

VLSI Support for Scheduling and Buffer Management in High Speed Packet-Switched Networks

by

MEHDI KAZEMI-NIA

B.Sc. in Elec. Eng., Sharif University of Technology, Tehran, 1988

M.A.Sc. in Elec. Eng., The University of Windsor, Windsor, 1994

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF**

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

**We accept this thesis as conforming
to the required standard**

THE UNIVERSITY OF BRITISH COLUMBIA

January 2000

© Mehdi Kazemi-Nia, 2000

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Elec & Comp. Eng.

The University of British Columbia
Vancouver, Canada

Date Jan 31 / 2000

ABSTRACT

This thesis presents a number of new approaches for designing fast, scalable queuing structures in VLSI for very high speed packet-switched networks. Such queuing structures are necessary for implementing packet buffers in switches and routers that have multi Gigabit-per-second (Gb/s) ports. The thesis addresses the design of two specific queue architectures: balanced parallel multi-input multi-output (MIMO) buffers, and systolic parallel priority queues (PPQ).

A methodology for the systematic design of order-preserving parallel MIMO buffers is presented. The MIMO buffer employs an arithmetic-free systolic routing network and bank of parallel FIFO buffers to yield a load-balanced realization with increased bandwidth. Using this methodology we derived scalable parallel buffer structures that can be designed to match the rate of ultra high-speed links using current memory technology that uses moderate clock rates. A small prototype of the MIMO buffer attains a rate of 10.6 Gb/s which is more than adequate to support a Sonet OC-192 link. The combined use of pipelined architecture and dynamic CMOS circuits resulted in significant reduction in design complexity and substantial performance gains in speed and area.

The thesis also addresses a generalization of the priority queue concept to a systolic parallel priority queue (PPQ) which can be scaled to meet the requirements of ultra high-speed links using standard CMOS technology. The PPQ has several applications in implementing real-time fair schedulers or buffer management algorithms in packet routers. The PPQ maintains prioritized access to the data it contains at all times, and the access time to the data is fixed and independent of the PPQ size, i.e. $O(1)$ -time access. The proposed systolic PPQ is rate-adaptive in the sense that the PPQ operates correctly even when the queue input rate and output rate are different. This

decoupling of the input and output packet flow rates is a distinguishing feature of the PPQ concept because in practice the output rate of the queue is controlled by the available link bandwidth which may vary (or even become zero) independent of the packet arrival rate.

TABLE OF CONTENTS

ABSTRACT	ii
LIST OF TABLES	vii
LIST OF FIGURES	viii
DEDICATION	xi
ACKNOWLEDGEMENTS	xii
 CHAPTER 1 Overview and Motivation	 1
1.1 Buffer Management for High Speed Links	4
1.2 Packet-Scheduling	5
1.3 Main Contributions and Thesis Organization	7
 CHAPTER 2 Survey and Background	 11
2.1 Parallel FIFO Buffers	12
2.2 Priority Queues	20
 CHAPTER 3 Parallel FIFO Buffers for Gigabit Packet Networks	 26
3.1 Introduction	26
3.2 MIMO Buffer Design Principals	29
3.2.1 Basic Operation of the MIMO Buffer	29
3.2.2 The MIMO Buffer Structure	30
3.3 The Balanced Distribution Network (BDN)	33
3.4 Dynamic Load Balancing on the BDN	38
3.5 Some Applications of the MIMO Buffers	46
3.6 Summary	54

CHAPTER 4	VLSI Implementation of the Parallel MIMO Buffer	55
4.1	Overview	55
4.2	VLSI Design Strategy	56
4.3	Circuit Diagram of a Switch Element	58
4.4	Design Simulation Results	65
4.5	Summary	68
CHAPTER 5	A Systolic Parallel Priority Queue for High Speed Packet Networks .	69
5.1	Overview	69
5.2	Scheduling Networks and Priority Queues	70
5.3	Systolic Parallel Priority Queue	71
5.3.1	VLSI Word-Model of a Parallel Sorter	74
5.3.2	Recursive Construction of Systolic PPQs	74
5.4	Retimed Design	79
5.5	Summary	83
CHAPTER 6	Scalable PPQs with Output Rate-Control	84
6.1	Introduction	84
6.2	A PPQ Design with Output Rate-Control	86
6.3	The PPQ Retimed Design with Output Rate-Control	91
6.4	Systolic PPQs with an Unequal Number of Inputs and Outputs .	96
6.5	Bit-Serial Realization of the Systolic PPQ	98
6.6	Structure of Parallel Sorters	100
6.6.1	Recursive Construction of Parallel Sorter	101
6.6.2	Direct Construction of Parallel Sorters	105
6.7	Primitive Sorter Timing	110
6.8	VLSI Design Strategy and Implementation Issues	112
6.9	Summary	115

CHAPTER 7	Conclusion and Future Directions	116
BIBLIOGRAPHY		121
APPENDIX A	ATM Technology	131
	A.1 ATM Networking and Switching	133
	A.2 ATM Switch Architectures	134
	A.3 ATM Buffer Management	140
APPENDIX B	Asynchronous System Design	142

LIST OF TABLES

Table 4.1: Transistors' sizes in the nMOS chain	64
Table 6.1: Comparing two methods of implementation of a $2m$ -cell sorter	109

LIST OF FIGURES

Fig. 1.1:	Serial-to-parallel conversion and byte staggering at the opto-electronic interface	4
Fig. 2.1:	(a) Operation of a shared-buffer that preserves the cell arrival sequence (b) Illustration of pack-and-shift approach in three time-slots	13
Fig. 2.2:	(a) Packing ranked cells on a butterfly network (b) Packing and shifting ranked cells on a butterfly network	15
Fig. 2.3:	Structure of shared buffers in the Knockout switch	17
Fig. 2.4:	MIMO buffer and its symbol	19
Fig. 2.5:	(a) Parallel buffer (b) ATM access point (c) Buffer concentrator	20
Fig. 2.6:	Inserting a new arrival into a non-systolic PQ	22
Fig. 2.7:	Buffer manager structure	23
Fig. 3.1:	MIMO buffer in a multi-stage switch and its symbol	28
Fig. 3.2:	Internal structure of the buffer	31
Fig. 3.3:	Internal Structure of an 8x8 BDN	33
Fig. 3.4:	Input/Output port alignment of the BDN	34
Fig. 3.5:	Two states of the switch element	36
Fig. 3.6:	I/O diagram of the SWT and cell format	37
Fig. 3.7:	Four routing phases of the buffer	39
Fig. 3.8:	Folded crossbar array of the BDN	42
Fig. 3.9:	Distributed control in the VEC	43
Fig. 3.10:	Two configurations for packing the incoming cells	45
Fig. 3.11:	Self routing property of banyan networks	46

Fig. 3.12: Internal blocking in banyan networks	49
Fig. 3.13: Expansion modules (ESEs)	51
Fig. 3.14: Buffered-concentration modules (BSEs)	52
Fig. 3.15: Alternative buffered core modules	53
Fig. 3.16: Dilated banyan network with internal buffers	54
Fig. 4.1: Part of the VHDL code	57
Fig. 4.2: Circuit diagram of the crossbar control of SWT	58
Fig. 4.3: Signal and data-path of a SWT	60
Fig. 4.4: Dynamic D-type flip-flop circuit	61
Fig. 4.5: nMOS transistors chain and the RC model	62
Fig. 4.6: Layout diagram of a 4x4 BDN	65
Fig. 4.7: Simulation result of the 4 x 4 BDN during three time-slots	66
Fig. 5.1: A typical scheduling network	71
Fig. 5.2: A PPQ contains $m \times w$ cells	73
Fig. 5.3: The I/O configuration of a $2m$ -cell sorter	75
Fig. 5.4: A priority queue with m inputs, m outputs, and depth 1	76
Fig. 5.5: Recursive construction of the parallel priority queue $PPQ(m, w)$	77
Fig. 5.6: Retimed design of $PPQ(m, 1)$	80
Fig. 5.7: Recursive construction of the systolic parallel priority queue $PPQ(m, w)$	82
Fig. 6.1: $PPQ(m, 1)$ with inhibit feature	86
Fig. 6.2: Building recursively priority queues with different depth sizes and inhibit feature	88

Fig. 6.3:	Several steps of the operation of a $PPQ(1, 5)$	89
Fig. 6.4:	Retimed design of $PPQ(m, 1)$ with inhibit feature	91
Fig. 6.5:	PPQs with different depth sizes and inhibit feature	92
Fig. 6.6:	Several steps of the operation of a systolic $PPQ(2, 3)$	94
Fig. 6.7:	PPQs with an unequal number of inputs and outputs	97
Fig. 6.8:	Two pictorial representations of a primitive sorter	99
Fig. 6.9:	Configuration of $ST(4)$ and its two pictorial representations	101
Fig. 6.10:	Realization of an 8-cell sorter	103
Fig. 6.11:	Recursive construction of a $2m$ -cell sorter and its two geometries	104
Fig. 6.12:	Configuration of a 6-cell sorter using primitive elements	106
Fig. 6.13:	Construction of an 8-cell sorter $ST(8)$ using primitive elements	107
Fig. 6.14:	Realization of a $2m$ -cell sorter $ST(2m)$ using primitive elements	108
Fig. 6.15:	Primitive sorter	110
Fig. 6.16:	Circuit diagram of the ST	113
Fig. A.1:	ATM cell format and header components	133
Fig. A.2:	An ATM switch	135
Fig. A.3:	Shared buffer switch architecture	137
Fig. A.4:	The buffer manager at each output port of an ATM switch	141
Fig. B.1:	Two-phase signalling scheme	145
Fig. B.2:	Four-phase signalling scheme	146
Fig. B.3:	Two-phase FIFO structure	150
Fig. B.4:	Four-phase FIFO structure	151

To the memory of my father

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Hussein Alnuweiri, for his friendly guidance and support throughout the progress of this thesis. My deepest appreciations go to Drs. Mike Jackson, and Steve Wilton for their valuable and enlightening discussions. I am also grateful to Drs. William Dunford and Alan Wagner for their helpful suggestions. My special thanks to Dr. Fayez El-Guibaly from University of Victoria for his helpful comments to bring this thesis to fruition.

Finally, I am indebted to my mother who has always encouraged me in following my dreams. Last but not least, my foremost appreciations go to my wife Elham, without whose love, inspiration, and support this path would likely not have been traversed.

CHAPTER 1

Overview & Motivation

The advances in data communications over optical-fibre serial links has paved the way for the development of high-speed packet switching technologies such as ATM (Asynchronous Transfer Mode) and wire-speed IP (Internet Protocol) packet forwarding. Until a few years ago, the performance of the Internet was limited by the speed of the long-haul links, not by capacity of switches and routers. However, the advancements in switch/router technologies have not kept pace with advances in optics, and with the rapid provisioning of higher speed links (such as the 10 Gigabit per second Sonet OC-192), and the introduction of wavelength division multiplexing (WDM), the bottleneck has moved from the links to the switches and routers. Carriers are

currently deploying very high-density OC-192 dense wavelength division multiplexing (DWDM) systems, and leading DWDM vendors have already announced availability of 40 Gigabit per second (Gb/s) OC-768 systems in the near future¹.

The demand for bandwidth is soaring and today's networks lack the ability to scale network backbone capacities at aggressive rates to accommodate this increased demand for bandwidth. The explosive growth of Internet users, the increased user demand for bandwidth, and the declining cost of technology have all resulted in the emergence of new classes high-speed distributed IP-router architectures with packet-forwarding rates on the order of Terabits per second (Tb/s)².

This recent and unprecedented growth in bandwidth availability, poses a heavy load on the underlying infrastructure, and there is an immediate need for backbone switch/routers that can scale to support these demands. There are a broad range of architectures and techniques being proposed, and several enhanced router architectures³ have been introduced to meet the demand of the ever-growing bandwidth requirements of the Internet. Vendors have been able to introduce routers that offer a range of Tb/s of non-blocking switching capacity⁴. Similarly, current commercial ATM switches support link speeds reaching up to 10 Gb/s (over Sonet OC-192), and experimental switches are supporting link speeds of 40 Gb/s⁵ [1], [2]. Implementations of these networks require the development of new hardware architectures using advanced VLSI technology that are capable of handling large amounts of traffic with throughputs on the order of Gb/s. Moreover, supporting quality-of-service (QoS) in packet networks requires deploying more sophisticated

1. See <http://www.eetimes.com/news/98/1026news/40-gbit.html>

2. See <http://www.pluris.com/html/product.html>.

3. Such routers are known as switched routers or routing switches because they normally employ a switching fabric and distributed architecture for forwarding packets across the router ports. These routers are also called layer-3 switches, wire-speed routers, or hardware-based routers.

4. See <http://www.pluris.com/html/product.html> and <http://www.alliancedatacom.com/lucent-nx64000.htm>.

5. See <http://www.cisco.com/univercd/cc/td/doc/product/atm/index.htm>.

queuing policies in the switching or routing nodes. The first-come first-serve (or FIFO) discipline is no longer adequate in networks that provide service differentiation. Providing QoS guarantees in both cell- and packet-based networks requires the use of a scheduling algorithm in the switches and network interfaces. These algorithms need to be implemented in hardware in a high-speed network. At minimum, some level of priority scheduling must be supported. In more advanced applications, buffers with rate adaptation features must be deployed between the switch and external links.

The main challenge is maintaining the correct operation at very high speeds. As the combined forces of increasing traffic levels and escalating performance demands continue to push network infrastructures to their limits, new buffering structures are required which can be scaled to match the ever-increasing link speeds. Such queuing structures are necessary for implementing packet buffers in switches and routers that have multi Gb/s ports.

Satisfying the QoS requirements for high-speed modern packet switching networks such as ATM requires a combination of high-level architectural synthesis methods (system design level), the use of state-of-the-art circuit design techniques, and the efficient realization of distributed control policies. However, advances in device technology cannot alone meet these demands or it would be cost prohibitive. Even though, the use of deep submicron CMOS technology as the industry standard for IC manufacturing will eventually enable designers to integrate a variety of components into a system-on-a-single chip solutions, limitations of deep submicron technologies can still limit the clock speed.

Therefore, there is an immediate need to achieve speedups using other techniques such as parallel processing. Employing parallel processing approaches to solve the scalability problem, this thesis

develops effective VLSI design methodologies for implementing scalable buffering schemes for high-speed networking applications, and presents a number of novel approaches. Our approaches combine the concepts of parallelism, pipelining, dynamic reconfiguration, and interconnection structures in a unified framework of system design and VLSI implementation.

1.1 Buffer Management for High Speed Links

The ultra high data rates of optical fibre links can not be matched directly by the microelectronic interfaces and components deployed in today's packet switches. Techniques such as serial-to-parallel conversion and byte staggering (or multiplexing) are normally employed at the opto-electronic interface to bring the "parallel" data rate down so that it can be handled by VLSI circuits running at a typical clock rate of up to few hundred MHz. The process is depicted in Fig. 1.1 for

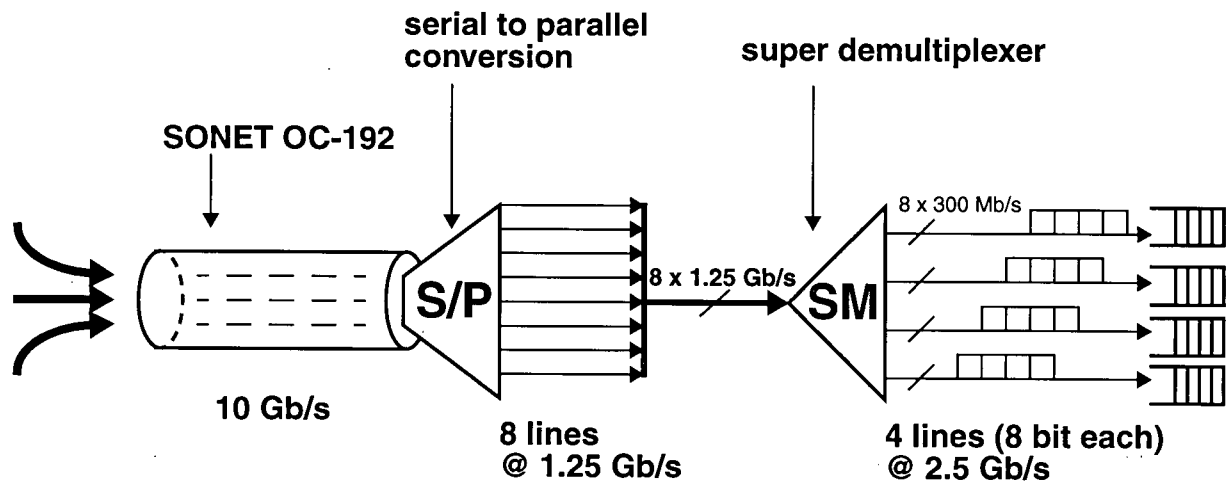


Fig. 1.1: Serial-to-parallel conversion and byte staggering at the opto-electronic interface

an OC-192 link that supplies data at a rate of about 10 Gb/s to a switch line-card with high-speed input buffers (and byte-wide inputs) which are triggered by 300 MHz clocks, i.e. each buffer has an input rate of 2.5 Gb/s. Because low-cost memories are still limited in speed, a 2.5 Gb/s buffer will likely be constructed from an interleaved, or a wide data-word, memory structure. Although faster memory technologies can be employed, such as RDRAM [3], this would normally increase the buffer cost significantly and would not be the ideal solution for the large capacity buffers normally associated with high-speed links.

