_{EST} = P_{INT} + P_{IO}$	$P_{EST} = 263.675\ mW$	$P_{EST} = 164.025\ mW$
Calculating maximum allowed power		
Maximum power (P_{MAX}) allowed for the device: $P_{MAX} = \frac{(T_J - T_A)}{\theta_{JA}}$	$P_{MAX} = 10.49\ W$	$P_{MAX} = 10.49\ W$
Comparing Maximum Power Allowed & Estimated power Is $P_{EST} < P_{MAX}$?	Yes	Yes

6.2 Future Work

Power has become a first-class architectural design constraint [3], especially when the allowable maximum power in the semiconductor devices no longer obeys Moore's law [80], as it is illustrated in FIGURE 6.1, which describes the projection of maximum power for semiconductor devices from the International Technology Roadmap for Semiconductors (ITRS) [81]. Instead of doubling every two years (Moore's law), the allowable maximum power is expected to remain constant for the next five years. Therefore, the semiconductor designers must ensure that the power dissipation in the devices does not exceed the maximum allowed power.

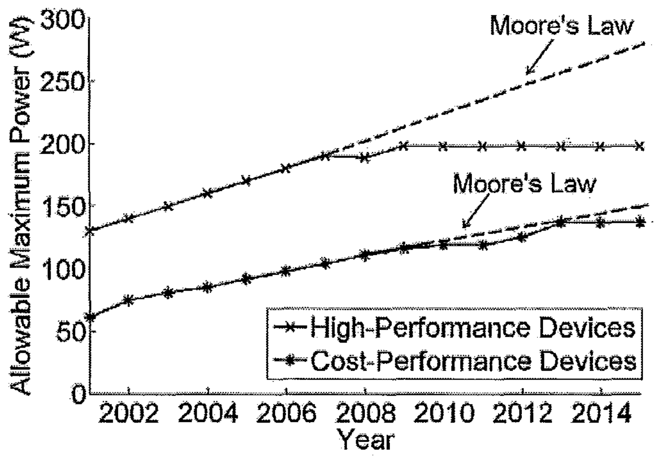


FIGURE 6.1: Maximum power trends of 2 types of semiconductor devices: (1) high-performance devices, for which a heat sink on the package is permitted, such as desktop devices; and (2) cost-performance devices, where only economical power management systems are provided, such as laptop devices.

As a result, it is crucial to have power tools to analyze the power of all switches, which are implemented using the CMOS technology. However, the power models proposed in this thesis have the following characteristics:

(1) Only power dissipation in the crossbar is considered: a real system needs VOQs and I/O line cards;

(2) Only uniform traffic is considered with two simple schedulers (RR and iSLIP).

In response to these limitations, the possible future research directions for the purpose of analyzing the power of *CIXQ* switches can be summarized as follows:

(1) Develop power models accounting for the I/O ports as well as other components of the switch design, such as the schedulers and line cards.

(2) Discuss the effect of different traffic patterns on the power consumption.

On the other hand, despite the fact that the *CIXQ* switch with RR arbitration achieves marginally better performance than the *IQ* switch with iSLIP scheduler under non-uniform traffic, it consumes three times as much power. In order to answer the question which switch design is better, a thorough study on the performance of the *CIXQ* and *IQ* switches with more sophisticated scheduling algorithms compared to these simple algorithms (Round Robin and iSLIP) is required, suggesting another direction for the future work.

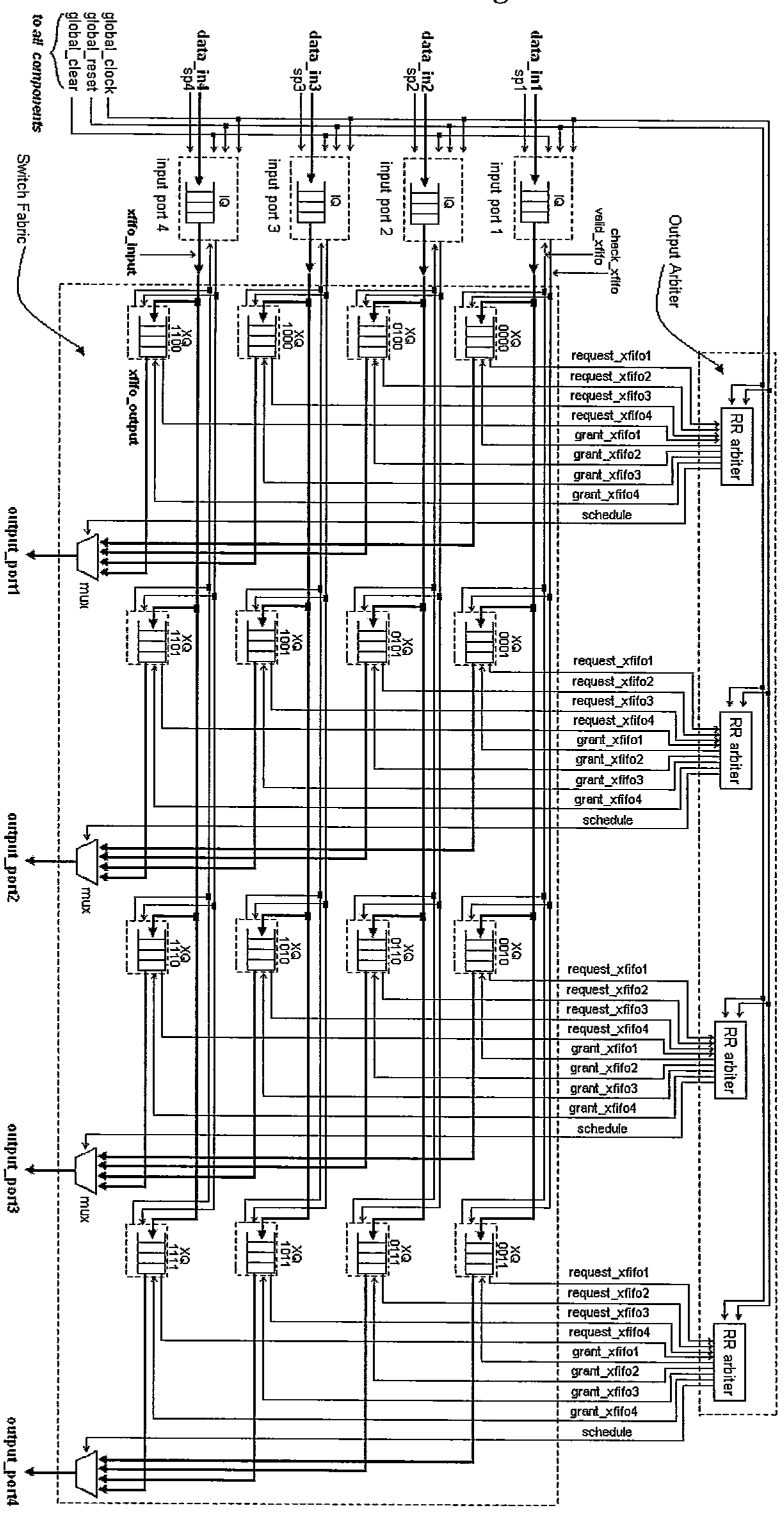
6.3 Final Conclusion

This thesis reviews the current research areas of *CIXQ* switches in the literature, which are the switching performance, design implementation and power dissipation. The performance of the *CIXQ* switch with RR arbitration is studied in a comparison with the *IQ* switch with iSLIP scheduler. In order to analyze the power consumption, the *CIXQ* switches based upon the broadcast-and-select design and DEMUX-MUX design are designed and implemented in an FPGA technology. As the major contribution, this thesis proposes the first power models for the *CIXQ* switches. The power models are reasonably accurate and allow a designer to explore the power-performance tradeoffs in the early stages of the design process.