The demand on the internal and/or the output buffers of a switch will be even higher because of the possible instantaneous traffic aggregation from multiple links over an outgoing link. To maintain low packet loss, it is not uncommon to have the output buffers supporting an input packet rate several times that of the input link rate, even for well-behaved input traffic [1]. For example, with 10 Gb/s input link rate, an output buffer that receives packets from up to four input links, simultaneously, must have an aggregate input rate of 40 Gb/s. Supporting this rate requires using a parallel arrangement of at least sixteen 300-MHz FIFO buffers with an 8-bit data-path and the associated demultiplexing and multiplexing circuitry. Implementation problems do not stop there. A very undesirable effect of using a parallel FIFO buffer arrangement is the possible out-of-order delivery of packets to the outgoing links. In ATM, preserving the order of cells is mandatory. In this case, costly cell reordering must take place before the cells leave the switch.

1.2 Packet-Scheduling

Providing QoS guarantees over packet-switched networks, such as the Internet, relies crucially on the scheduling and buffer management capabilities of the network switches and routers. Packet-switched networks with efficient packet-scheduling mechanisms can support the diverse QoS

requirements of various real-time applications.

Multimedia sessions may have QoS specifications that include bounded end-to-end delay, bounded delay-jitter, bounded cell-loss rates, and guaranteed minimum link bandwidth. Scheduling is required to order packets, for example, to satisfy delay bounds of real-time sessions or to enforce fair bandwidth allocation for sessions sharing a link.

Packet-switched networks exploit a variety of scheduling schemes to provide the QoS guarantees for each connection. Priority-based scheduling is one of the most widely used techniques in packet switches and routers. In this scheme, packets are organized in different priority levels and served according to their priorities. Normally a packet is assigned an attribute which becomes its priority number and it is appended to the packet as an additional field. Packets in a buffer are then ordered and delivered according to their priority value by the scheduler. For example, the priority number may be a finish-time computed by a weighted fair queuing (WFQ) algorithm [4], [5], or a class index assigned by a class-based queuing (CBQ) algorithm [6], [7], etc. In any case, accessing the highest-priority packet requires the use of a *priority queue* structure. By definition a priority queue is a queuing structure for which the dequeue operation always removes the highest (or lowest) priority element in the queue. Arbitrating between a large number of packets on a high-speed link requires an efficient hardware implementation of a priority queue. In high-speed networks, the main challenge is maintaining the correct operation of the priority queue at very high link speeds, e.g. 2.4 to 40 Gb/s [1].

1.3 Main Contributions & Thesis Organization

This thesis presents a number of new approaches for designing fast, scalable queuing structures in VLSI for very high speed packet-switched networks. Such queuing structures are necessary for implementing packet buffers in switches and routers that have multi Gb/s ports. Chapter 2 will draw some comparisons between systolic and non-systolic processing techniques, and survey shared buffer and priority queue hardware implementations. It will also point out how these issues have been addressed by various investigators.

The main objective of the thesis is to develop effective VLSI design methodologies for implementing scalable buffering schemes for high-speed networking applications. The thesis addresses the design of two specific architectures: balanced parallel multi-input multi-output (MIMO) buffer, and systolic parallel priority queue (PPQ). In the following, we elaborate on these two contributions and compare them to previous related research.

Balanced Parallel MIMO Buffer: The thesis presents a methodology for the systematic design of order-preserving MIMO buffers. Using this methodology we derive scalable parallel FIFO buffer structures that can be designed to match the rate of ultra high-speed links using current memory technology that uses moderate clock rates. Chapters 3 and 4 develop the design concept and VLSI implementation for a parallel MIMO buffer structure which can be used in ultra high-speed network interfaces and similar applications. The buffer is capable of inputting and/or outputting multiple packets while maintaining their FIFO (first-in first-out) order. These two chapters focus on buffer design for fixed-size packets, such as ATM cells (see Appendix A), but the design strategy can be extended to variable-length packets. Some applications of the MIMO buffers are outlined in Chapter 3 too.

Our approach employs a systolic routing network and bank of parallel FIFO buffers to yield a load-balanced multi-input multi-output FIFO realization with increased bandwidth [8], [9]. The MIMO buffer adaptively manages the available buffer space for statistically multiplexed input traffic. Our systolic routing network provides significant advantages over previously proposed banyan/butterfly networks since the systolic network eliminates the need for parallel-prefix adders that compute packet ranks before concentrating them on the output ports. The combined use of pipelined architecture and dynamic CMOS circuits resulted in significant reduction in design complexity and substantial performance gains in speed and area. A small 4-input 4-output prototype buffer has been designed using a triple-metal/single-poly 0.5- μ m CMOS technology to demonstrate our concepts. The buffer can attain a rate of 10.6 Gb/s which is more than adequate to support a Sonet OC-192 link.

Although the idea of using a parallel arrangement of FIFO buffers to realize a faster shared FIFO buffer has been around since the inception of the Knockout ATM switch [10], our realization and use of such buffer structures differ significantly from previous approaches, as will be pointed out in the next chapters. Our design approach is truly scalable and has real advantages from VLSI perspective. Also, in our designs, load-balancing is done using arithmetic-free circuits. As it will be explained in more details in Chapter 2, typically load-balancing is done using a multistage network of adders to do packet-ranking, and routing is done by a second pass through another multistage network as was done in the Knockout switch [10], and Multinet switch [11], [12].

Systolic Parallel Priority Queue: The thesis also generalizes the priority queue concept to a *parallel priority queue* (PPQ) which can be scaled to meet the requirements of ultra high-speed links using standard CMOS technology. Chapters 5 and 6 present an area-efficient, systolic design of the PPQ for VLSI implementation. The proposed PPQ is rate-adaptive in the sense that the PPQ operates correctly even when the queue input rate and output rate are different [21]. This is an important feature because in practice the output rate of the queue is controlled by the available link bandwidth which may vary (or even become zero) independent of the packet arrival rate. This decoupling of the input and output packet flow rates is a distinguishing feature of our PPQ concept and has not been addressed in previous literature [22].

In many real applications, it is desirable to decouple the input process to the PPQ from the output process. Specifically, the PPQ outputs may be intermittently blocked from sending cells due to a link congestion. This process is usually called output rate-adaptation, and it requires significant modifications to the conventional architectures. To provide rate-adaptation capability to our PPQ, we strategically employ data steering blocks, between pipeline stages of PPQ, which are controlled by a global “go/stop” signal.

In non-systolic architectures, data insertion is normally performed over a global bus connected to the inputs of all the modules in the queue [23], as it will be explained in Chapter 2. This creates a bus loading problem, which adds to the hardware costs (buffers), and decreases the maximum operating speed of the queue clock [24]. In a systolic PQ implementation, the clock can be faster, but technology limitations can limit the clock speed. This problem is more serious when dealing with ultra-fast data links operating at 10 Gb/s (e.g. Sonet OC-192) to 40 Gb/s (e.g. Sonet OC-768) rate. In this thesis we employ a parallel processing approach to solve this scalability problem for

systolic PQs. Our approach embodies a combination of hardware design with more theoretical parallel processing concepts. The combined use of a systolic structure, and dynamic CMOS circuits facilitated the balancing of design complexity and performance gain, as will be shown in Chapters 5 and 6.

CHAPTER 2

Background & Survey

Buffering structures are integral components of packet networks. With the rapid proliferation of high speed packet-switching systems, the demand for high speed data buffers and scheduling networks has increased. Special requirements for such applications, suggest the use of customized memory-based architectures such as parallel buffers and priority queues. Before proposing efficient scalable queuing architectures to implement buffering and scheduling schemes using current VLSI technology, this chapter reviews previously proposed techniques for designing these buffers and queuing structures.

2.1 Parallel FIFO Buffers

We start this section by considering previously proposed techniques for designing parallel multi-input multi-output (MIMO) buffers that preserve the FIFO ordering of cells. It is worth mentioning that such buffers have been also called “shared” FIFO buffers in previous literature and have been used before in several switch designs, for example, see [10]-[20].

As shown in Fig. 2.1, if a high-speed buffer is constructed from multiple FIFO buffers (as discussed in Chapter 1), then a dynamic balanced distribution network (BDN) stage is needed to *pack* and *shift* the incoming cells so as to balance the load on all FIFO modules while preserving the arrival order of the cells at the same time. The buffer shown in the figure has 5 input ports and 5 output ports. The five ports are labeled a, b, c, d, and e. The labeled squares in the figure represent cells arriving in three different time-slots (1, 2, and 3), and the label of each cell indicates its arrival time-slot and arrival port. For example, the square label (2c) represents a cell arriving on port c in time-slot 2.

Typically load-balancing is done using a multistage network of adders to do packet-ranking, and routing is done in a second pass through another multistage network as was done in the Knockout switch [10], and Multinet switch [11], [12]. Note that the BDN packs incoming cells in adjacent output locations and shifts them appropriately to achieve a balanced loading of the FIFO buffers. Note also the order of cell arrivals is preserved. Fig. 2.1(b) shows the dynamic configuration of the BDN in each time-slot. In short, the BDN distributes cells arriving in the same time-slot among the FIFO buffers in a cyclic (or round robin) fashion so that the difference between the contents of any two buffers does not exceed one cell. Note, that load balancing eliminates the need for combinational concentrators and restricts the causes of cell loss to buffer overflow only.

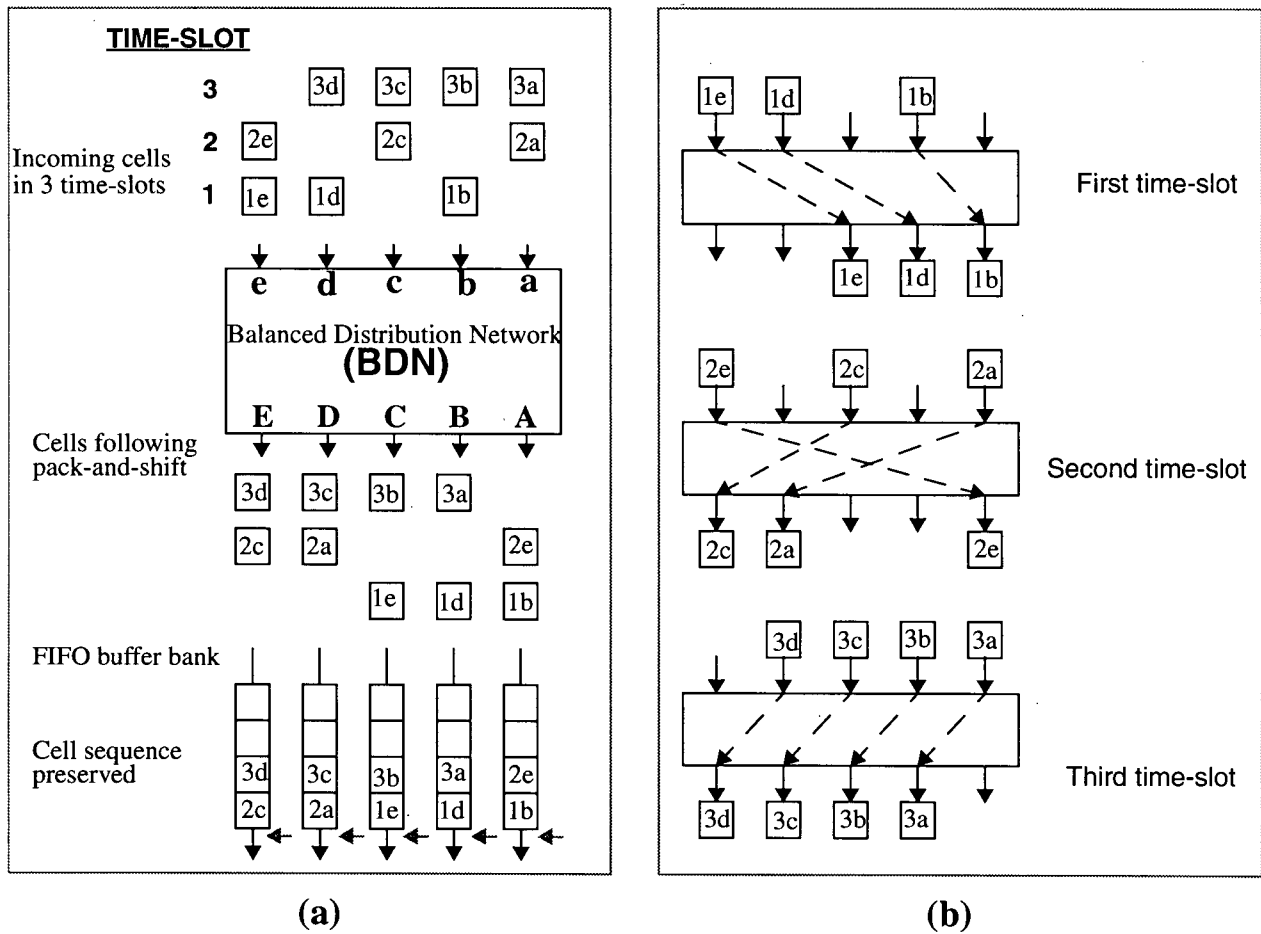


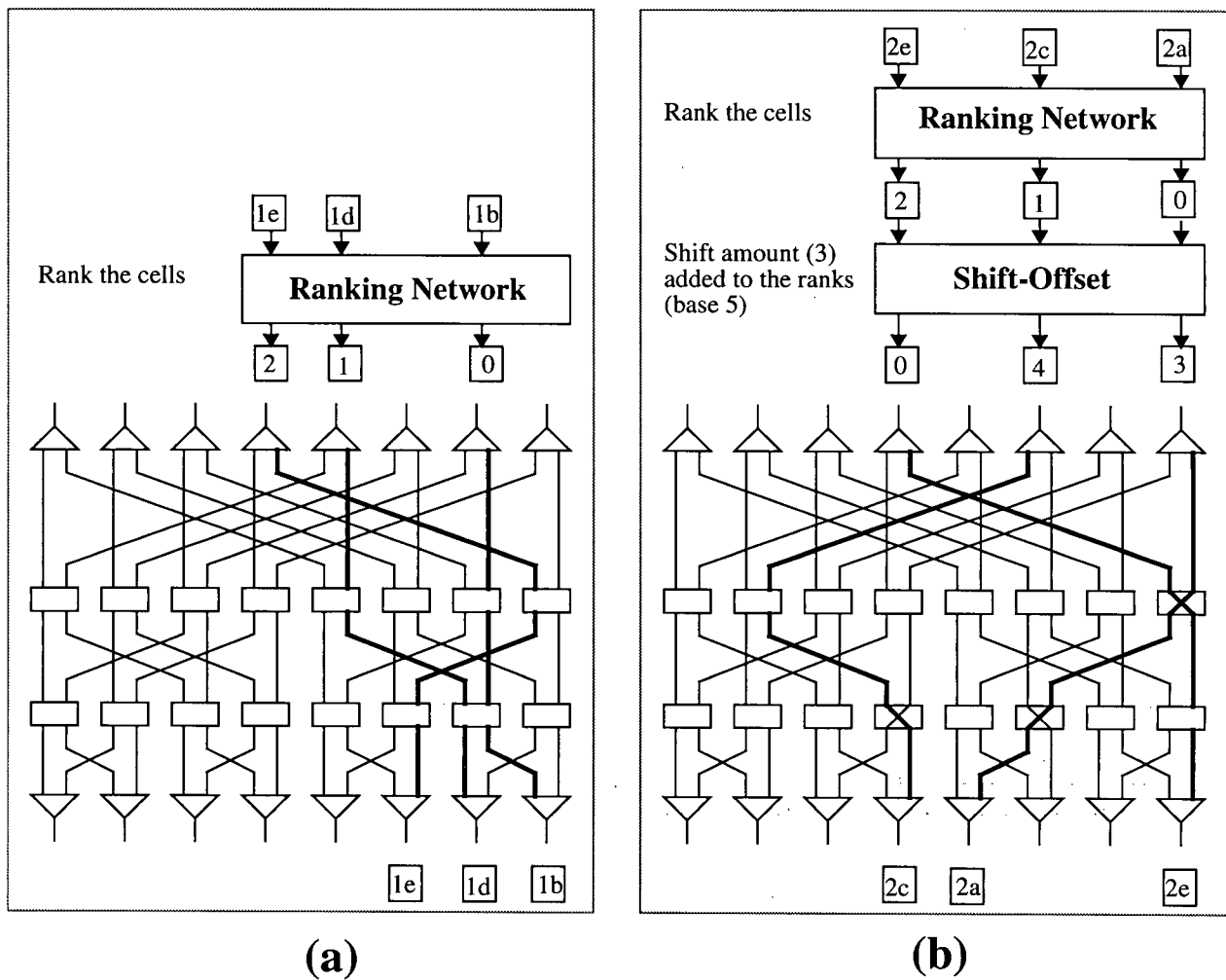
Fig. 2.1: (a) Operation of a shared-buffer that preserves the cell arrival sequence
(b) Illustration of pack-and-shift approach in three time-slots

Several schemes have been proposed to achieve the BDN function using multistage interconnection networks (cf. [25], and the references therein). To perform the pack-and-shift function, cells are first packed using either a concentrator or a routing network then shifted cyclically to distribute the packed cells evenly among the FIFO buffers at the output. Both the pack and shift func-

tions can be implemented by the well known butterfly (which is a banyan type) network [25]. The butterfly network nodes must have an arithmetic addition capability to perform parallel prefix addition for ranking the cells arriving at the same time in a given order. Ranking the cells can also be done by a low-latency adder network, such as the one described in [25].

Fig. 2.2 shows how cells can be packed and shifted (after they have been ranked) on a butterfly network for a 5×5 (5-input/5-output or 5 I/O) buffer architecture. Referring to the example in Fig. 2.2(a), the three cells labeled 1b, 1d, and 1e, are first assigned the ranks 0, 1, and 2, respectively, then packed by a butterfly network. These ranks will be used as local routing headers to route these cells to output ports 0, 1, and 2 using a greedy self-routing strategy. At the output of the ranking network of Fig. 2.2(a) the cells appear labeled by their computed ranks. Fig. 2.2(b) illustrates a situation where the cells are ranked first, then packed and shifted cyclically by 3 positions. The shift-offset function modifies the ranks of the cells, so that they can be packed and shifted by the same butterfly network as the one used in Fig. 2.2(a). The routing procedure utilizes the monotonic routing property of the butterfly to pack-and-shift input cells with no path contention.

It is clear that the above realization of a BDN adds a high degree of complexity and latency to the buffer design. This approach has been employed in the Multinet switch [11], [12]. The internal buffers in the Multinet switch implement a partial buffer (PS) sharing discipline that maintains cells in the correct order [11]. Each PS buffer in the switch is constructed using a reverse banyan network (similar to the butterfly network illustrated in Fig. 2.2), of appropriate size, that feeds cells to a bank of FIFO buffers. The reverse banyan network is equipped with a Fetch-and-Add capability that contributes to balancing the load in the bank of FIFO buffers constituting the PS



- 0 cell with rank 000
- 1 cell with rank 001
- 2 cell with rank 010
- 3 cell with rank 011
- 4 cell with rank 100

Fig. 2.2: (a) Packing ranked cells on a butterfly network
(5 I/O - first time-slot)
(b) Packing and shifting ranked cells on a butterfly network
(5 I/O - second time-slot)

buffer space. Despite claims of simplicity, the Multinet switch is actually complex and cells can experience significant delays, mainly because of the structure of the PS buffers. The Fetch-and-Add reverse banyan network employed in each PS buffer performs arithmetic as well as routing operations. Three passes of computing and routing through the reverse banyan are required before cells are finally fed to the bank of FIFO buffers of PS buffer [12]. In current deep submicron CMOS technologies, the delay in signal propagation over longer wires becomes a significant design limitation, particularly for a shared buffer architecture with large number of I/O ports. This limitation is particularly applicable to high wire organizations such as the butterfly network (or reverse banyan network) typically used in interconnection networks [25].

A somewhat different approach has been employed by the Knockout switch [10] and many of its realizations (such as the growable packet switch [13], [14], and the MOBAS switch [16]) to achieve load balancing. The knockout approach employs a lossy deep multi-stage concentration network followed by a shifter network, as shown in Fig. 2.3. The concentration is performed by N-to-L combinational tournament circuit, in which packets losing contention are knocked-out [10]. The parameter L (for the N-to-L concentrator) is selected in the Knockout switch to achieve a particular cell loss, and cell loss can be reduced by increasing L. The original Knockout concentrators have depth proportional to N, and their complexity as well as delay can be substantial for large N.

The PINIUM switch [18] is basically another Knockout switch with the concentrators modified to perform priority routing of cells. Each such concentrator is constructed from modified bitonic sorters and mergers [18], [25]. An N-to-L concentrator is constructed iteratively from L-input bitonic sorters and 2L-to-L bitonic mergers. As in the Knockout switch, the value parameter L determines the cell loss ratio in the memoryless routing and concentrating part of the switch. The

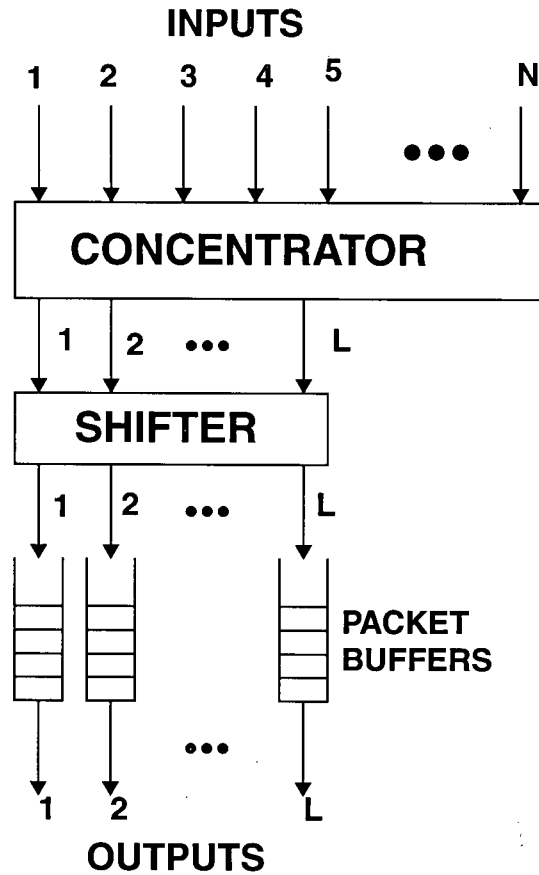


Fig. 2.3: Structure of a shared buffer in the Knockout switch

main weakness of the switch is the use of sorters in the priority concentrator blocks. The sorters can incur significant routing delays in large scale versions of the switch. The delay caused by the use of sorting in the priority concentrators becomes unacceptable at high link speeds. It is also not clear whether the use of priority concentration can lead to unfairness.

In the Helical switch [19], which is a self-routing multistage banyan network with output as well as internal buffers, each internal buffer consists of a non-blocking concentrator feeding a bank of FIFO buffers. Each non-blocking concentrator is constructed using a running adder and a two-

dilated reverse banyan network of appropriate size. Correct cell sequence is maintained by the concentrators. A concentrator routes the incoming cells to the next available concentrator output in round-robin order. At the concentrator output dummy cells are generated and inserted into the proper FIFO to maintain the integrity of cell ordering. The dummy cells generated by one stage of concentrators are not routed by the next stage of concentrators, rather they are used to create idle cycles that keep later cells from advancing past earlier cells. The concentrators employed in each stage of the switch can cause significant cell delays. This is caused by the insertion of dummy cells in internal FIFOs as well as the delays caused by the running adder. In other words, the delays will grow as the number of stages increase in large switches and the insertion of dummy cells in internal buffers will aggravate this problem. Additional hidden delays are caused by the running-adder operations in the reverse banyan networks employed in every stage.

One of the main contributions of this thesis is a very efficient implementation of the pack-and-shift function using a dynamically reconfigurable systolic interconnection array which is managed by two simple distributed controllers. The systolic architecture essentially realizes a pipelined switch array with embedded load-balancing and order-preserving capabilities (see Section 2.2 for definition of systolic architectures). To our knowledge, the work on parallel FIFO implementation has been done mainly under the concept of shared output buffer design in high speed packet switches as reported in [10]-[20]. Our work improves on these techniques by proposing an arithmetic-free, pipelined, load balancing network that forwards cells to a parallel MIMO FIFO bank [8]. Our design approach is based on a three-stage VLSI architecture as shown in Fig. 2.4 [9]. The first stage (BDN) is a systolic array of simple switches which performs load-balancing and ensures proper cell ordering. The second stage is a bank of single-input single-output FIFO buffers which operate in parallel. We enhance the flexibility of the design by adding a rotating multi-

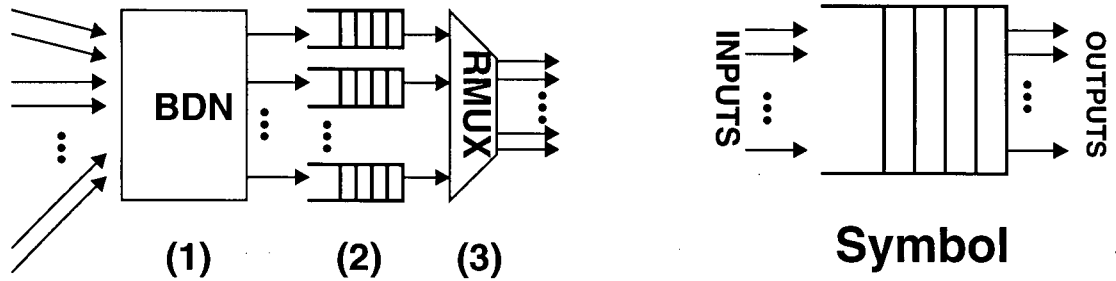


Fig. 2.4: MIMO buffer and its symbol

plexer (or RMUX) at the output to adapt the buffer rate to that of the output link.

In the system design of the MIMO buffer the following objectives have been pursued: 1) maximum buffer space, 2) minimal routing delay, and 3) packet arrival in the correct order [26]. In Chapters 3 and 4, the detailed structure of the above realization is reported. Here, it is worth mentioning that because of the uniform structure of the proposed MIMO buffer, it has the advantages of simple synchronization and expansion due to the modular structure of the systolic switch array. It is a truly scalable architecture, and amenable to simple VLSI implementation. As shown in Fig. 2.5, if a rotating multiplexer (RMUX) is applied in the last stage of the parallel buffer, the buffer can also be employed as an ATM access point or as a buffer concentrator [26]. In addition to their use in high speed interfaces, these buffers can be utilized in a number of switch architectures [27]. Some of these applications are addressed in Section 3.5.

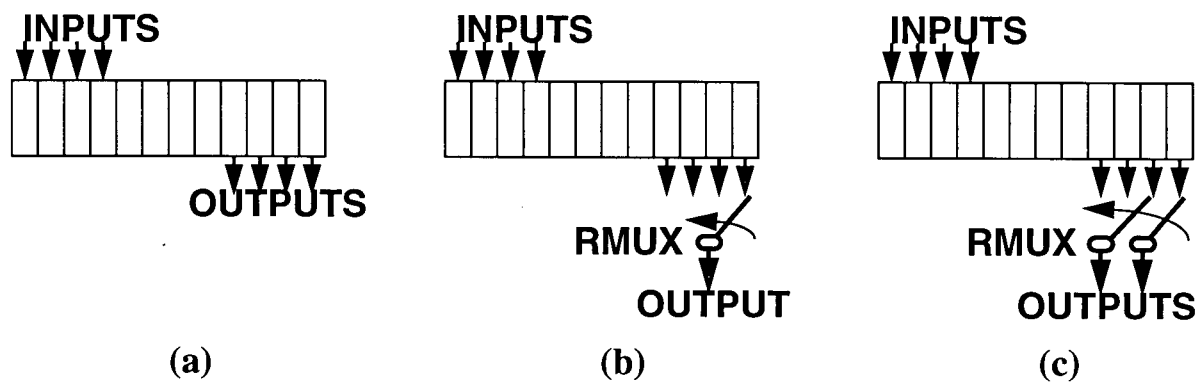


Fig. 2.5: (a) Parallel buffer, (b) ATM access point, and (c) Buffer concentrator

2.2 Priority Queues

As mentioned in Chapter 1, priority queues are essential in implementing link-scheduling algorithms. Priority queues can be implemented in software or hardware. Due to the high speed at which the networks operate, a hardware priority queue is needed to transmit packets at link rates. For example, in a 10 Gb/s ATM network, an ATM cell can be transmitted every about 42 ns. In a worst-case scenario the priority queue must determine the next highest priority cell (dequeue operation) every 42 ns, while being able to accept new cells (enqueue operation) from all incoming links within the same time unit. Software solutions are typically not fast enough to keep up with this packet transmission rate [24]. On the other hand, a scalable hardware solution can operate close to the operating speeds of the link. Also, a hardware solution can overlap enqueue and dequeue operations with packet transmission to avoid wasting link bandwidth. In hardware implementations, a sequencer determines the departure sequence for the packets that are stored in a buffer bank. In general, hardware solutions can be categorized in two families: systolic and non-systolic.

Systolic Arrays: The term systolic array, which became popular in the early 80's, refers to certain processing structures with regular cellular organization and a large number of neighbouring processing elements with same function [28], [29]. They communicate via direct physical interconnections only with a limited number of adjacent processing elements to exchange data. Therefore, interconnections are local and in a regular pattern. VLSI technology has made one thing quite clear: simple and regular interconnections have substantial advantages over complicated interconnections. The original motivation for proposing the systolic array concept was their high efficiency for implementing VLSI systems [28]. Being highly regular and modular structures, they do not suffer from the VLSI design problems which are typical of other systems. Fitting the principal constraints imposed by VLSI design and technology, the systolic systems should, in principle, be able to profit from VLSI as a potential implementation means [29].

A systolic priority queue architecture typically consists of a large number of similar processing elements which are interconnected to exchange data. In fact, many processors work in parallel and the data is processed and transferred by pipelining (as will be described in Chapter 5). The first systolic priority queue was proposed by Leiserson [30]. In recent years, systolic [31], and non-systolic [33], priority queues were proposed for high-speed network buffers.

In non-systolic architectures [34], data cell insertion is normally performed over a global bus connected to the inputs of all the modules in the queue. A priority queue always serves the packet that has the highest priority. It determines the departure sequence for the packets that are stored in a buffer bank in such a way that higher priority values are always at the right of lower priority values so they will be accessed sooner. Each entry in the priority queue contains the pair (P, A) , where P is a priority field and A is an address pointer to a packet in a packet memory. In a non-systolic hardware structure, as shown in the example in Fig. 2.6, these pairs are stored in the sequencer in

a descending order. Now, let us consider an example of a newly arrived data cell with $P=5$ and $A=A_n$. This pair is broadcast to all modules. All pairs on the right of A_j , including the A_j itself, remain at their positions while others shifted to the left, and the vacant position is replaced with the new pair $(5, A_n)$. When the PQ is full, the priority field at the leftmost position of the sequencer is compared to that of the newly arrived data cell. If the new's one is smaller than it, the pair (P, A) at the leftmost position is pushed away from the sequencer as the new pair is inserted in the sequencer.

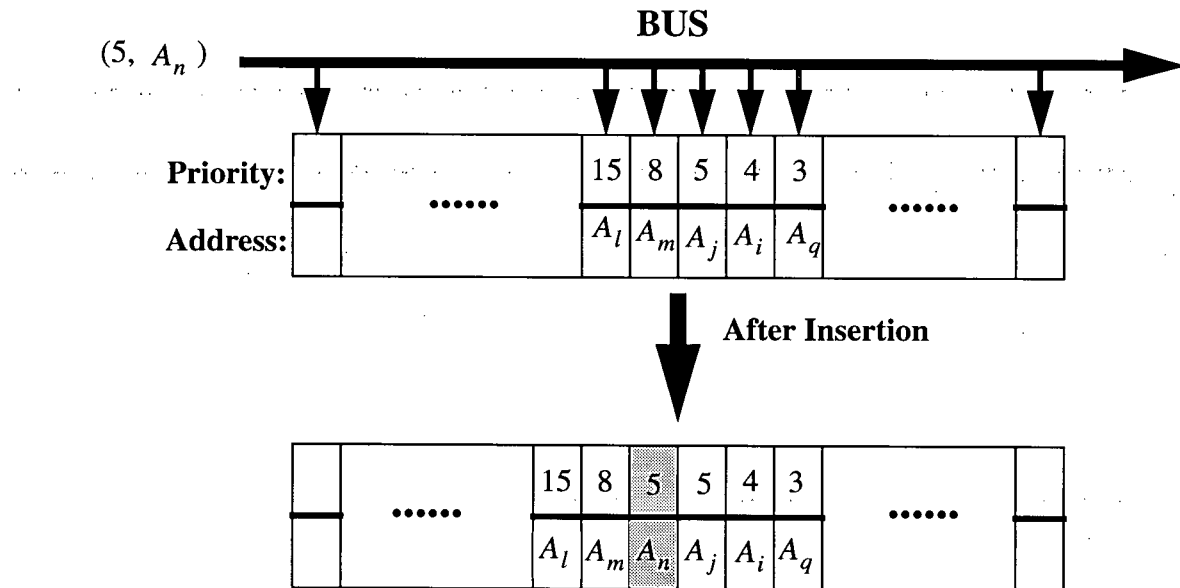


Fig. 2.6: Inserting a new arrival into a non-systolic PQ

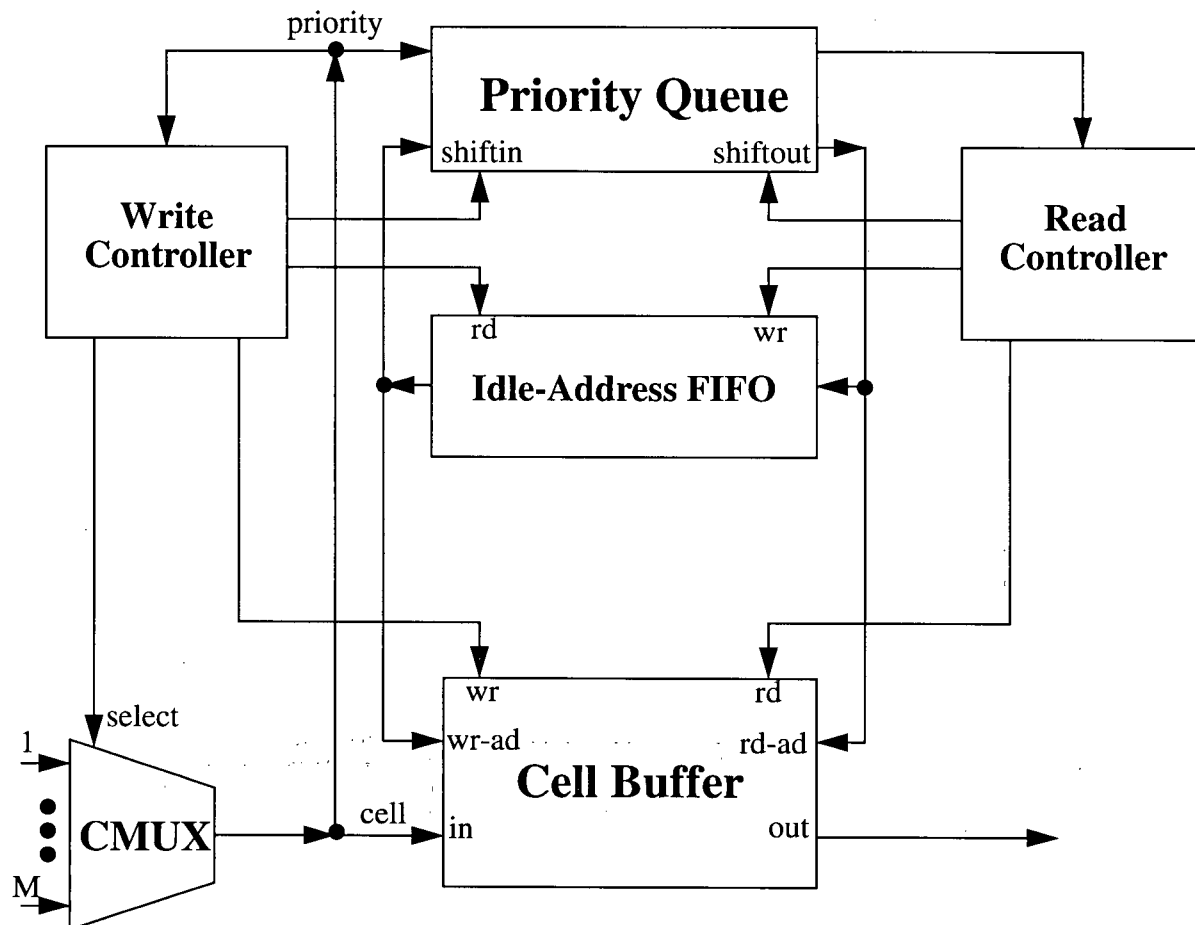


Fig. 2.7: Buffer manager structure

Fig. 2.7 shows a typical buffer manager structure for an ATM switch (see Appendix A) that uses a priority queue or sequencer [32]. The buffer manager consists of a cell multiplexer (M inputs and one output), a cell buffer, an idle-address FIFO, read and write controllers, and a priority queue. The cell multiplexer (CMUX) located after a switch fabric receives and multiplexes up to M cells from the switch fabric, where M is the number of cells that are routed through an ATM switch network and arrived at an output port. Cells from the inputs are multiplexed onto the cell buffer at idle-addresses retrieved from a FIFO (only valid cells are stored in the cell buffer). The read/write

controllers generate proper control signals for all other functional blocks. The buffer manger always serves the cell that has the smallest priority value of the P first and they will be accessed earlier by the read controller. In the buffer manager structure, the sequencer determines the departure sequence for the cells that are stored in the cell buffer.