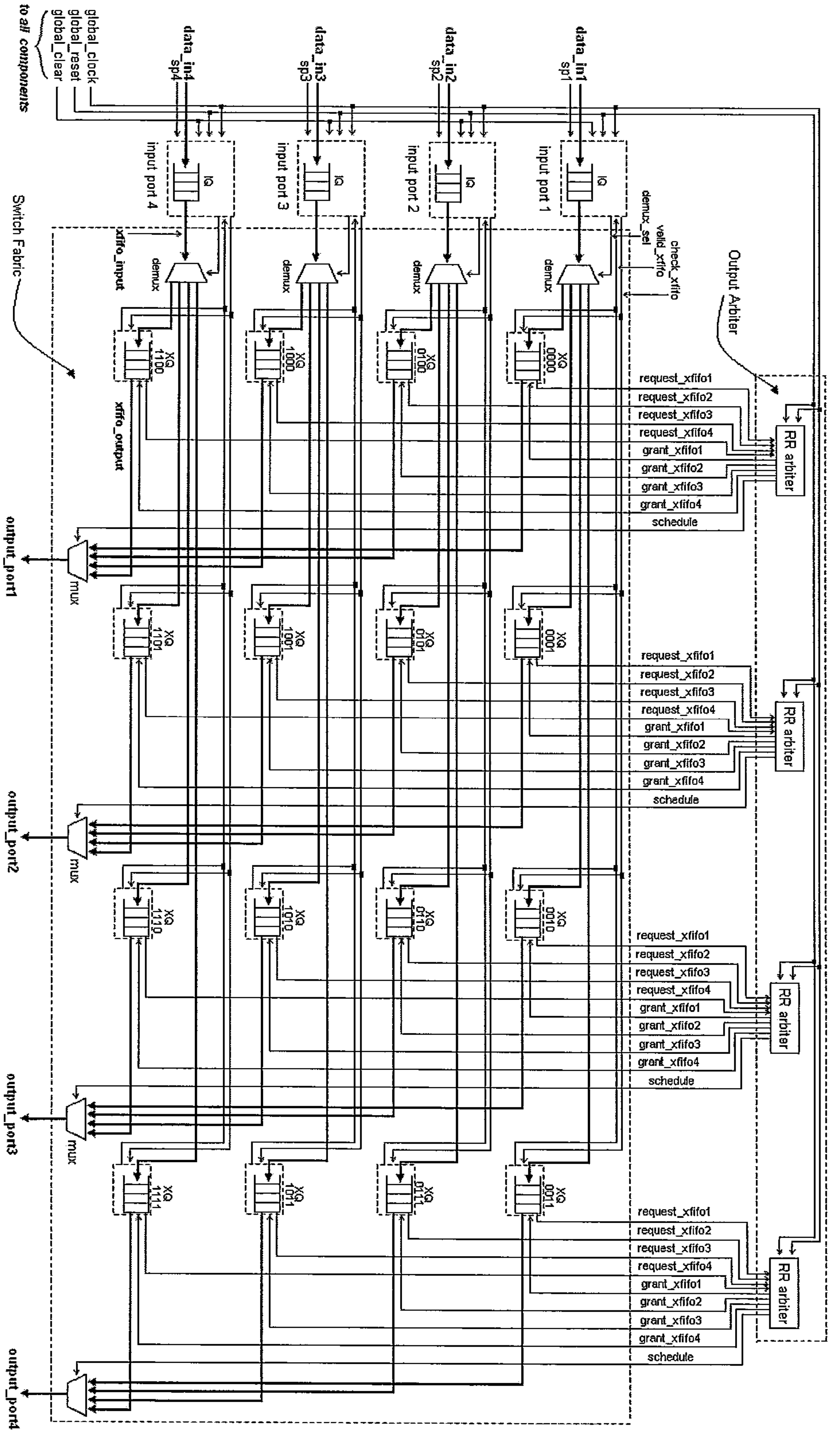
In conclusion, the *CIXQ* switch may simplify the scheduling compared to an *IQ* switch at the expense of adding N^2 FIFO queues in the crossbar, which increases the design difficulty and cost. In addition, the *CIXQ* switch requires abut three times as much power as the *IQ* switch for modest sizes *(4x4, 8x8 and 16x16*) in an FPGA implementation.

Appendix A: High-level Schematic of *CIXQ* Switch

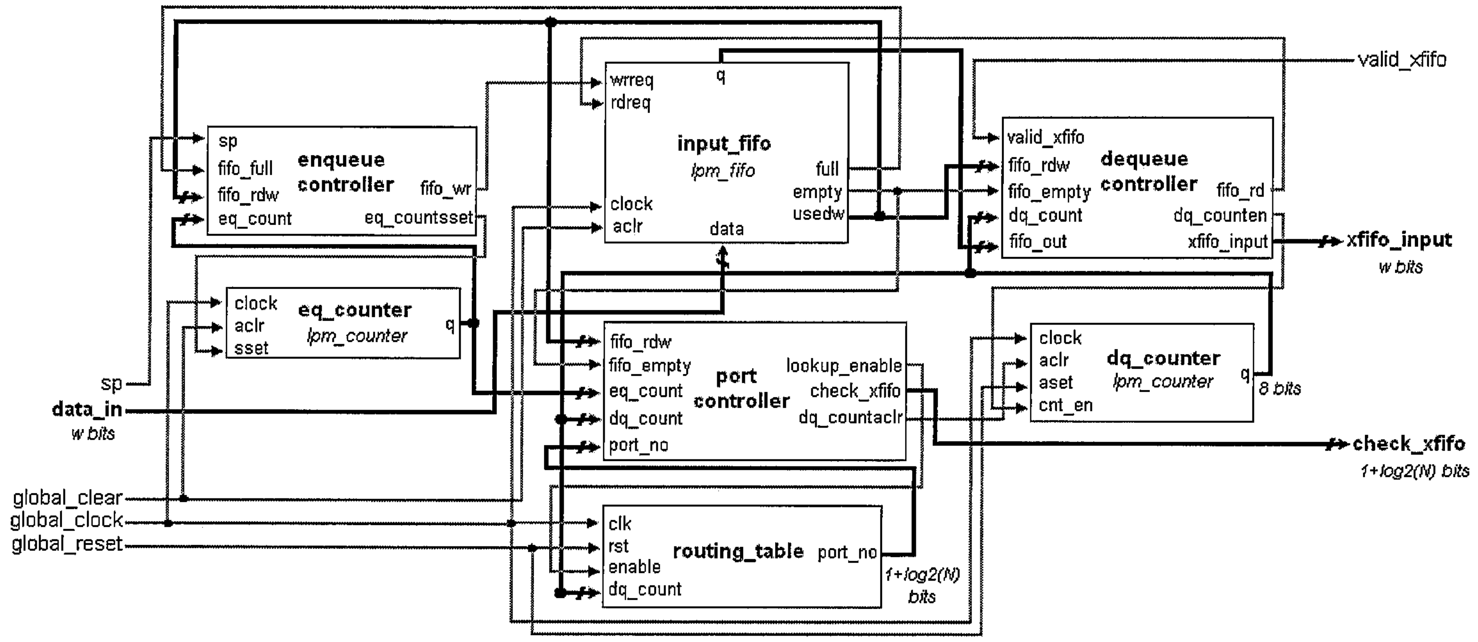
A.1: A *CIXQ* Switch in Broadcast-and-Select Design



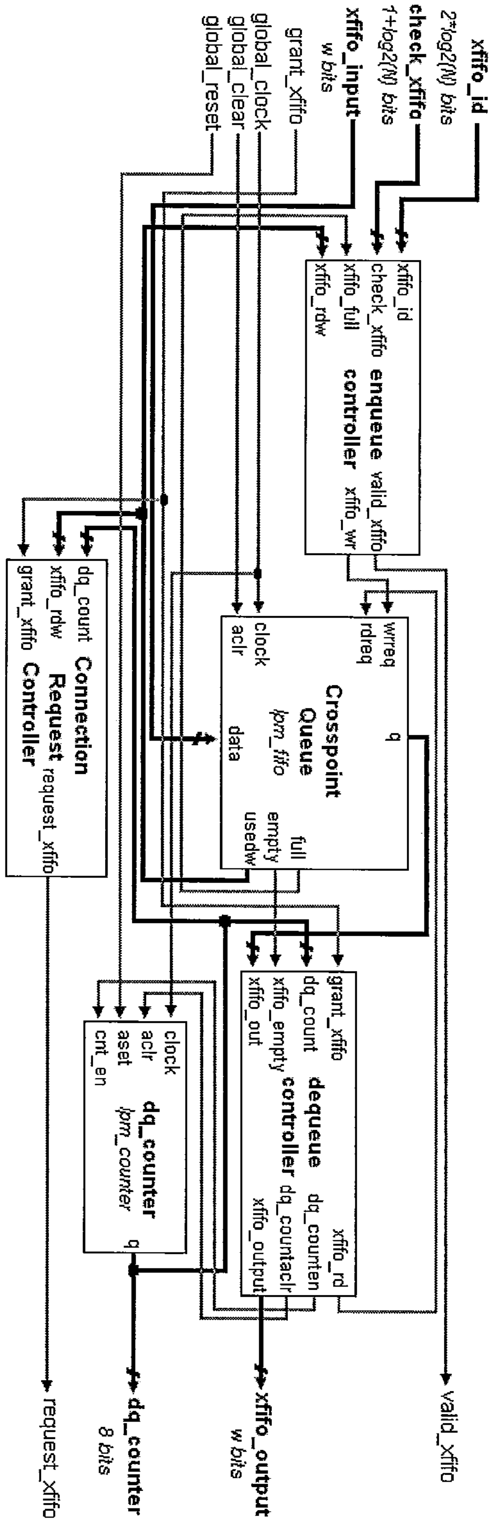
A.2: A *CIXQ* Switch in DEMUX-MUX Design



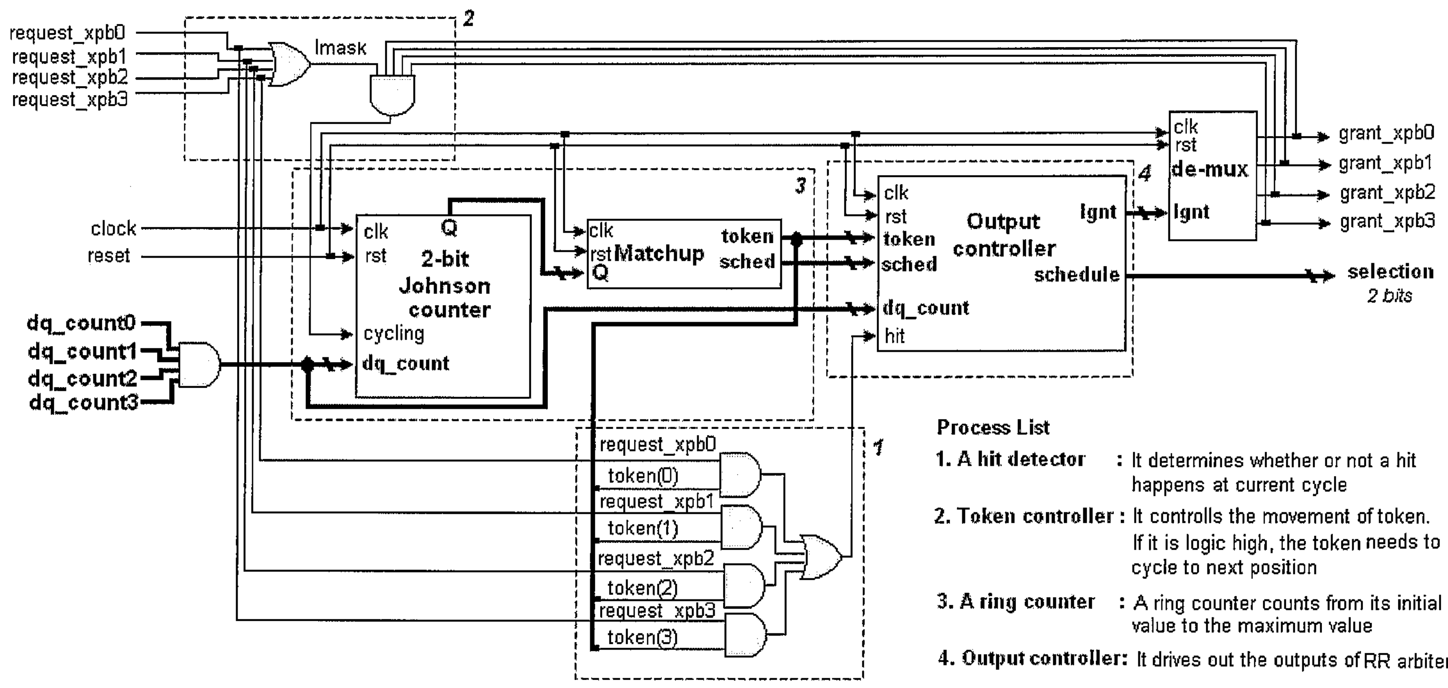
Appendix B High-level Schematic of an Input Port