In the non-systolic architecture presented in [34], which is one of the only few dedicated priority queue designs to be implemented in silicon, a bus structure has been used to access all of the buffering modules. At each module the data cell on the common bus is compared to the data cell in the module. However, because data insertion is performed over a global bus, the queue cannot operate at very high speed. As seen in Fig. 2.6, before any decision can be made by each module during an enqueue operation, the new entry must be present at the inputs of all modules. At the VLSI level, the bus must be routed all over the queue. This creates a bus loading problem, which adds to the hardware costs (buffers), and decreases the maximum operating speed of the queue clock [24].

The design presented in [36] is also a priority queue controller designed for use in ATM switches. The chip provides no data storage, but rather maintains a separate logical FIFO queue in external RAM (random access memory) for each of its four supported priority levels. Although this simplifies comparator logic, maintaining separate FIFO's is impractical in the presence of a larger number of priority levels. Moreover, since ATM switches have equal input and output data rates, this chip was not designed to accommodate different read and write clock speeds.

A binary tree comparator architecture was presented in [37] whose output is the highest-priority entry among those in a storage. The architecture consists of a storage block and a comparator tree. A feedback mechanism is also applied in the design to remove the output of the tree from storage. Problems with this design include bus loading problem for distributing the new entry to each stor-

age element in the storage block, and increased dequeue time resulting from an increase in depth of the comparator tree. A possible solution to the increased dequeue time is to pipeline the comparator tree operation to reduce the clock period and increase performance [24].

Chapters 5 and 6 present an alternative design technique using systolic design concept, which can achieve higher speed, to meet the requirements of ultra high-speed links using standard CMOS technology [22]. The proposed PPQ has several applications in implementing real-time fair schedulers or buffer management algorithms in packet routers.

CHAPTER 3

Parallel FIFO Buffers for Gigabit Packet Networks

3.1 Introduction

This chapter presents a methodology for the systematic design of order-preserving multi-input multi-output (MIMO) buffers for large scale or ultra fast packet switches. It focuses on buffer design for fixed-size packets, such as ATM cells, but the design strategy can be extended to variable-length packets. These buffers are capable of inputting and/or outputting multiple packets while maintaining their FIFO (first-in first-out) order, and can be used in ultra high-speed network interfaces and similar applications. Our approach employs a systolic routing network and bank of

parallel FIFO buffers to yield a load-balanced MIMO FIFO realization with increased bandwidth [8]. The MIMO buffer adaptively manages the available buffer space for statistically multiplexed input traffic. Our systolic routing network provides significant advantages over previously proposed banyan/butterfly networks since the systolic network eliminates the need for parallel-prefix adders that compute packet ranks before concentrating them on the output ports [9]. Also, in our designs, load-balancing is done using arithmetic-free circuits.

Using this methodology we derive scalable parallel FIFO buffer structures that can be designed to match the rate of ultra high-speed links using current memory technology that uses moderate clock rates. Our design approach is truly scalable and has real advantages from a VLSI perspective. As it was described in Chapter 2, if a high-speed buffer is constructed from multiple FIFO buffers then a balanced distribution network (BDN) stage is needed to *pack* and *shift* the incoming cells so as to balance the load on all FIFO modules while preserving the arrival order of the cells at the same time. Our design approach is based on a three-part VLSI architecture as shown in Fig. 3.1. The first part is a systolic two-dimensional array (folded crossbar) of simple switches which performs load-balancing and ensures proper cell ordering. The second part is a bank of single-input single-output FIFO buffers which operate in parallel. The last part is a rotating multiplexer (RMUX) which adapts the buffer output rate to the output link rate. Note that the folded crossbar is used for load balancing in a single MIMO buffer and not for general routing of packets.

An example of how MIMO buffers can be used in a multistage switch is shown in Fig. 3.1. Note that a MIMO buffer spreads its cells equally among its output links thus achieving load balancing on the links as well. Each bundle of links connected to the output of a MIMO buffer can be viewed as a “super link” with aggregate bandwidth equal to the total bandwidth of its individual links. The proper operation of multiple stages of MIMO buffers is possible because of their order

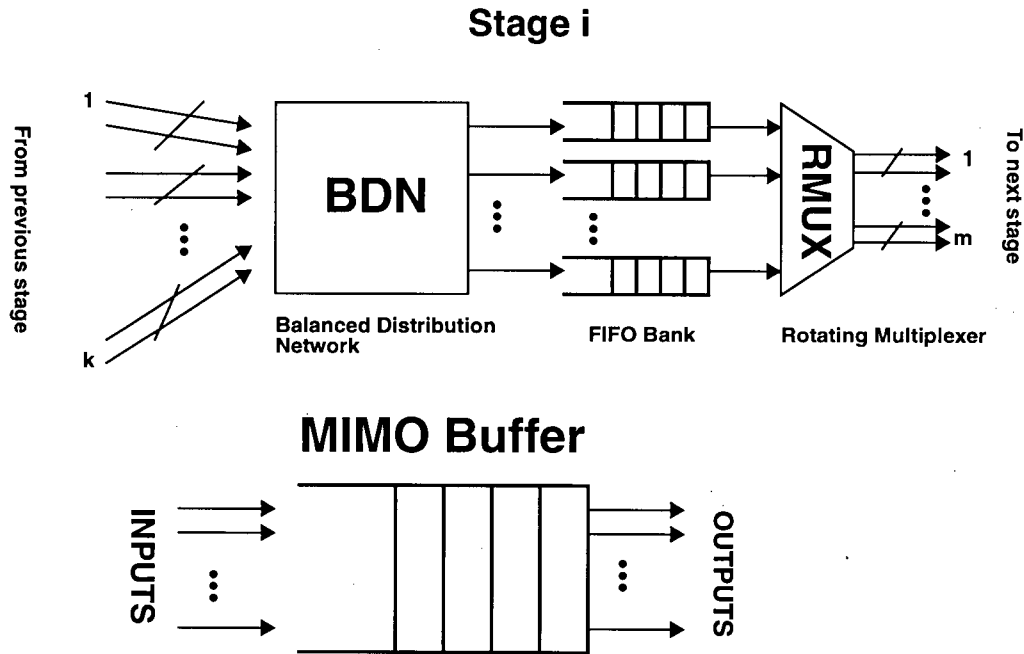


Fig. 3.1: MIMO buffer in a multi-stage switch and its symbol

preserving capability.

The rest of the chapter is organized as follows. The next section outlines the basic operation of the MIMO buffer. The buffer architecture and its main components are presented in Section 3.3. Section 3.4 describes the operation of a balanced distribution network (BDN) and illustrates its pipelined operation and dynamic reconfiguration for load balancing. In Section 3.5 some applications of the MIMO buffer are discussed, followed by a summary in Section 3.6.

3.2 MIMO Buffer Design Principals

This section outlines the main principles underlying the VLSI design and performance of MIMO buffers with the aforementioned properties. Our design approach exploits pipelining, parallelism, distributed control, and dynamic reconfiguration, as key techniques for achieving scalability and optimal buffer utilization. Pipelining increases switch utilization, and is used as the main tool for maximizing hardware utilization. Both data and control paths are pipelined yielding a true systolic design with short nearest-neighbor interconnections which is ideal from VLSI design perspective. Parallelization of the data or packet path is employed as the main tool for achieving significant speed gains in a scalable manner. However, parallelizing the buffer structure may lead to underutilization of memory and wire resources and may also cause out-of-sequence packet delivery as was pointed out earlier, hence the need for load-balancing and order-preserving control in the MIMO buffer.

3.2.1 Basic Operation of the MIMO Buffer

The main contribution of this chapter is a very efficient implementation of the pack-and-shift function using a dynamically reconfigurable systolic interconnection array which is managed by two simple distributed controllers. The systolic architecture essentially realizes a pipelined crossbar switch array with embedded load-balancing and order-preserving capabilities. Systolic crossbar interconnection networks have better composite VLSI performance than other interconnection networks that use higher wire area such as the butterfly and omega networks [25], especially in submicron CMOS technologies. Crossbars have the advantage of employing short wire segments only which allows the use of faster clock rates and achieves higher utilization of silicon area.

The proposed folded crossbar network is dynamically reconfigurable in the sense that the reconfiguration is achieved by locally adjusting the switch position within each switching element based on single-bit information in the cell headers. Each switch element has a set of locally controllable switches which enable its I/O ports to be connected internally in various configurations. Switch control provides for connection autonomy in the sense that different switch elements can select different configurations based on local decisions made within the element. In the following section, the structure of this realization is described.

3.2.2 The MIMO Buffer Structure

The proposed MIMO buffer consists of a balanced-distribution network (BDN) feeding a bank of FIFO buffers, as illustrated in Fig. 3.2. A rotator multiplexer (RMUX) can be attached to the outputs of the buffer bank to decouple their rate from the output link rate, as will be explained below. Before proceeding to architectural details, it is useful to identify the main design parameters that can be used to specify the MIMO buffer architecture based on aggregate bandwidth requirements and the clock rate of the implementation technology. The main design parameters are:

- k : the number of FIFO buffers used and also the number of input links to the MIMO buffer assuming a constant data-path width of w bits (wires) per link,
- m : the number of output links of the MIMO buffer assuming a constant width of w bits (wires) per link, and
- r : the rate of each input (or output) link.

Proper values for the above parameters are determined based on two key restrictions:

- R : the aggregate link-rate (in bits-per-second) at the input of the MIMO buffer, and
- c : the clock-rate of the MIMO buffer circuit.

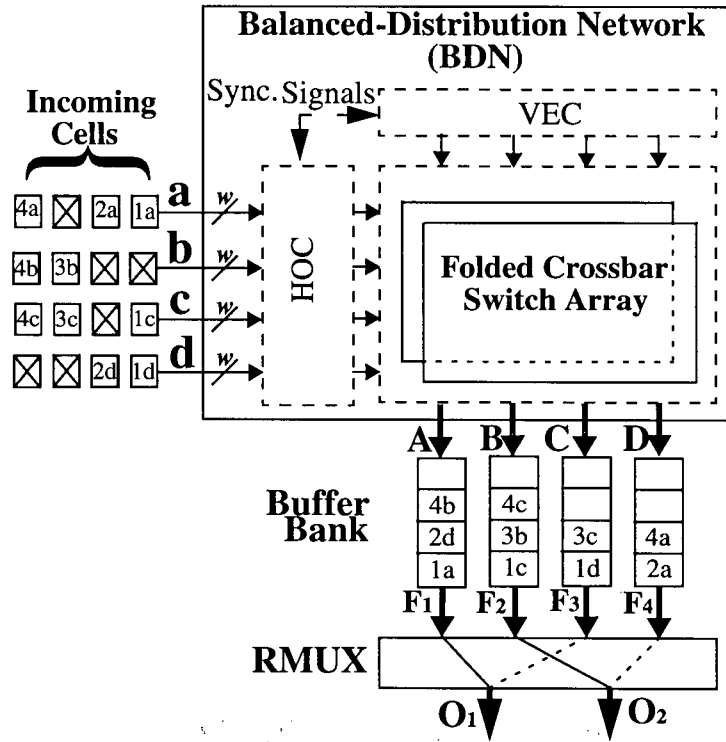


Fig. 3.2: Internal structure of the buffer

Note that c is a parameter that depends mainly on the technology used. Because of the high-degree of pipelining used in our design, c will be determined mainly by the access-time of the FIFO buffers, since all other components have delays much shorter than that.

Given a FIFO buffer whose bit rate is r and clock rate is c , then $r = c \cdot w$. To match an aggregate input link rate of R , k input links and k FIFO buffers must be used, where $k = \lceil R/r \rceil$. The RMUX allows further design scalability by decoupling the FIFO rate from the output link rate. This is an important property of the design since it is normally the case that the aggregate input rate is higher than the aggregate output rate in the buffers of a packet switch. For example, if the number of input links is an integer multiple of the output links, i.e. $k = n \cdot m$ for some integer

n , then an RMUX with k inputs and m outputs can be used to forward packets from the FIFO buffers onto the output link. The RMUX samples the FIFO outputs m at a time, and visits all FIFO outputs in n cycles. Essentially, the RMUX enables the MIMO buffer to operate as a buffered concentrator which does not incur any packet loss unless the FIFO buffers are full.

As an example, consider the design of a MIMO buffer with an aggregate input rate of 10 Gb/s and an output rate of 5 Gb/s, when the data-path width is $w = 8$, and the clock rate that can be supported by the CMOS technology at hand is 320 MHz. In this case, the input link-rate is 2.5 Gb/s, and so $k = 4$, i.e. the MIMO can be designed using a 4×4 folded crossbar switching array, 4 FIFO buffers, and a 4-to-2 RMUX.

The scalability feature of BDN can be used to construct very high bandwidth packet memories from standard FIFO memory units of moderate speed. Specifically, the BDN allows a packet memory of bandwidth $2(k \cdot r)$ to be realized from k memory units of bandwidth $2r$ each [9]. The BDN also ensures equal utilization of the memory units under all traffic patterns. In the following section we explain the BDN structure in more detail.

3.3 The Balanced Distribution Network (BDN)

The BDN is the primary routing engine in the MIMO buffer, performing both pack and shift operations. Additionally, the BDN maintains the state of the last shift operation, which indicates the position of the last FIFO buffer to which a cell has been forwarded. Maintaining this state is essential for the load balancing operation. The 8×8 BDN shown in Fig. 3.3, is comprised of a folded crossbar simple switching elements (SWTs), a horizontal controller (HOC), and a vertical controller (VEC). For FIFO operation, the HOC and VEC are constructed from simple networks

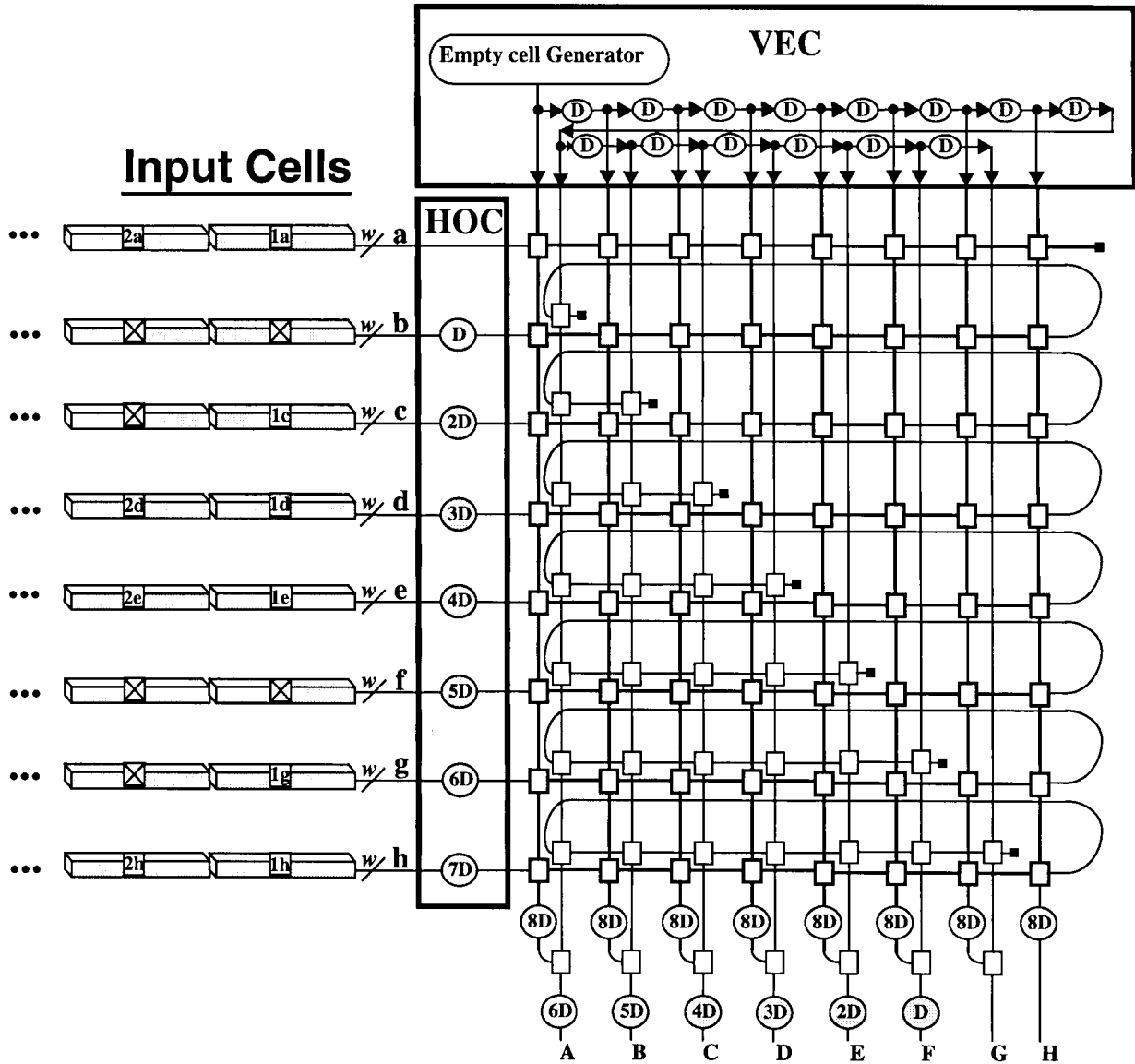


Fig. 3.3: Internal Structure of an 8x8 BDN

of delay elements for the purpose of synchronizing the arrivals of cells and control signals at the appropriate switching elements. In Fig. 3.3, SWTs are represented by squares while delay chains are represented by circles labeled by the amount of delay, e.g. a circle labeled 5D represents a delay chain of length 5, where a unit of delay corresponds to one clock period.