Appendix C: High-level Schematic of a Crosspoint Queue



Appendix D: High-level Schematic of a RR arbiter

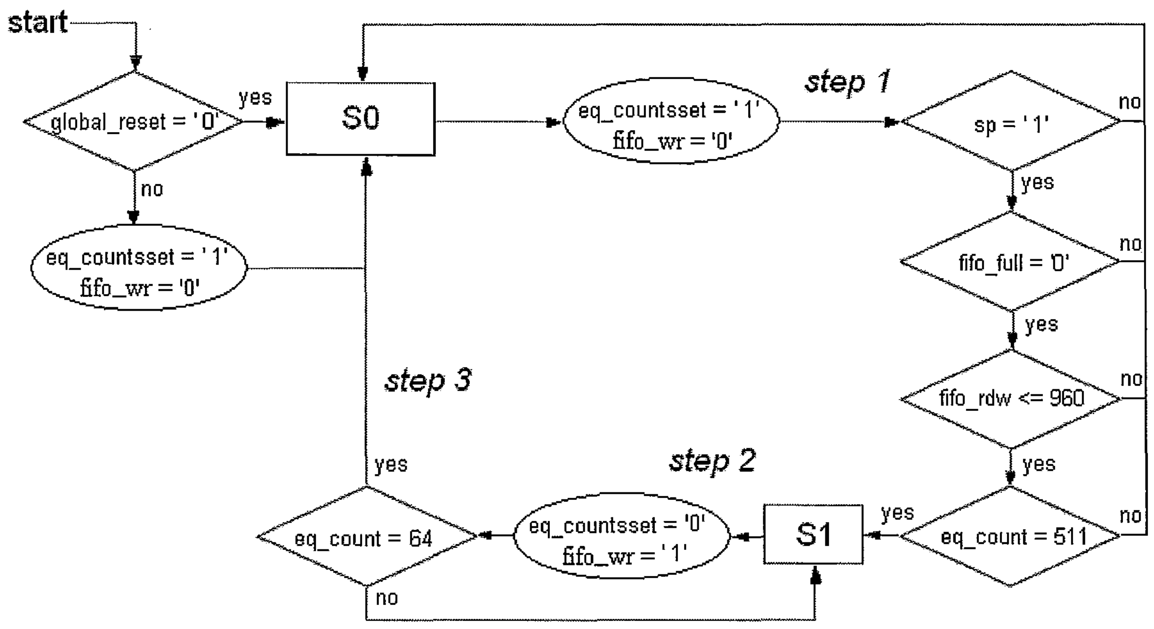


Process List

1. A hit detector	: It determines whether or not a hit happens at current cycle
2. Token controller	: It controlls the movement of token. If it is logic high, the token needs to cycle to next position
3. A ring counter	: A ring counter counts from its initial value to the maximum value
4. Output controller:	It drives out the outputs of RR arbiter

Appendix E: Design State Diagrams

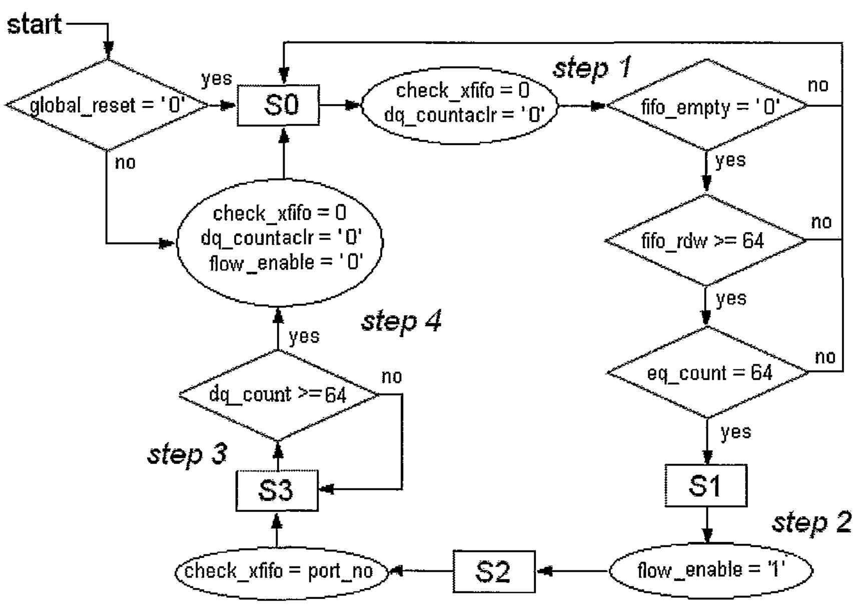
E.1: Enqueue controller state machine for Input Port



S0: The initial state. No data is being enqueued in IQ

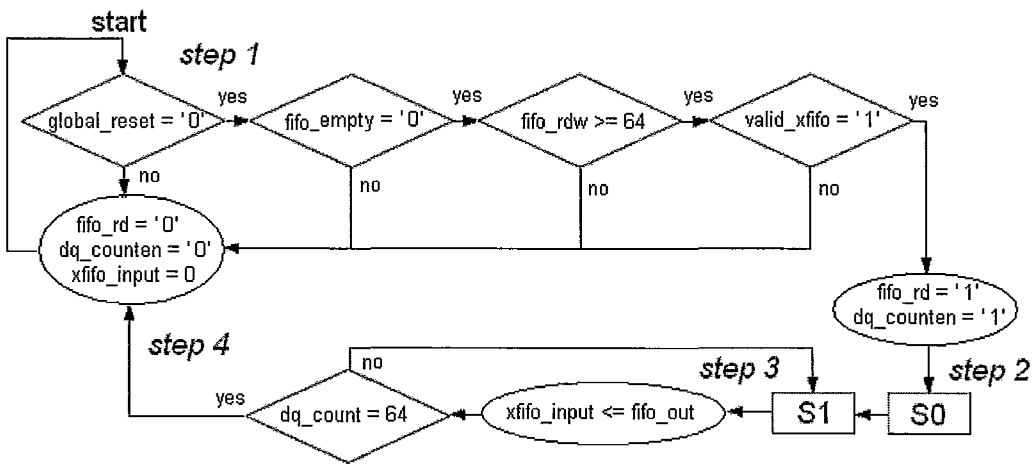
S1: The equeue process is active. It will go back to initial state when 512 bits of data have been completely enqueued in IQ

E.2: Port controller state machine for Input Port



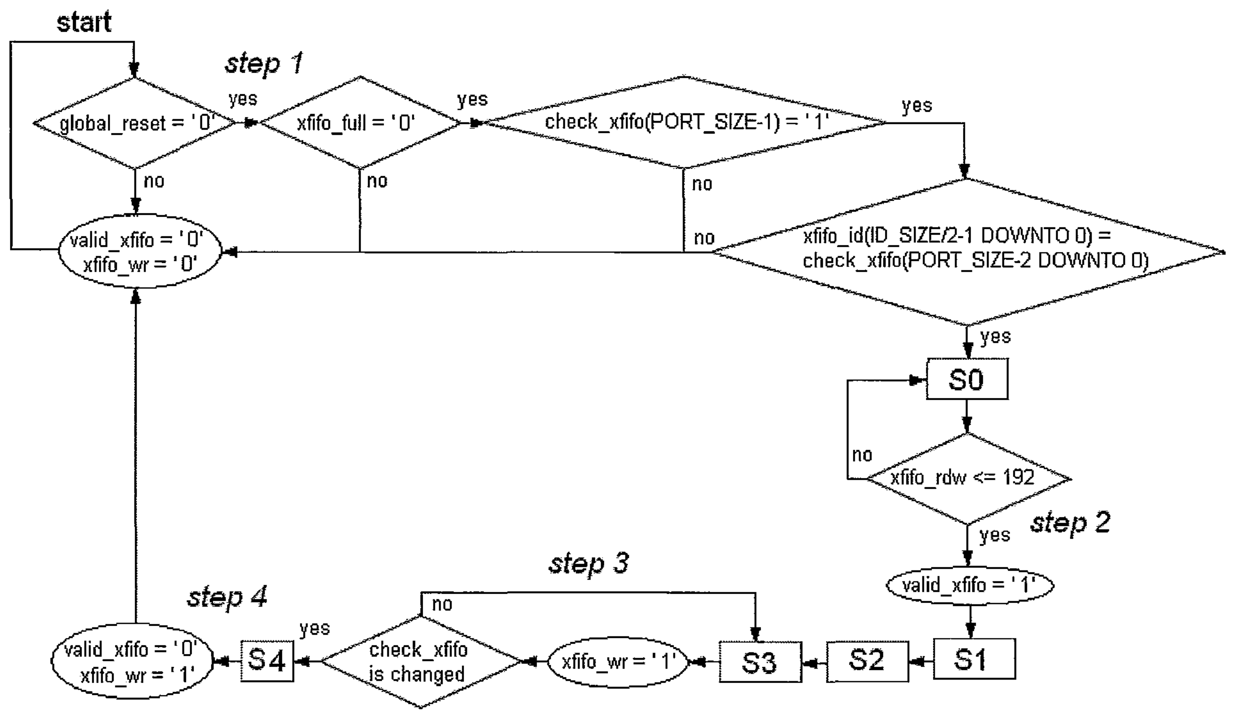
S0: The initial state. The dequeue counter is cleared and check_xfifo is initialized
S1: Enable to look up destination output port from flow controller
S2: Broadcast a query to check if the corrponding XQ is eligible for incoming data
S3: Wait until a packet is completely dequeued to XQ while the dequeue process is active, then disable the flow controller and go back to initial state

E.3: Dequeue controller state machine for input port



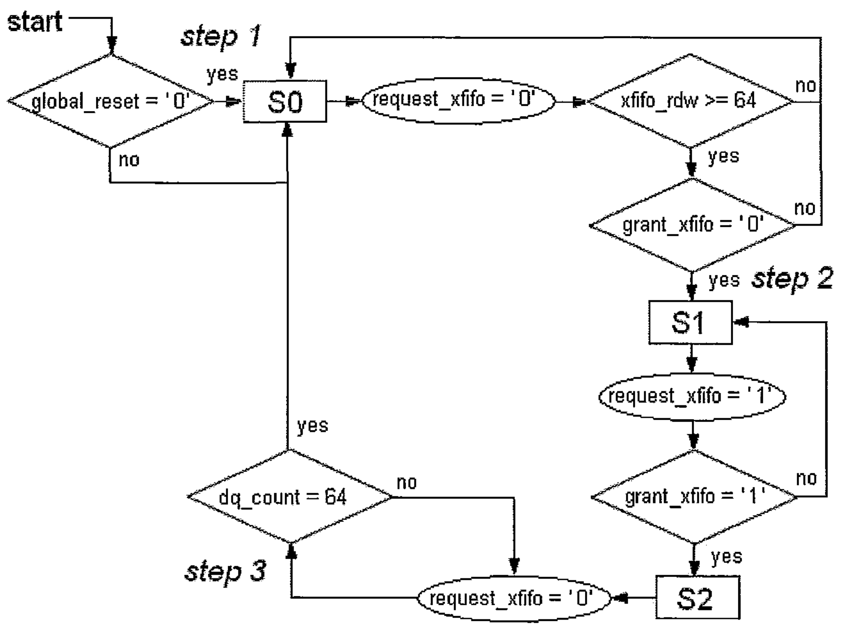
S0: The dequeue process is enabled
S1: The data is being dequeued from IQ to XQ

E.4: Enqueue controller state machine for XQ



S0: The initial state. The incoming packet is destined to the current XQ
S1: A waiting time for input port to receive *valid_xfifo*
S2: A waiting time for input port to enable its dequeue process
S3: The XQ receives the data from input port until *check_xfifo* is changed
S4: Keep writing for one more clock cylce