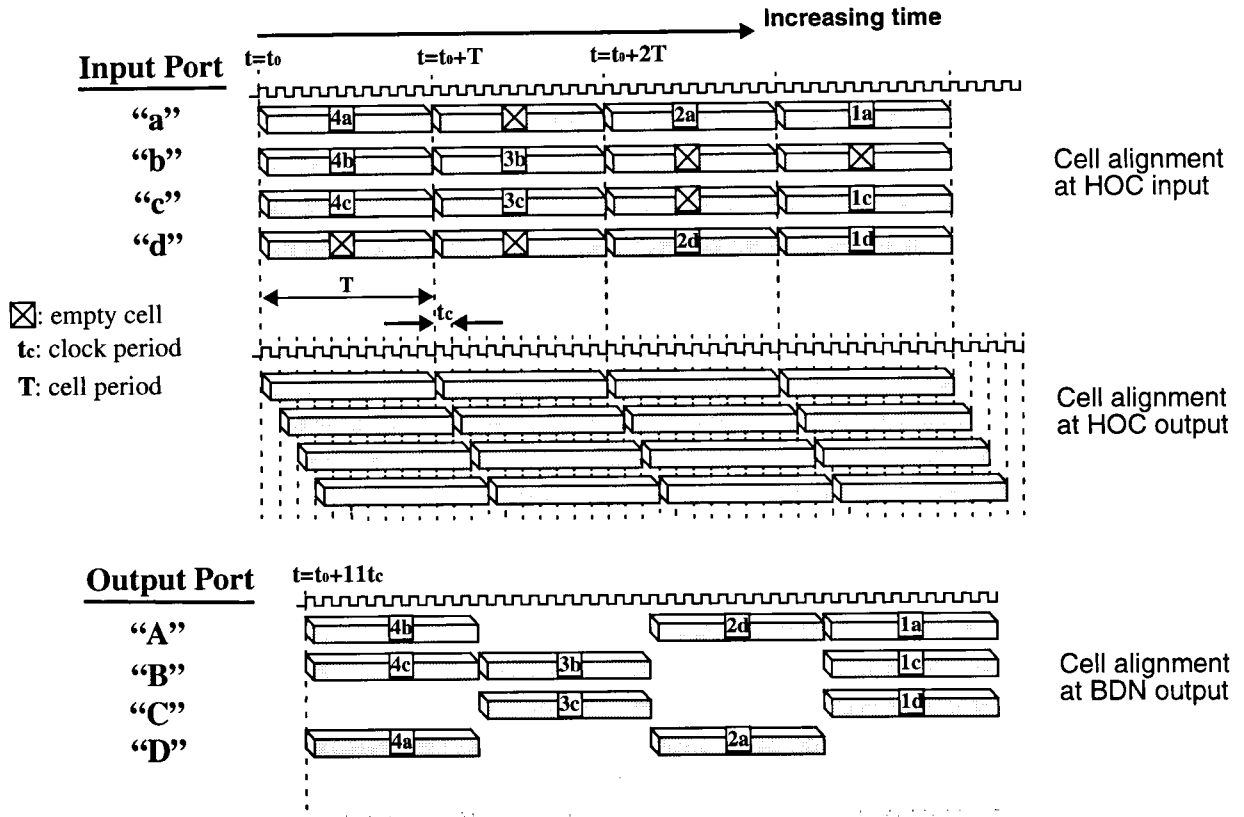


Fig. 3.4: Input/Output port alignment of the BDN (crossed squares indicate empty cells)

The BDN operates in a synchronous time-slotted manner, with each time-slot equivalent to one cell (fixed-size packet) time over a w -bit link. Cell arrivals are always aligned with the beginning of time-slot for all BDN inputs. Within a time-slot, a BDN receives either a valid or an invalid (or empty) cell. We assume that a special bit in the cell header is used to indicate whether the cell is valid or not. Such empty cells will be removed of the cell stream during the packing operation in the BDN. Note that before entering the BDN, the cells are passed through the HOC which routes incoming cells through proper delay stages before they enter the BDN.

Fig. 3.4 explains the timing alignments of the cells at the input of the HOC for the 4×4 BDN

example of Fig. 3.2, also at the input of the folded crossbar (i.e. the output of the HOC), and at the output of the crossbar before being written into the FIFO buffers. Note the difference between a clock period which defines one *time-unit* versus a cell period which defines one *time-slot*. Basically, the incoming cells (whether valid or empty) are always aligned at the HOC inputs. The HOC then “stagger” the cells before entering the crossbar. This is a necessary step because cells in the crossbar can turn-around and rendezvous with one another at a SWT as will be explained later. However, before leaving the crossbar, the cells are aligned back again.

As mentioned above, the folded crossbar network consists of simple 2-input 2-output switching elements. Cells enter a SWT from its north port (i.e. port D_N) and/or from its west port (i.e. port D_W). Based on its local state and the information available at its input ports, a SWT will steer the incoming cells to the proper output ports. A SWT has a set of locally controllable transmission circuits which enable its four I/O ports to be connected internally in various configurations.

As shown in Fig. 3.5, each SWT can be in one of two modes representing its local state: a *cross* (or *unswitched*) mode and a *toggle* (or *bar*) mode. Normally, the SWT is in the cross mode, i.e. cells from its north (west) port are routed to its south (east) port. Fig. 3.5 also shows the four possible configurations of a SWT based on the type of simultaneous cell arrivals on its two inputs. Note that empty cells are always routed eastwards where eventually they will be discarded (the small solid black squares in Fig. 3.3 and other figures in this chapter indicate a cell-discard port). On the other hand, valid cells are always routed southwards, except when both cell arrivals at a SWT are valid cells. In this case priority is given to the valid cell from the north port, and the cell from the west port is propagated eastwards until it is eventually switched to a south port. Other possible states of the SWT will be discussed under dynamic load balancing on the BDN in the

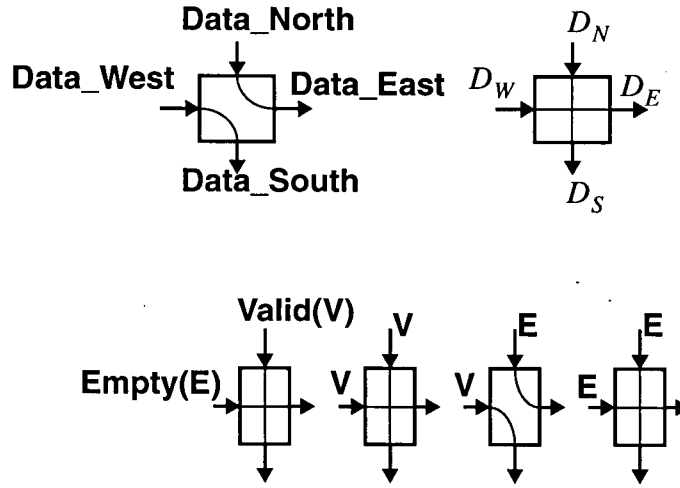


Fig. 3.5: Two states of the switch element

next section.

The horizontal controller (HOC) synchronizes the entry of data cells, valid bit indicator and cell length signals into the folded crossbar switch array. It ensures that cells are properly aligned when they arrive at each switch element as shown in Fig. 3.4. The HOC function is realized by instantiating some *delay elements* as illustrated in Fig. 3.3. In Fig. 3.3, and other figures in this chapter, we use the symbol D to denote a basic clocked delay element. Delay chains in the BDN are represented by circles labeled by a multiple of D (e.g. $5D$) indicating the length of the delay chain. The basic delay elements ensure the integrity of the wavefront at any switch element. A delay element D is basically a clocked register of width w bits ($w=8$ is assumed throughout this chapter). For proper operation of the BDN, we require such registers to be of the master-slave type. All SWTs and delay elements are triggered by the same clock. *Delay element* nD can be defined as a register with a delay equal to n clock periods [8].

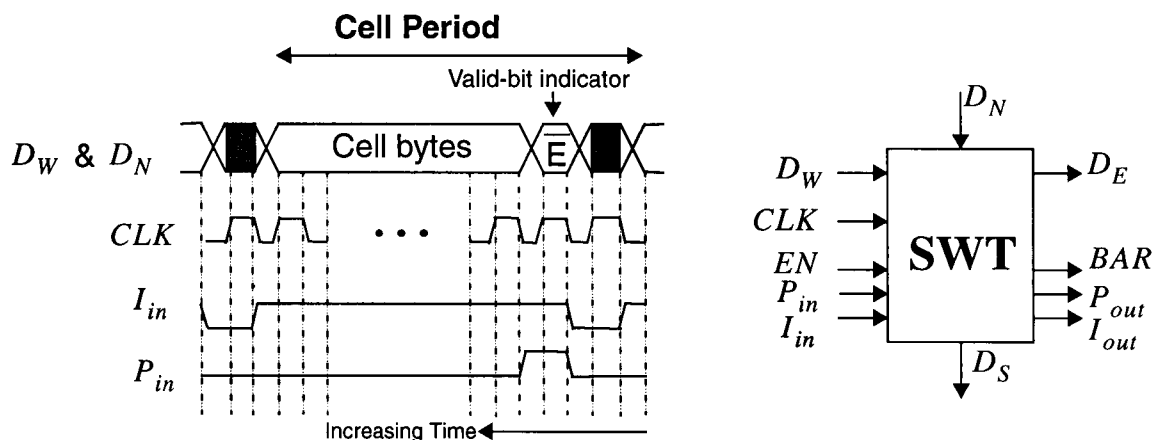


Fig. 3.6: I/O diagram of the SWT and cell format

Timing Details for a SWT: Before moving on to further discussion of the BDN operation, it is important to clarify the timing details for a SWT during normal operation. A more detailed diagram of the SWT and its data input signals format are shown in Fig. 3.6. In addition to its 4 ports (D_W , D_N , D_E , and D_S), a SWT has a clock signal (CLK), an enable signal (EN), and 5 additional control signals: two input signals I_{in} , and P_{in} , and three output signals I_{out} , P_{out} , and BAR . The I_{in} signal is used to indicate the beginning of a cell time-slot, while the P_{in} signal is employed to determine whether two cells arrive at the D_W and D_N ports are valid or empty cells. BAR is a special signal which is propagated among SWTs in the same column to maintain certain state information, as will be explained in the next section. The correct operation of a SWT assumes that two byte-aligned cells arrive simultaneously at the D_W and D_N ports and also synchronize with the I_{in} and P_{in} signals.

P_{out} and I_{out} are two delayed versions of I_{in} and P_{in} . As demonstrated in the last section, ATM

cells are propagated in the SWT array similar to a wave propagating along the diagonal direction toward the bottom right corner. The control signals which communicate between adjacent SWTs ease the synchronization problem. This requires the same phase to the signal arriving at each SWT. For this reason, the P_{in} and I_{in} signals are applied from the top left of the SWT array, and each SWT distributes these signals through the P_{out} and I_{out} output signals to its east neighbor with the exception of the SWTs in the first column of the *square array* which disperse these two signals to both their east and south neighbors. These signals have to pass through delay elements in each SWT of the architecture as will be shown later. In order to increase speed and minimize the silicon area of the switch elements, a modified version of domino circuits [38] has been employed (as will be explained in Chapter 4). The operation of the BDN will be discussed in the next section.

3.4 Dynamic Load Balancing on the BDN

This section provides additional details on the folded crossbar array architecture and how it performs load balancing on the FIFO buffers. We will start by illustrating the operation of an “unfolded” version of the folded crossbar array, then provide a more formal description of the folded crossbar architecture and control signals.

Consider the simplified 4×4 BDN architecture shown in Fig. 3.7. The crossbar switching array consists of a primary (square) array and a secondary (triangular) array of SWTs, and both arrays are arranged in an unfolded configuration. In the topologically equivalent folded crossbar version, the secondary array is overlaid on the primary array so that the SWTs in column i of the primary and column i of the secondary array are adjacent.

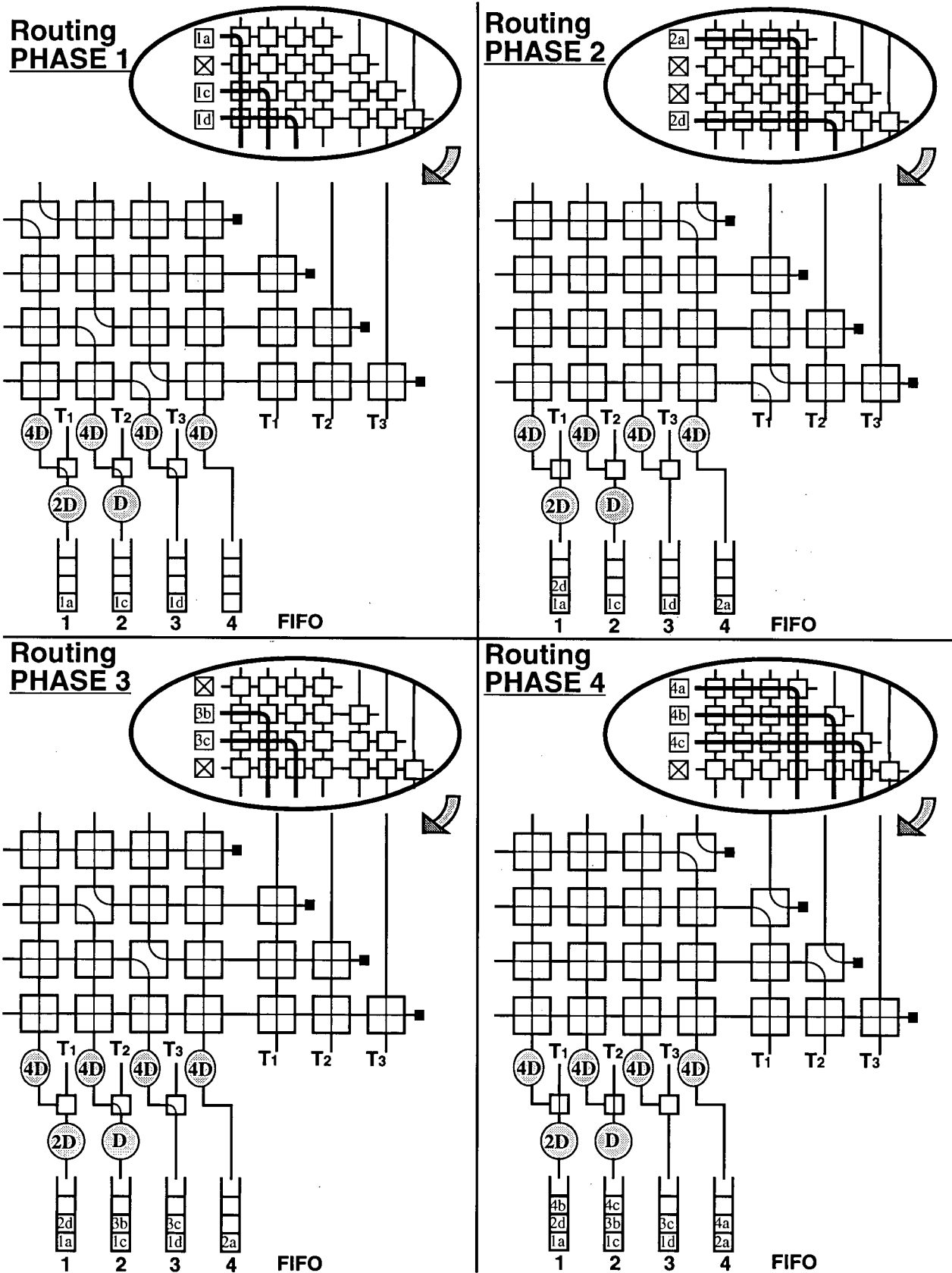


Fig. 3.7: Four routing phases of the buffer (configuration of the SWTs in the example of Fig. 3.2)

Fig. 3.7 illustrates the load-balancing action of the BDN under four waves of input cells according to the arrival pattern of example of Fig. 3.2. For simplicity, we suppress unnecessary details of the pipelined operation and explain the operation in terms of routing phases. In a *routing phase* all the cells arriving in the same time-slot will be routed completely through the crossbar array, packed, shifted, and stored in their destination FIFO buffers. A new routing phase is always started in the primary array. In cases where the shift operation results in cells being pushed out of the rightmost column of the primary array, the secondary array will complete the routing for such cells.