E.5: Output request controller state machine for XQ

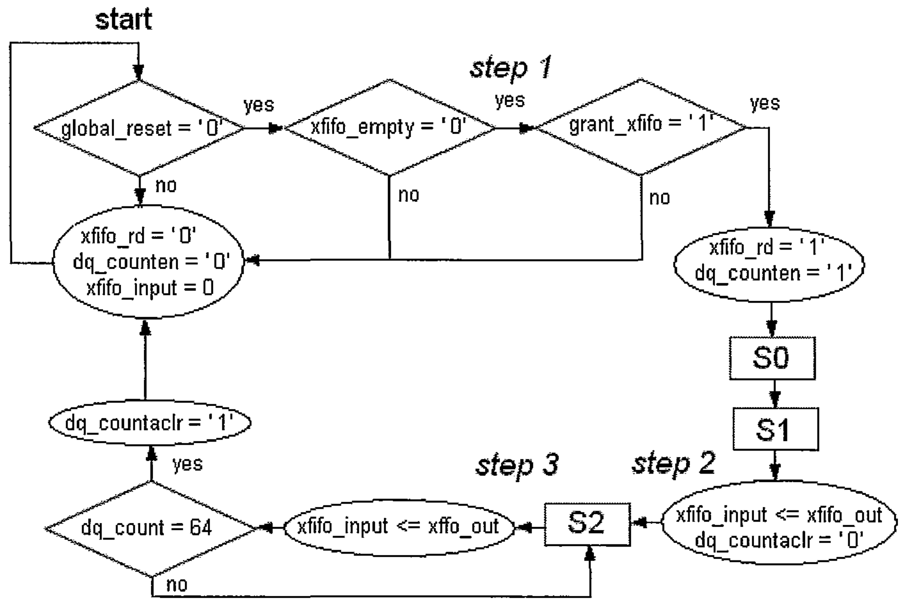


S0: The intital state. No request is issued

S1: Wait for a grant after a request is made

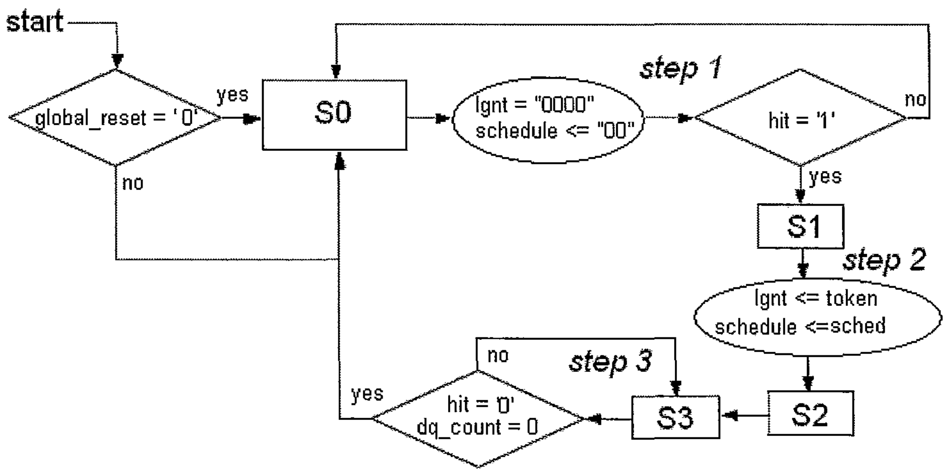
S2: The request is granted. Reset the state machine after a packet is completely dequeued to output port

E.6: Dequeue controller state machine for XQ



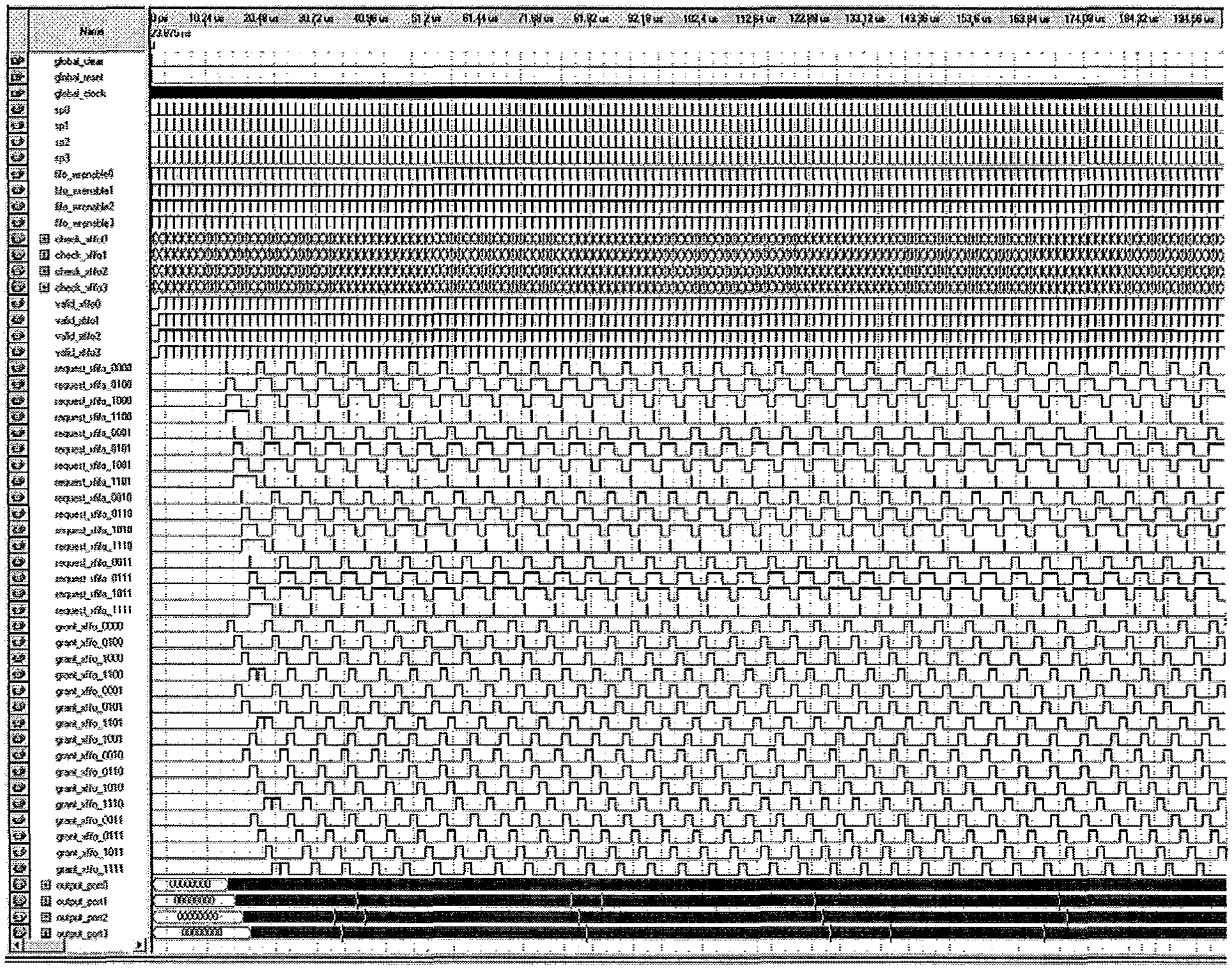
S0: The initial state. The dequeue counter is enabled and XQ is ready to dequeue data
S1: The dequeue counter is set to zero and start moving data out of XQ
S2: Continue moving data out of XQ until a packet is completely dequeued to output port

E.7: State Diagram for Scheduler



S0: The initial state. The grant and selection signals are initialized to zero.
S1: A hit occurs. The output are driven according to output of the ring counter.
S2: Wait for the internal delay
S3: The output remain unchanged until the hit signal is reset and XQ has completely dequeue a packet

Appendix G: Simulation Result of *4x4 CIXQ* Switch



Reference

1. H. Ahmadi and W. E. Denzel, "A survey of modern high-performance switching techniques", *IEEE JSAC*, Volume 7, pp. 1091-1103, Sept. 1989.

2. R. B. Magill, C. E. Rohrs, and R. L. Stevenso, "Output-queued switch emulation by fabrics with limited memory", *IEEE Journal on Selected areas in comm.*, vol. 21, no. 4, pp. 606 – 615, May 2003.