As shown in Fig. 3.7, the first routing phase will use the first three columns of the primary array. We say a column has been “used” in a routing phase if one of its SWTs has been set to the toggle mode. When a SWT is in the toggle mode, it will broadcast its status to all SWTs in the same column (and to the VEC unit) over a special control line. This status is maintained by a special “used” flag in each SWT. Therefore, in our example, the “used” SWTs can easily identify their status through the “used” flag (i.e. output signal *BAR* in the actual SWT circuit). At the end of the first routing phase, the three incoming cells will end up in FIFO buffers 1, 2, and 3.

In the second routing phase, all SWTs in the first three columns (i.e. columns used in the previous phase) will be set to the cross configuration. Therefore, any incoming packets in the second phase will bypass the first three columns of SWTs of the primary array and routing towards the FIFO buffers will start at column 4 of the primary array as desired. The two incoming packets in the second phase will be routed through column 4 of the primary array and column 1 of the secondary array, and will end up in FIFO buffers 4 and 1 respectively. Once the last column of the primary array has been used (column 4 in this example), the next phase always involves transferring the state of the columns used in the secondary to the corresponding columns in the primary array, and routing starts from the next available column in the primary array. In other words, we never start a

new routing phase at a column in the secondary array.

The state transfer is carried out by the VEC unit in a straight forward manner. Basically once a SWT in some column j ($j = 1$ at the third routing phase of our example) of the secondary array has been set to toggle mode, its column will be marked as “used” as was explained before. In the next routing phase the status of columns 1 through j of the primary array is set to “disable” to mirror the state of the corresponding columns in the secondary array. In the example of Fig. 3.7, we have $j = 1$, and the third routing phase will start at column 2 of the primary array as shown. Then, the fourth routing phase will begin at column 4 of the primary array as shown, i.e. $j = 3$ at the fourth routing phase of the example. Note that last column (i.e. column N) of the primary array is always set to “enable” (in all routing phases). Also, it is important to emphasize that a balanced load on the FIFO buffers is always maintained at the end of each routing phase.

The details of the structure of a folded crossbar array are shown in Fig. 3.8. Note that the primary array consists of N^2 SWTs arranged in N columns and N rows, with each switch element $S_{i,j}$ (where $i, j = 1, 2, \dots, N$) connected to each of four nearest-neighbor SWTs through its four ports. The secondary array consists of $N((N-1)/2)$ switch elements $T_{i,j}$ (where $i = 2, 3, \dots, N$ and $j = 1, 2, \dots, N-1$) arranged in a triangular plane with $N-1$ columns and $N-1$ rows. Each switch element (SWT) C_i (where $i = 1, 2, \dots, N-1$) acts as a multiplexer, selecting inputs from either the triangular array or the square array and switching them to the output ports. The appropriate number of clock delays are added to ensure that output cells in each cell cycle appear at the output at the same time. Therefore, the total delay of the BDN is equal to $(3N-1) \times t_c$, where t_c is the clock period.

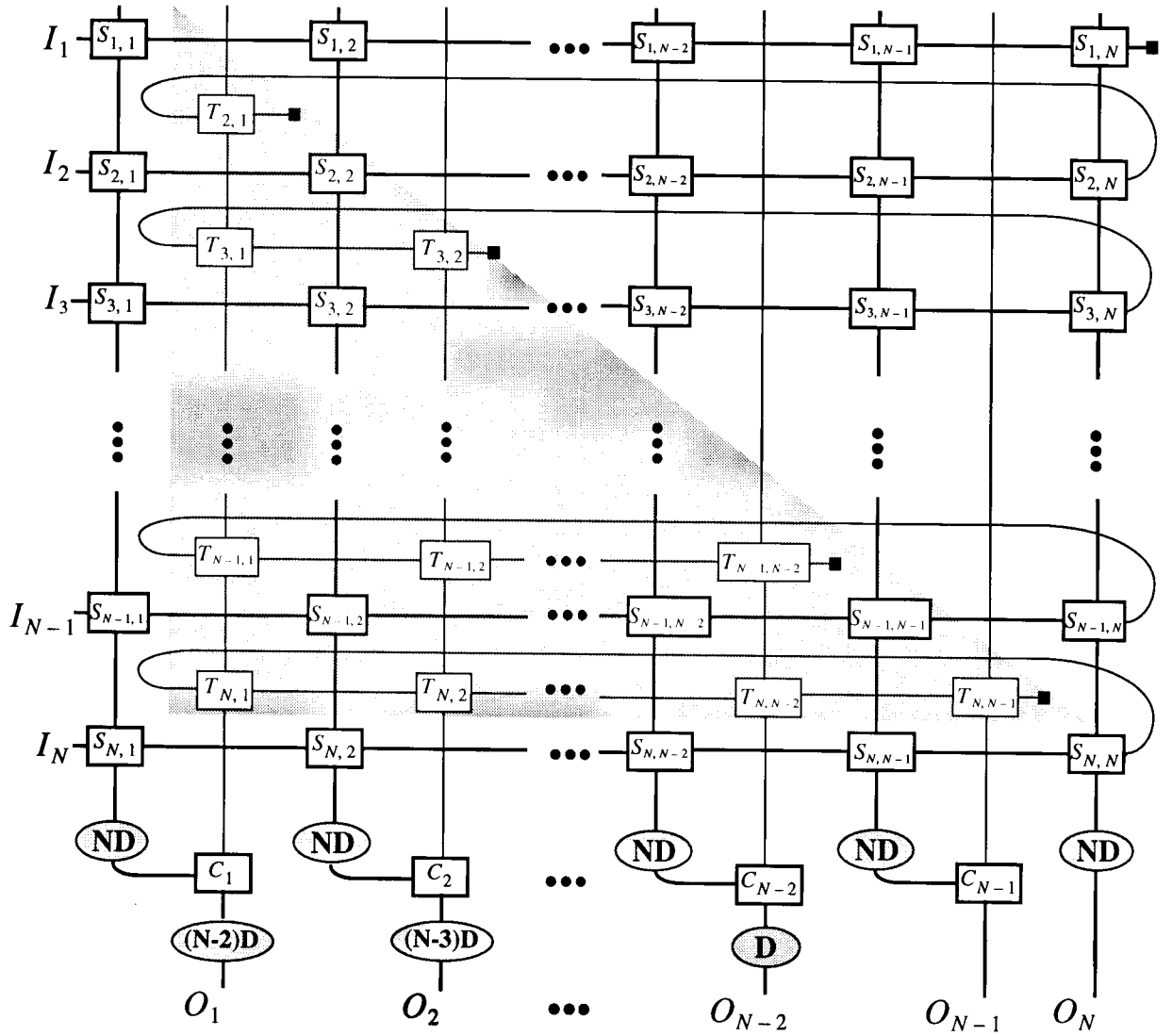


Fig. 3.8: Folded crossbar array of the BDN

The control of data flow through the folded crossbar is done in a distributed fashion using the VEC and the status bits of the SWTs (i.e. *BAR* signals). As shown in Fig. 3.9, in the VEC unit there are N , 1-bit register flags, such that register F_i is associated with column i of the primary array as well as column i of the secondary array. The connectivity between the SWTs and a regis-

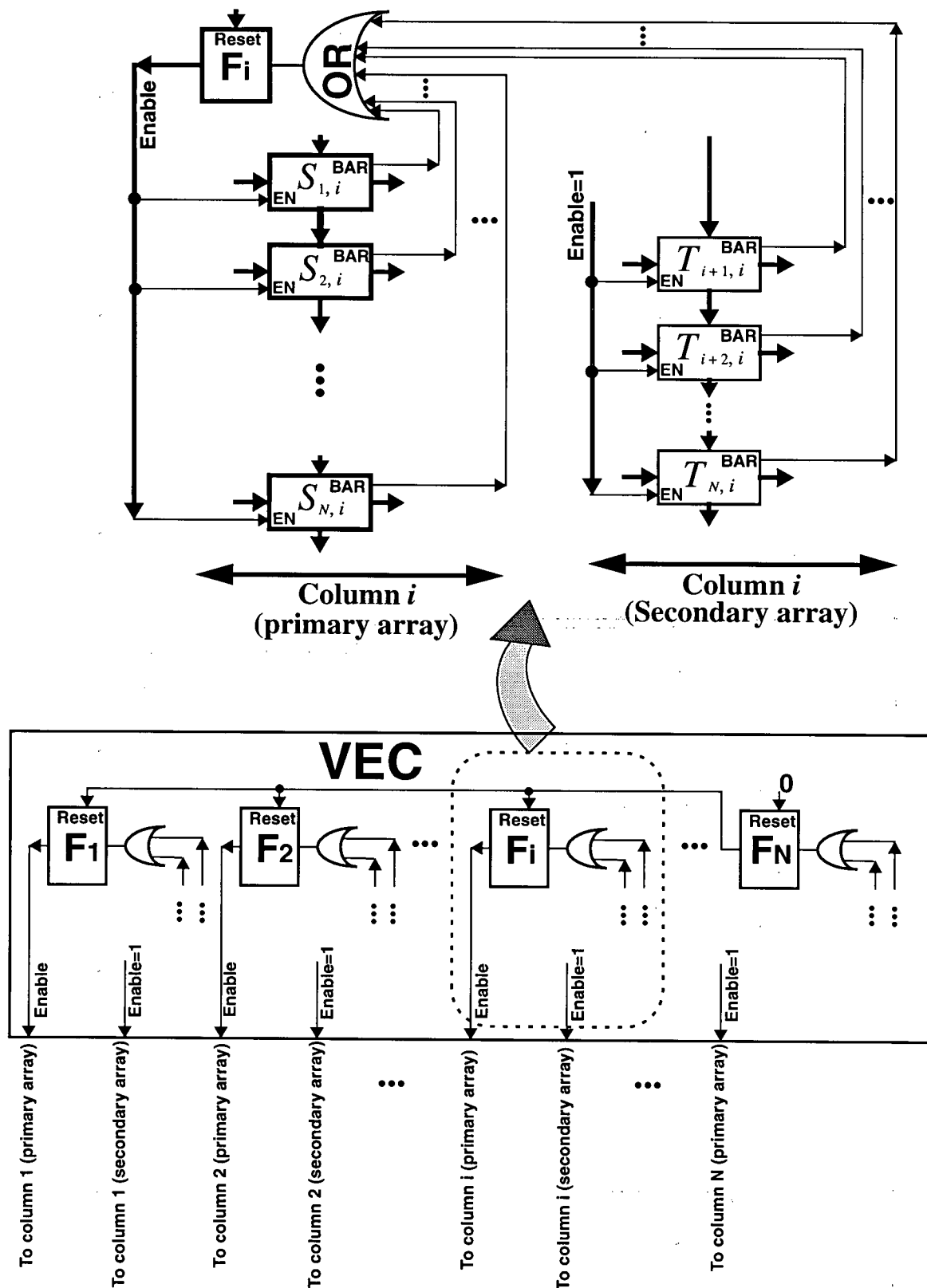


Fig. 3.9: Distributed control in the VEC

ter i is detailed in Fig. 3.9. Basically, a SWT in column i (of either array) which is set to a toggle mode in the current routing phase, will force F_i output to “High” in the next routing phase. Since F_i is connected to their Enable (EN) inputs, all SWTs in column i of the primary array will be forced into the cross mode in the routing phase when $F_i = 1$. Note that the F_i output does not affect SWTs in the secondary array as well as SWTs in the last column (column N) of the primary array, i.e. these SWTs are always enabled. This is because the secondary array SWTs must be always ready to receive packets from the primary array where all new routing phases start. Also, note that the broadcast delay is ignorable and the dominant delay will be the packet length.

Because data destined to a FIFO buffer may come from either the primary or the secondary array, a 2-to-1 multiplexer is used to select one of the two inputs. Such multiplexers are denoted by C_i s in Fig. 3.8. In any routing phase, only one of the two columns feeding a multiplexer is enabled. Note that each primary array column is connected to the multiplexer through a delay chain of length N (labeled ND in Fig. 3.8). This is to equalize the delays in the crossbar so that all packets will start leaving the crossbar at the same time. Since in practical designs, the values that N can take will not be very large, e.g. $N < 8$, the size of and delay of the folded crossbar will not be significant compared to packet length and buffering delays elsewhere in the design. Observe that each switching element or delay element introduces one clock delay where a clock delay is equivalent to transmitting a unit of data (bit, byte, ...) through a SWT. So the dominant delay will be the packet length, followed by the much longer delay in the FIFO buffer (multi-packet delay, etc). Actually, the delay of the BDN is completely transparent, because the packet head reaches the output FIFO while its tail is still at the input port because we use a pipelined (wormhole) routing technique. The packet timing in Fig. 3.4 illustrates these facts.

Alternative Design: The folded crossbar design described above performs both pack and shift functions. In situations where an efficient design of a shifter (e.g. a barrel shifter) already exists, the design of the folded crossbar can be greatly simplified since it now needs to perform only the packing function. Two possible configurations of the packing crossbar network are shown in Fig. 3.10; the crossbar of Fig. 3.10(b) has a shorter overall delay as well as the minimal number of SWTs.

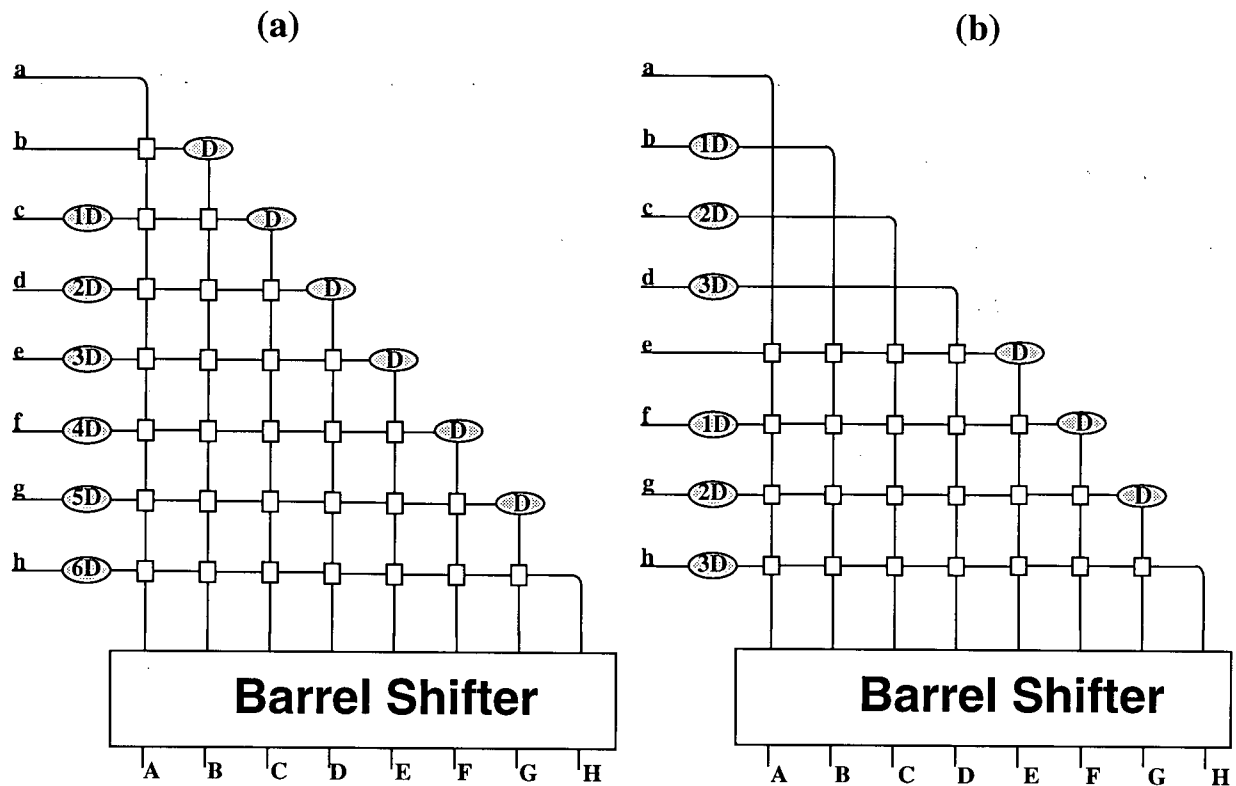


Fig. 3.10: Two configurations for packing the incoming cells (shifting by a barrel shifter)

3.5 Some Applications of the MIMO Buffers

It has been shown that pure output buffering is inadequate for providing acceptable cell loss under heavy non-uniform cell traffic for fast packet and ATM cell switching [20]. A network switch element can employ cell queueing in order to prevent cell loss when congestion occurs in the switch element. Adding small amounts of internal buffering in the switch element can result in significant performance gains [52], [53]. The problem therefore is to provide a buffer for use in the internal stages of data packet switch which requires a minimal increase in hardware and reduces losing packet within the switch. The MIMO buffer structure which is proposed in this thesis fulfills this requirement for high-speed networking applications [8].

For example, the Knockout switch [10] is one of the most extensively analyzed switch designs and is often used as a standard against which to compare other designs. This is an output buffered switch in which the buffers are FIFO memories. The switch simply connects all the inputs to each output through a concentration stage. Each output port is a module consisting of a concentrator and a shared buffer. The shared buffer stores the cells until they can be transmitted out the output port and the concentrator determines which cells to pass through the shared buffer. However, this type of the pure output buffering is inadequate for general input cell traffic and internal buffering should be located. Even small amounts of internal buffering can result in significant performance gains [52]. The MIMO buffer is a hardware solution for this problem.

Furthermore, a VLSI implementation of a simple hierarchical modular ATM switch was proposed in [27] using the MIMO buffer. The design is based on a generalization of the concept of the growable structure developed in [42], and employs shared buffers in the front-end of the architecture. The proposed switch architecture is based on a buffered fat-tree (BFT) structure [42] which

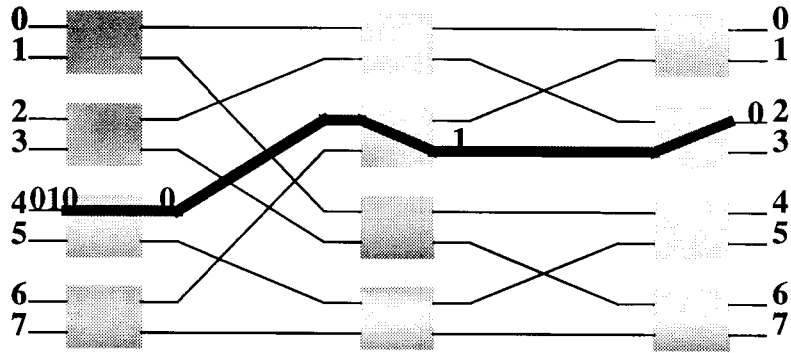


Fig. 3.11: Self routing property of banyan networks

is characterized by full buffer utilization as well as high bandwidth utilization, both achieved with optimal delay-throughput performance. The BFT switch has a tree topology with hierarchical sharing of buffer space and bandwidth resources. Each internal node of the BFT is constructed using shared buffers, each dedicated to a group of output nodes. The main advantage of the BFT switch over other known output-buffered switch architectures is its robustness to unbalanced (or non uniform) traffic [42]. The proposed ATM switch architecture has a uniform and regular structure and, has the advantages of: 1) relaxed synchronization for data and clock signals; 2) high integration density for VLSI implementation; and 3) easy expansion due to the modular structure of the buffer.

Moreover, the self-routing property of banyan networks, as shown in Fig. 3.11, provides great simplicity in the control of the switching elements and hence makes them attractive for implementing high speed switching nodes [121]. In self-routing, an destination address field (A) is appended to each cell at the input port before the cell enters to the switch fabric. Each bit of the destination address field is examined by each stage of the switch element. If the bit is 1, it is routed to its lower output. As shown in Fig. 3.11, a cell whose destination address is 2 (010) is

routed at the input port 4. The first bit of the destination address (0) is examined by the first stage of the switch element. The cell is routed to the upper output and goes to the second stage. The next bit (1) is examined by the second stage and the cell is routed to the lower output of the switch element. At the last stage of the switch element, the last bit (0) is examined and the cell is routed to its upper output corresponding to the output port 2. Once the cell arrives at the output port, the destination address is removed.

However, because of the internal blocking property of banyan networks, as illustrated in Fig. 3.12, they cannot be directly used for switching purposes. Therefore various topologies based on the banyan network with internal buffering have been studied in the literature [43]-[47]. Besides buffering, internal bandwidth of memoryless interconnection is the other important resource of a switching fabric. The main bottleneck in a banyan network is the limited bandwidth of the inner links which causes the blocking property. Dilated networks have been proposed in various forms to increase the internal bandwidth of a switch [49]-[51]. By employing sufficient dilation, the QoS coupling of different flows within the switch fabric is limited to cell loss which can be made arbitrarily small. Alternatively, researchers have considered increasing the number of ports of a switching element to increase bandwidth. Dilated networks are attractive as they provide sufficient bandwidth [49]. However, conventional dilated networks do not efficiently utilize the internal bandwidth, while networks with large number of ports per switching element, do not provide sufficient bandwidth. Therefore, a dilated banyan network model using our novel MIMO buffer module can be considered to address the problem. The buffer has the ability to increase the throughput of a network by enabling different switches on a path to operate on different packets at the same time. This is a form of pipelining that increases the number of packets that can be passed through the network in a given period of time.

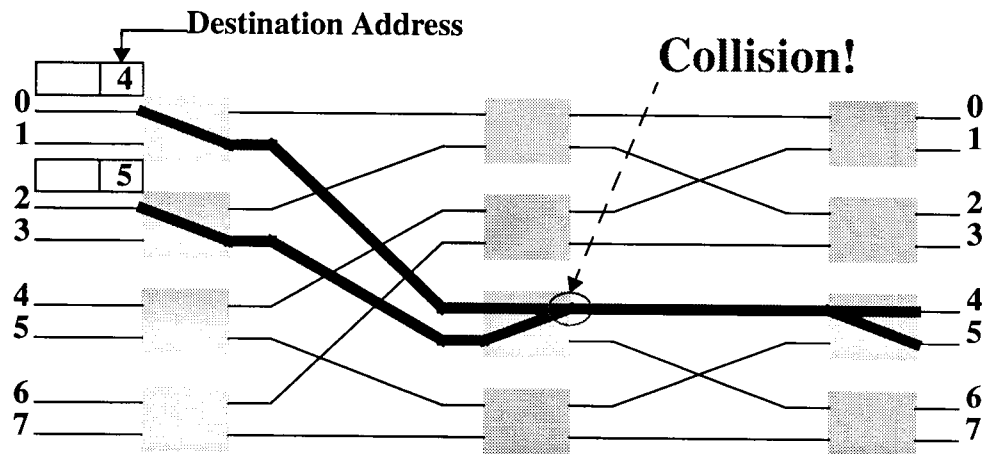


Fig. 3.12: Internal blocking in banyan networks

Dilated Banyan Architecture with Internal Buffers: One of the main objective of a switch design is to develop a highly optimized architecture with respect to bandwidth and buffer utilization under arbitrary traffic patterns. At the same time the switch should be amenable to VLSI implementation and scalable. The dilated banyan switch originally proposed in [48] is based on fully utilizing the internal bandwidth of the banyan network. The architecture has a feasible realization using single type of switch and memory modules to construct fabrics of arbitrary dilation. The uniformity and modularity of the architecture makes it an ideal candidate for practical synthesizable design.

This section provides a modified dilated banyan model which incorporates parallel MIMO buffers in the internal stages to provide significant performance enhancement under bursty traffic. VHDL (VHSIC Hardware Description Language [55]) development tools have been employed for the design and modeling of the switch. Our VHDL model is parametrized in the number of input/out-

put ports of the switch as well as arbitrary levels of dilation and the number of stages, so that switches of different sizes can be synthesized from the same VHDL code. The VHDL model has been also used for timing and functional verification. The switch architecture is outlined in the following. An investigation of such factors as the impact of different traffic patterns, use of shared output queueing and various levels of dilation are discussed elsewhere [54].

The dilated banyan architecture is a packet switch with switching elements having multiple input and output links per port. Basically, the architecture can be constructed from two building modules: expansion switch-element (ESE) and buffered-concentration switch-element (BSE).

A primitive ESE module is defined with two inputs and four outputs, and is denoted by ESE(1:2). As shown in Fig. 3.13, module ESE(1:2) is constructed from two basic 1x2 switch-elements (SE's). Expansion modules of larger sizes can be built from a single stage of primitive modules, as illustrated in Fig. 3.13. Note that in general module ESE(n:2n) is reared by n primitive ESE(1:2) modules in a single stage.

A basic buffered module (BSE) can be defined as a switching module with four inputs and two outputs, and is denoted by BSE(2:1). In its simplest form, a buffered module BSE(2:1) is constructed from one ESE(2:4) module (or two ESE(1:2) modules), and two shared buffers each with four inputs and one output, as shown in Fig. 3.14. BSEs of larger sizes can be built from a single stage of primitive ESE(1:2) modules and buffers. Generally, a BSE(2n:n) module is built using n primitive BSE(2:1) modules (or 2n ESE(1:2) modules), and 2n shared buffers. In other words, a parallel arrangement of ESEs and shared buffers can be used to construct BSEs of varying size. This enables the modular growth of the proposed architecture.

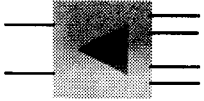
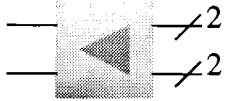
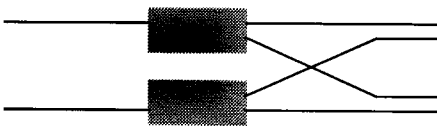
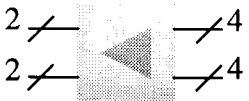
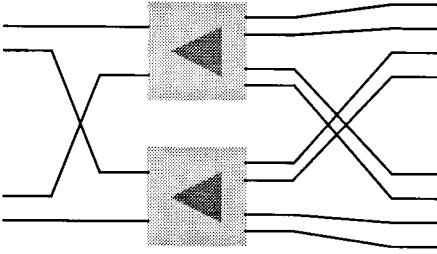
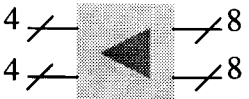
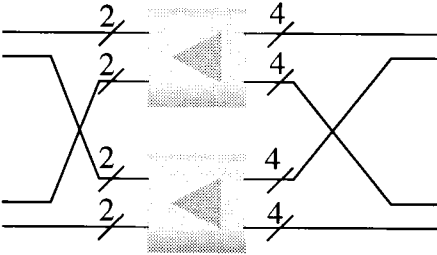
<u>Name</u>	<u>Symbol</u>	<u>Structure</u>
ESE(1: 2)	 <p>or</p> 	
ESE(2: 4)		
ESE(4: 8)		

Fig. 3.13: Expansion modules (ESEs)

Fig. 3.15 shows our proposal for constructing BSEs of arbitrary input and output dilation using ESEs of fixed dilation pattern. For example, if module BSE(4:1) is defined as the basic (or core) module, as shown in Fig. 3.15, it can be constructed from one ESE(4:8) (or four ESE(1:2)), and two shared buffers (each with eight inputs and one output). Alternatively, BSE(4:2) can be defined as the core module. As illustrated in Fig. 3.15, BSE(4:2) can be also built from one ESE(4:8), and two shared buffers (each with eight inputs and two outputs). This primitive is preferable if two stages of buffered-concentration modules are needed [54]. Appropriate core modules can be engi-

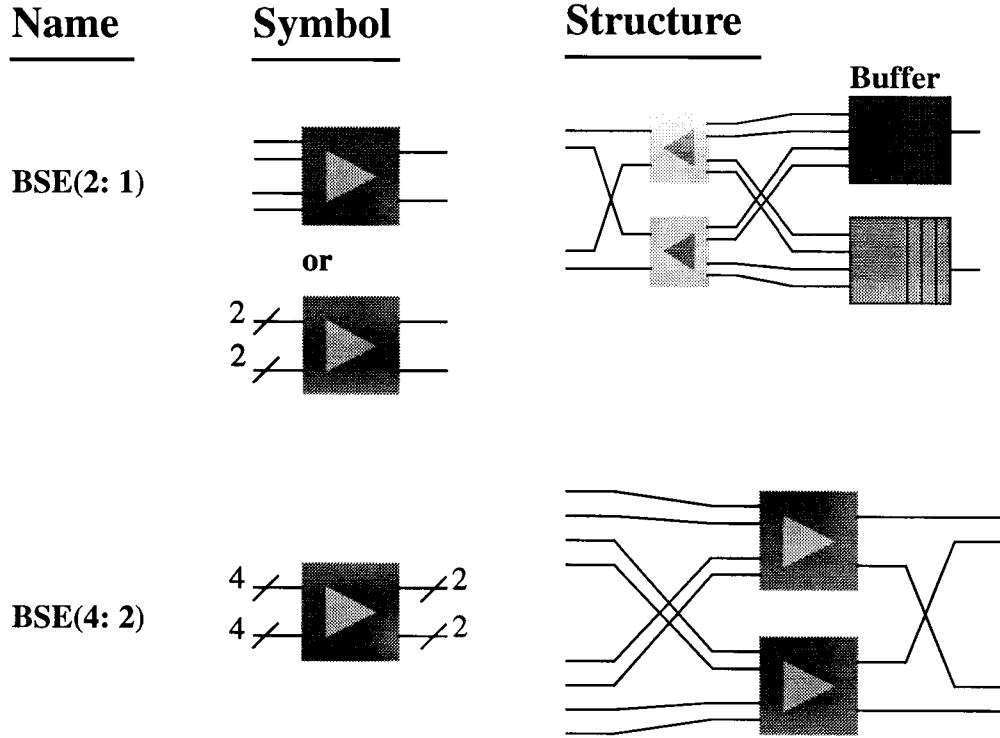


Fig. 3.14: Buffered-concentration modules (BSEs)

neered through performance simulations [54]. Therefore, shared buffers with various number of inputs and outputs are needed in this network. The MIMO buffer is an ideal candidate for this application.

Using the described building block modules, a dilated banyan networks with internal buffering can be designed. In particular, in the proposed architecture the number of input and output links will grow in the first few stages of the switch. However, in the remaining stages, the number of links can remain fixed or even decrease. This is based on the observation that in multibuffered banyan networks, performance improvement has been achieved for buffer sizes of up to four cells [52], [53]. With buffer sizes greater than that, the improvement in performance is negligible.

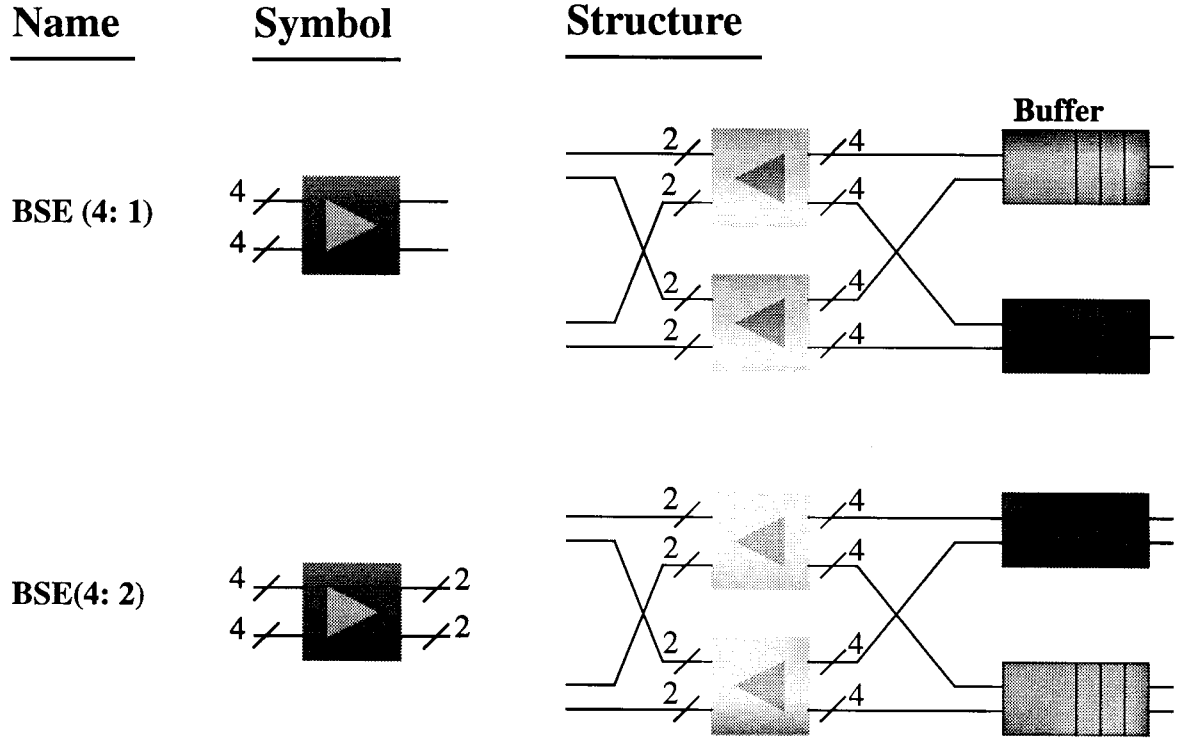


Fig. 3.15: Alternative buffered core modules

As an example, an 8x8 switch using the proposed architecture is shown in Fig. 3.16. The switch is constructed using four ESE(1:2), four ESE(2:4), and four BSE(4:2) modules. The result can be translated to 28 primitive ESE(1:2) modules, and 8 shared buffers, each with eight inputs and two outputs, or alternatively, 16 shared buffers, each with four inputs and one output. Note that other various levels of dilations are also possible. Therefore, a MIMO buffer is the most important block of the proposed architecture.