3. T. Mudge, "Power: a first class architectural design constraint", *IEEE Computer*, pp. 52 – 58, May 2001

4. F. J. Pollack, "New microarchitecture challenges in the coming generation of CMOS process technologies," *32nd Annual IEEE/ACM Int. Symposium on Microarchitecture (MICRO'99)*, Haifa, Israel. pp. 2, Nov. 1999.

5. J. Wang and T. H. Szymanski, "A power analysis of Input-Queued and Crosspoint-Queued crossbar switches", *CCECE09*, St. John's, Canada, May 2009.

6. M. J. Karol, M. Hluchyj, and S. Morgan, "Input vs. Output Queueing on a space-division packet switch", *Proc. GLOBECOM*, pp. 659 – 665, 1986.

7. N. McKeown, "The iSLIP scheduling algorithm for input-queued switches", *IEEE Trans. Networking*, volume 7, pp. 188-200, 1999.

8. H. J. Chao and J-S. Park, "Centralized contention resolution schemes for a larger-capacity ATM switch", *Proc. IEEE ATM Workshop*, Fairfax, VA, pp. 10-11, May 1998.

9. T. Anderson, S. Owicki, J. Saxie, and C. Thacker, "High speed switch scheduling for local area networks", *ACM Trans. Comput. Syst.* , vol. 11, no. 4, pp. 319 – 352, Nov. 1993.

10. N. McKeown, A. Mekkittikul, V. Anantharam, J. Walrand, "Achieving 100% Throughput in an input-queued switch," *IEEE Trans. Comm.*, Vol. 47, No. 8, pp. 1260 – 1272, 1999.

11. S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input output queued switch", *IEEE J. Select. Areas Comm.*, vol. 17, pp. 1030 – 1039, June 1999.

12. I. Stocia and H. Zhang, "Exact emulation of an output queueing switch by a combined input output queueing switch", *Porc. 6th IEEE/IFIP IWQoS*, Napa, CA, pp. 218 – 224, 1998.

13. R. Bakka and M. Dieudonne, “Switching circuit for digital packet switching network”, U. S. Patent 4 314-367, 1982.

14. S. Nojima, E. Tsutsui, H. Fukuda, and M. Hashimmoto, “Integrated services packet network using bus matrix”, *IEEE JSAC*, Volume SAC-5, pp. 1284-1291, Oct. 1987.

15. I. Elhanany and M. Hamdi, “High-performance packet switching architecture”, Springer London Publisher, ISBN: 978-1-84628-273-7, 2007.

16. A. Gupta, L. Barbosa, and N. Georganas, “16x16 limited intermediate buffer switch module for ATM networks”, *Porc. of IEEE GLOBELCOM*, pp 939 – 943, 1991.

17. A. Gupta, L. Barbosa, and N. Georganas, “Limited intermediate buffer switch modules and their interconnection networks for B-ISDN”, *Proc. of IEEE ICC*, pp. 1646 – 1650, 1992.

18. E. Re and R. Fantacci, “Performance evaluation of input and output queueing techniques in ATM switching systems”, *IEEE Trans. on Comm.*, pp. 1565 – 1575, 1993.

19. M. Nabeshima, “Performance evaluation of a combined input- and crosspoint- queued switch”, *IEICE Trans. on Comm.*, pp. 737 – 741, 2000.

20. K. Yoshigoe and K. Christensen, “A parallel-polled virtual output queued switch with a buffered crossbar”, *Proc. of IEEE Workshop on High Performance Switching and Routing*, pp. 271 – 275, 2001.

21. R. Rojas-Cessa, E. Oki, and H. Chao, “CIXB-1: combined input-one-cell crosspoint buffered switch”, *Proc. of IEEE Workshop on High Performance switching and routing*, pp. 324 – 329, 2001.

22. R. Rojas-Cessa. R, Oki. E, Chao. H. J, “On the combined input-crosspoint buffered switch with round-robin arbitration”, *IEEE Trans. on Volume 53*, pp. 1945-1951, Nov. 2005.

23. Y. Zheng, W. Gao, “A dual round-robin algorithm for combined input-crosspoint-queued switches”, *IEEE Int. Conf.*, pp. 193-198, Oct. 2005.

24. I. Radusinovic and M. Pejanovic, “Impact of scheduling algorithms on performances of buffered crossbar switch fabric”, *Proc. of IEEE ICC*, pp. 2416 – 2420, 2002.

25. L. Mhamdi and M. Handi, “Output queued switch emulation by one-cell-internally buffered crossbar switch”, *Proc. IEEE Global Telecom. Conf.*, Vol. 7, pp. 3688 – 3693, Dec. 2003.

26. R. B. Magill, C. E. Rohrs, and R. L. Stevenson, "Output-queued switch emulation by fabrics with limited memory", *IEEE J. Sel. Area Comm.*, vol. 21, pp. 606 – 615, May 2003.

27. F. Abel, C. Minkenberg, R. Luijetn, M. Gusat, and I. Illiadis, "A four-terabit packet switch supporting long round-trip times", *IEEE Micro Magazine*, vol. 23, no. 1, pp. 10 – 24, Jan/Feb 2003.

28. Vitesse Semiconductor, 3.6 Gb/s 144x144 Crosspoint Switch datasheet, VSC 3140, Camarillo, California. Available: www.vitesse.com

29. TRIQUINT Semiconductor, TQ8033 Data Sheet, Hillsboro, OR. Available: www.triquint.com

30. H. Kariniemi, J. Nurmi, P. Fagerlund, J. Liitola and J. Alinikula, "ATM Switch for 2.488 Gbit/s CATV Networks on FPGA with a High-Throughput Buffering Architecture", *IEEE*, pp. 127 – 130, 2002.

31. K. Yoshigoe, K. Christensen, and A. Jacob, "The RR/RR CICQ switch: hardware design for 10-Gpbs link speed", *IEEE Int. Performance, Computing, and Comm. Conf.*, pp. 481 – 485, April 2003.

32. M. Katevenis, G. Passas, D. Simos, I. Papasfestathiou and N. Chrysos, "Varibale packet seize buffered crossbar (CICQ) switches", *IEEE Int. Conf. on Comm.*, pp. 1090 – 1096, June 2004.

33. Synopsys Design Compiler; http://sysnopsys.com.

34. Cadence Design Systems; Silicon Encounter Place-and-Route Reference Manual; http:www.cadence.com

35. Jaidhar C. D and A. V. Reddy, "Variable length packets switching using novel combined input crossbar queue switch with virtual crossbar queues", Int. Journal of Computer Science and Engineering Systems, Vol. 1, No. 2, pp. 125 – 129, April 2007.

36. Altera, "FPGA power management and modeling techniques", White paper, ver. 1.0, Altera Corporation, November 2007.

37. D. Soudris, C. Piguet and C. Goutis, "Designing CMOS circuits for low power", Kluwer Academic Publishers, ISBN: 1-4020-7234-1, 2004.

38. D. Duarte, N. Vijaykrishnan, M. J. Irwin, H-S Kim and G. McFarland, "Impact of scaling on the effectiveness of dynamic power reduction schemes", Proc. of the 20th Int. Conf. on Computer Design (ICCD), Freiburg, Germany, September 16 – 18. 2002.

39. F. Najm, "A survey of power estimation techniques in VLSI circuits", *IEEE Trans.s on VLSI Systems*, pp. 446-455, December 1994.

40. F. Najm, "Transition density: a new measure of activity in digital circuits", *IEEE Trans. on Computer-Aided Design*, vol.12, No.2, pp.310-323, February 1993.

41. R. Burch, F. Najm, P. Yang, and T. Trick, "A Monte Carlo approach for power estimation," *IEEE Trans. on VLSI Systems*, vol.1, no.1, pp.63-71, March 1993.

42. K. Muller-Glaser, K. Krisch, and K. Neusinger, "Estimating essential design characteristics to support project planning for ASIC design management", *IEEE Int. Conf. on Computer-Aided Design 1991*, pp.148-151, November 1991.

43. F. Najm, "Towards a high-level power estimation capability", *1995 Int. Symposium on Lower-Power Design*, pp.87-92, April 1995.

44. D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures of energy consumption at register transfer level", *1995 Int. Symposium on Low-Power Design*, pp.81-86, April 1995.

45. S. Powll and P. Chau, "Estimating power dissipation of VLSI signal processing chips: the PFA technique", *VLSI Signal Processing IV*, pp.250-259, 1990.

46. T. Sato, Y. Ootaguro, M. Nagamatsu, and H. Tago, "Evaluation of architecture-level power estimation for CMSO RISC processors", *1995 Symposium on Low-Power Electronics*, pp.44-45, October 1995.

47. P. Landman and J. Rabaey, "Activity-sensitive architectural power analysis", *IEEE Trans. on Computer-Aided Design*, June 1996.

48. R. Mehra, "High-level power estimation and exploration", *1994 Internal Workshop on Low Power Design*, pp.197-202, April 1994.

49. N. Kumar, S. Katkoori, L. Rader, and R.Vemuri, "Profile-driven behavioural synthesis of lower-power VLSI systems", *IEEE Design and Test of Computers*, pp.70-84, Fall 1995.

50. V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization", *IEEE Trans. on VLSI Systems*, pp.437-445, December 1994.

51. D. Lidsky and J. Radaey, "Early power exploration: a World Wide Web application," *33rd Design Automation Conference*, 1996.

52. Actel, Power Analysis Tools – Power Calculator, available at http://www.actel.com.

53. Altera, PowerPlay Early Power Estimator, available at http://www.altera.com.

54. Lattice Semiconductor, ispLEVER – Power Calculator, available at http://www.latticesemi.com

55. Xilinx, XPower Estimator (XPE), available at http://www.xilinx.com

56. Actel, Power Analysis Tools – SmartPower, available at http://www.actel.com.

57. Altera, PowerPlay Power Analysis in Quartus II Development Software Handbook, Version 8.1, Volume 3, Altera Corporation, USA, Nov. 2008.

58. Xilinx, XPower Analyzer, available at http://www.xilinx.com

59. Altera, Evaluating Power for Altera Devices, Application Note 74, version 3.1, Altera Corporation, July 2001.

60. K. K. W. Poon, Power Estimation for Field Programmable Gate Arrays, Master thesis, the University of British Columbia, August 2002.

61. C. Patel, S. Chai, S. Yalamanchili, D. Shimmel, "Power constrained design of multiprocessor interconnection network", *Int. Conf. on Computer Design,* 1997

62. H. Zhang, M. Wan, V. Gerorge, and J. Rabaey, "Interconnection architecture exploration for low-energy reconfigurable single-chip DSP", *Proc. IEEE Computer Society Workshop on VLSI,* 1999.

63. L. angen, D.; Brinkmann, A.; Ruckert, U., "High level estimation of the area and power consumption of on-chip interconnects", *IEEE Int'l ASIC/SOC Conf.*. 2000.

64. Wassal, A. G.; Hasan, M. A., "Lower-power system-level design of VLSI packet switching fabrics", *CAD of Integrated Circuits and Systems, IEEE Trans.*, June 2001.

65. E. Geethanjali, V. Narayanan and M. J. Irwin, "An analytical power estimation model for crossbar interconnects", *IEEE ASIC/SOC conf.*, pp.119-123, Sept. 2002.

66. T. T. Ye, L. Benini, and G. D. Micheli, "Analysis of power consumption on switch fabrics in network routers", *IEEE Proc. of Design Automation Conf.*, pp.524-529, June 2002.

67. H. S. Wang, L. S. Peh, S. Malik, "A power model for routers: modeling the Alpha 2136 and Infiniband routers", *IEEE Micro*, pp.26-35, Jan – Feb 2003.

68. T. Wu, C-Y Ying, and M. Hamdi, "A 2Gb/s 256x256 CMOS crossbar switch fabric core design using pipeline MUX", *IEEE Int. Symp. Circuits and Systems*, vol.2, pp.568-571, May 2002.

69. T. H. Szymanski, H. Wu, A. Gourgy, "The power complexity of multiplexer-based optoelectronic crossbar switches", *IEEE Trans. on VLSI Systems,* vol.13, No.5, pp.604-617, May 2005.

70. D. G. Simos, "Design of a 32x32 variable-packet-size buffered crossbar switch chip", Technical report FORTH-ICS/TR-339, Inst. Of Computer Science, July 2004.

71. Stewart V. Hoover and Roanl F. Perry, "Simulation: A problem-solving approach", Addison-Wesley Publishing Company Inc., pp. 227 – 228, August 1990.

72. A. Baranowske, G. Danilwicz, W. Kabacinski, J. Kleeban, D. Parniewics and P. Dabrowskil, "Performance evaluation of the multiple output queueing switch under different traffic patterns", *IEEE GLOBECOME'05*, 2005.

73. Raj Jain, "The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling", Wiley publisher, New York, ISBN: 0471503363, 1991.

74. J. Ge, "Round-robin arbiter design", CDES 2006, Las Vegas, Nevada, USA, pp. 24-28, June 2006.

75. S. Brown and Z. Vranesic, "Fundamentals of digital logic with VHDL design", McGraw-Hill Science, second edition, July, 2004, ISBN: 0072499389

76. "Quartus II Handbook Version 9.0 Volume 3", Altera Corporation, March 2009.

77. "Cyclone II Device Handbook, Volume 1", Altera Corporation, Feb. 2008.

78. Altera, "Chapter 1: Stratix III device family overview", Stratix III device handbook, version 1.8, Altera Corporation, USA, May 2009.

79. Altera, "Stratix III device datasheet", version 2.0, Altera Corporation, May 2009.

80. Available: http://en.wikipedia.org/wiki/Moore's_law

81. The International Technology Roadmap for Semiconductors, 2007 Edition, International Sematech, Austin, Texas, 2007. Available: www.itrs.net