3.6 Summary

This chapter developed the design concept and hardware architecture of a parallel FIFO buffer structure for high-speed packet networks. Our approach employs a systolic routing network and

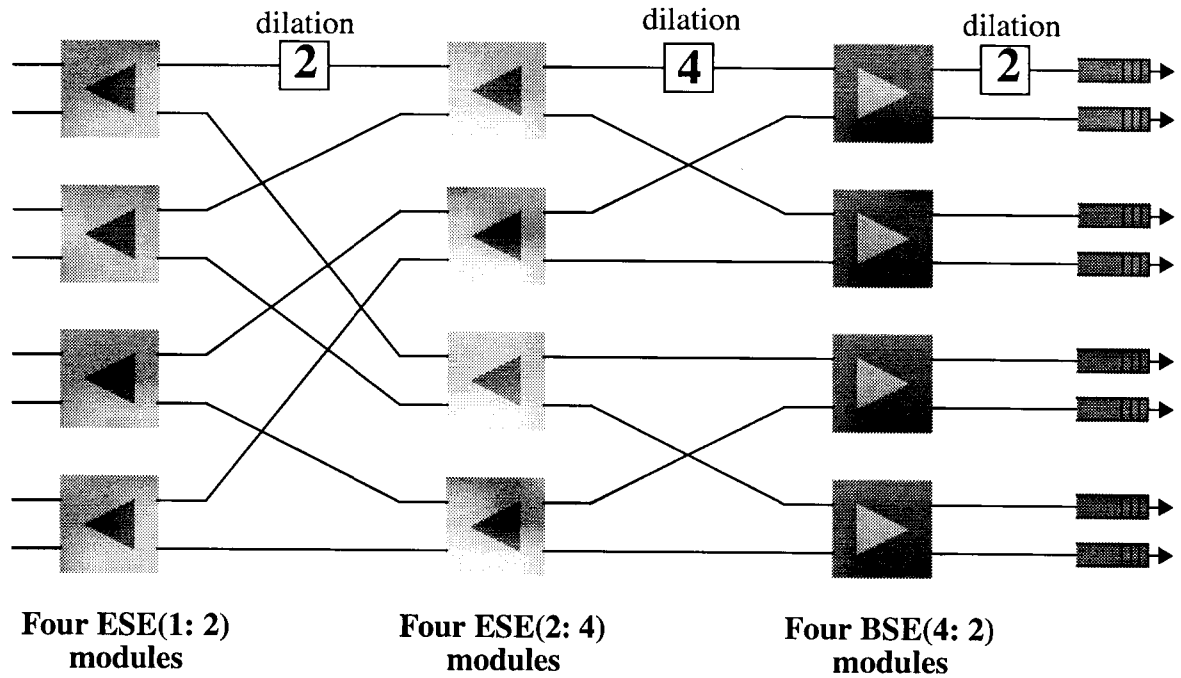


Fig. 3.16: Dilated banyan network with internal buffers (an 8x8 network example)

bank of parallel FIFO buffers to yield a load-balanced multi-input multi-output FIFO realization with increased bandwidth. The MIMO buffer adaptively manages the available buffer space for statistically multiplexed input traffic. Our systolic routing network provides significant advantages over previously proposed banyan/butterfly networks since the systolic network eliminates the need for a running-adder circuit for ranking cells. Load-balancing is done using arithmetic-free circuits in our design approach. Using this approach we derive scalable parallel FIFO buffer structures that can be designed to match the rate of ultra high-speed links using current memory technology that uses moderate clock rates. Our design approach is truly scalable and has real advantages from VLSI perspective. Some applications of the MIMO buffer were also discussed in this chapter. The next chapter presents VLSI implementation of the buffer.

CHAPTER 4

VLSI Implementation of the Parallel MIMO Buffer

4.1 Overview

This chapter develops VLSI implementation of the parallel multi-input multi-output (MIMO) buffer structure (presented in Chapter 3) which can be used in ultra high-speed network interfaces and similar applications. The combined use of pipelined architecture and dynamic CMOS circuits resulted in significant reduction in design complexity and substantial performance gains in speed and chip area. We demonstrate our design approach by showing a balanced MIMO buffer design in 0.5- μ m CMOS technology operating at about 330 MHz [8]. At this clock rate, and using an 8-

bit wide data-path, we show the design of a 4-input 4-output buffer with a throughput of about 10.6 Gb/s. Such a buffer is capable of interfacing to a Sonet OC-192 link with 10 Gb/s bandwidth. In the next section, we elaborate on VLSI design strategy on the implementation of the buffer. Circuit design issues are presented in Section 4.3. Then, simulation results are discussed in Section 4.4.

4.2 VLSI Design Strategy

Two different design strategies have been used to implement our MIMO buffer. The first strategy involved developing a synthesizable VHDL model of the buffer. Our VHDL model is parametrized in the number of ports of MIMO buffer as well as the data-path width so that buffers of different sizes can be synthesized from the same VHDL code. A portion of the VHDL code that describes an SWT is shown in Fig. 4.1. The VHDL model has been also used for timing and functional verification. However, because our main target was to operate the MIMO buffer at the maximum possible clock rate, our second design strategy was focused on transistor-level circuit development.

The first step in our design flow involved full-custom circuit design of the basic buffer using 0.5- μ m CMOS technology, followed by simulating the topmost hierarchical schematic using a circuit simulator such as Hspice and Spectre provided by our CAD tools from *Cadence*TM¹. After simulation, a standard cell layout is developed, Design Rule Check (DRC) is performed, and Logic Versus Schematic (LVS) is applied to compare the netlist extracted from the layout and the netlist from the schematic. This step ensures that all nets and blocks defined in the schematic are presented and connected in the layout. After performing place and route (P&R), DRC is applied

1. Cadence Design Systems Inc., San Jose, CA, USA, <http://www.cadence.com>.

```

entity XBAR_CELL is
  port ( clk : in STD_LOGIC;
        en : in STD_LOGIC;
        i_in: in STD_LOGIC;
        d_n, d_w : in STD_LOGIC_VECTOR (XBAR_IN_WIDTH-1 downto 0);
        p_in : in STD_LOGIC;
        bar : out STD_LOGIC;
        i_out: out STD_LOGIC;
        d_s, d_e : out STD_LOGIC_VECTOR (XBAR_IN_WIDTH-1 downto 0);
        p_out : out STD_LOGIC);
end XBAR_CELL;

architecture RTL of XBAR_CELL is
  signal checked, switch : STD_LOGIC;
  signal latch_switch : STD_LOGIC;
  begin
    logic : process (i_in, p_in, checked, d_n, d_w)
    begin
      if (i_in = '1') then
        checked <= '0';
        switch <= '0';
      else
        if ((p_in = '1') and (checked = '0')) then
          if ((d_n(0) = '0') and (d_w(0) = '1')) then
            switch <= '1';
          else
            switch <= '0';
          end if;
          checked <= '1';
        end if;
      end if;
    end process logic;
  ...

```

Fig. 4.1: Part of the VHDL code

again to check for design rule errors that may have occurred during P&R, then LVS is used to check for discrepancies between schematic and layout. Final simulation is performed to check the effect of parasitic resistors and capacitors introduced in the layout. Circuit simulation results assure that the BDN unit functions correctly up to about 330 MHz in 0.5- μ m CMOS technology.

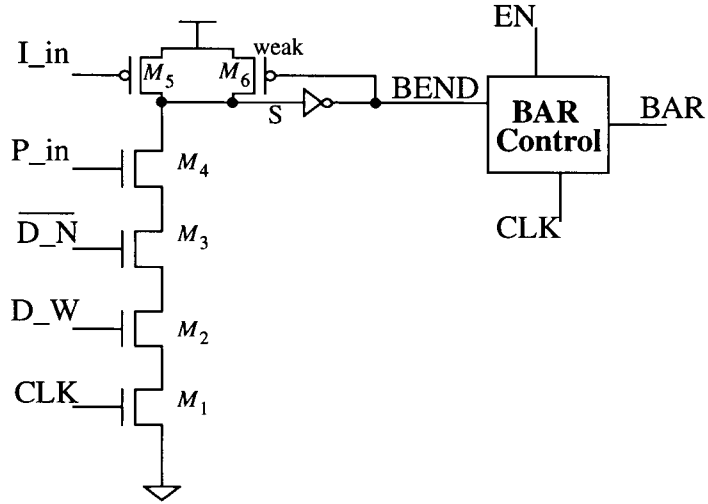


Fig. 4.2: Circuit diagram of the crossbar control of SWT

4.3 Circuit Diagram of a Switch Element

The circuit diagram of the crossbar control part of a SWT is shown in Fig. 4.2. The modified domino CMOS circuit technique uses two different clock signals, a (system) *clock* and a *cell clock* (i.e. CLK , and I_{in}). The use of such dynamic CMOS circuits achieves significant reduction in the number of transistors and, more importantly, a higher switching speed compared to static circuits. The circuit operation is based on first precharging the output node capacitance and subsequently, evaluating the output level according to the applied inputs [38].

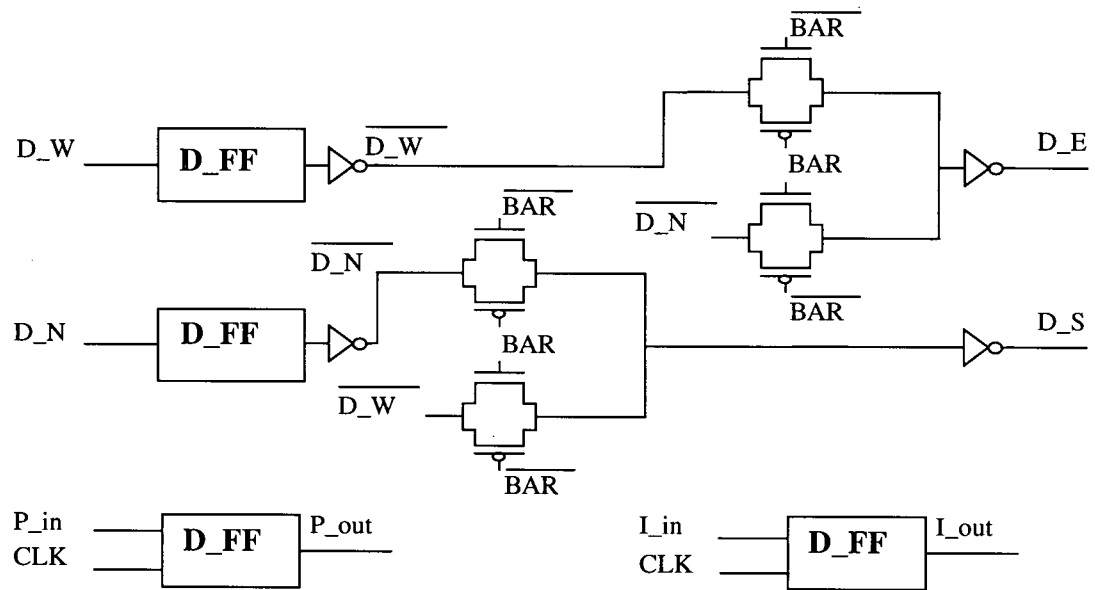
In Fig. 4.2, the 1-bit active low *cell-clock* signal, I_{in} , is applied to precharge the internal node S in the SWT in the beginning of every cell cycle. The “*BAR Control*” block has a simple logic which generates an active high signal for the entire cell period if both the enable signal (EN) and the internal signal $BEND$ are active. As mentioned before, the BAR signal is passed as an output of

the SWT to indicate the mode of the switch element in each time-slot.

At the beginning of each cell cycle, I_{in} goes low to precharge the common Drain node of the M_5 and M_6 transistors (i.e. node S), forcing the SWT to enter or stay in a cross mode (i.e. the internal node $BEND$ is high). Once node S is precharged to the threshold of the inverter, the output of the inverter will go low, which turns on the weak p-transistor M_6 [39]. This will keep node S high as long as there is no path that will pull the node voltage to ground through the cascaded group of transistors lead by M_4 . When P_{in} is asserted, activity bits of the west and north data inputs (D_W and D_N) are compared by M_1 , M_2 , M_3 , and M_4 transistors. If D_W is high and D_N is low, i.e. a valid cell from the west and an empty cell from the north, these four transistors discharge node S . The output of the inverter, i.e. node $BEND$, will then go high once the threshold of the inverter is reached.

The complete signal and data-path of a SWT is shown in Fig. 4.3. Data inputs D_W , and D_N , and control signals I_{in} and P_{in} are first latched by four D_FF's. The BAR signal specifies the direction of data-path in each cell period. The control signals P_{in} and I_{in} are passed as two output signals P_{out} and I_{out} . The data words D_W and D_N are also passed to neighboring SWTs in the same fashion. Recall that the data-path through a SWT can be w bits wide, as indicated in Fig. 3.2, and Fig. 3.3.

Fig. 4.4 shows a fast dynamic D flip-flop circuit that we employed in the implementation. The area of the flip-flop is $592.8 (\mu m)^2$ based on a $0.5\text{-}\mu m$ CMOS technology, and circuit simulations assured that it can operate at clock speeds exceeding 330 MHz.



D_FF: D-type flip-flop

Fig. 4.3: Signal and data-path of a SWT

Circuit Design Issues: In Fig. 4.3, note that the low on-resistance of the CMOS transmission gate usually results in a smaller transfer time compared to those for nMOS-only switches. Normally, nMOS transistors produce “strong zeros” and pMOS devices generate “strong ones”. The single-device pass gate has the disadvantage that the resistance of the switch increases dramatically when the output voltage reaches the threshold voltage $V_{in} - V_{th}$, as the transistor goes into linear operation mode. Adding a pMOS transistor in parallel with the nMOS solves these problems. In other words, an nMOS transistor provides a poor “1” and a pMOS a poor “0” level; therefore, the combination of the two results in a better switch. Also, there is no threshold voltage drop across the CMOS transmission gate, i.e. threshold loss. When designing static-pass transistor networks,

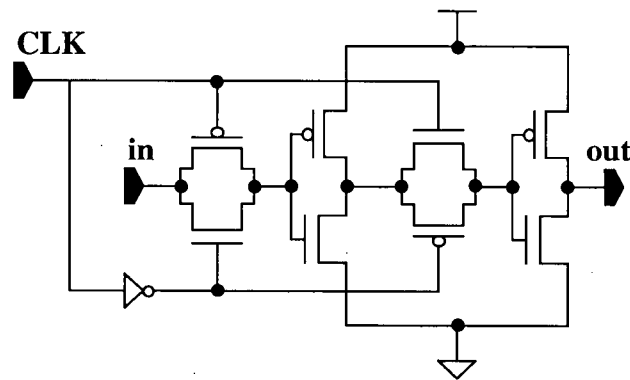


Fig. 4.4: Dynamic D-type flip-flop circuit

it is essential to adhere to the low-impedance rule under all circumstances. Also, devices close to minimum-size should be used, unless an external load capacitance is the dominating factor, which is not the case here because of the systolic architecture.

Also note that the data and control signal path in a SWT involves only one stage of CMOS transmission gates which can be sized properly to achieve maximum speed. Therefore, the main speed limitation of the SWT is due to its controller circuit shown in Fig. 4.2. In the SWT controller circuit, the series connection of nMOS transistors must be designed carefully to avoid unnecessary delays. The series connected nMOS transistors, which appear in a number of design styles basically for their simplicity and possibly their improved performance, requires careful consideration and transistor sizing to improve the speed performance of our circuit [40]. The transient performance of the circuit can be improved by adjusting the nMOS transistor sizes in the pull-down path, with the objective to reduce the discharge time.

Although, the calculation of the channel resistance and the parasitic capacitance of an nMOS in

the series chain is complicated [40], studying the simple RC low-pass filter model, in which each nMOS is replaced by one link of RC chain consisting of the channel resistance and the parasitic capacitance, provides a good measure of delay associated with a discharge through a chain of transistors. Using Fig. 4.5 as a typical transistor chain structure from Elmore's RC model [38], the delay t_d of the chain of transistors can be represented by the expression:

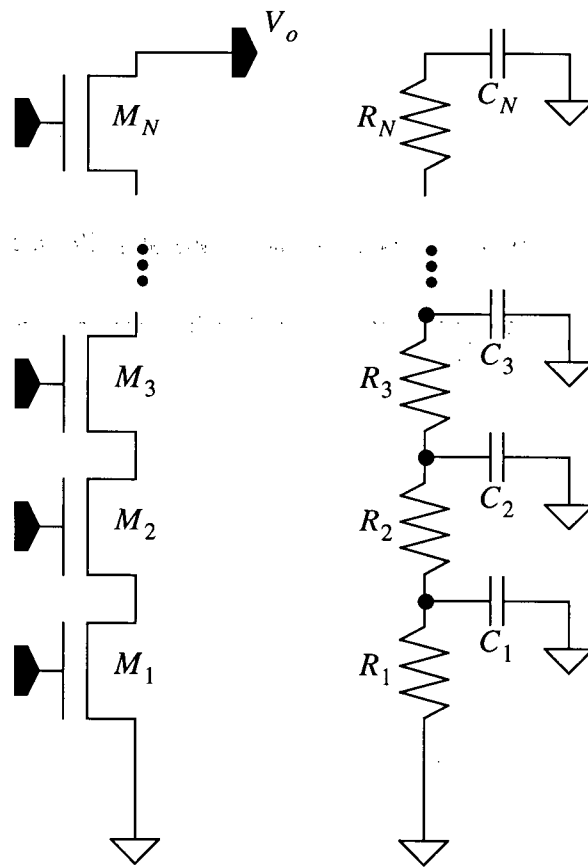


Fig. 4.5: nMOS transistors chain and the RC model

$$t_d \equiv \sum_{i=1}^N \tau_i \text{ where } \tau_i = \left(\sum_{j=1}^i R_j \right) \cdot C_i$$

τ_i basically describes the RC time constant for discharging the i th capacitor through all the resistors in the discharge path to a 63% level. Deriving the propagation delay now becomes identical to the analysis of the resulting RC network. Although this would be only an approximation of t_d [40], nevertheless we can get a deeper insight into the problem by studying the simple RC model. In the case for the chain network of Fig. 4.5 where there are 4 stages, i.e. $N = 4$, the relation that describes the discharge path can be analyzed as follows. Delay associated with each stage are τ_1 , τ_2 , τ_3 , and τ_4 , from which a total delay time can be driven:

$$t_d \equiv \tau_1 + \tau_2 + \tau_3 + \tau_4 = R_1 C_1 + (R_1 + R_2) C_2 + (R_1 + R_2 + R_3) C_3 + (R_1 + R_2 + R_3 + R_4) C_4$$

The delay model highlights that C_4 contributes the most to the delay. The effect of scaling the device sizes in each of the stages is seen by rewriting the delay model in the form:

$$t_d \equiv R_1 (C_1 + C_2 + C_3 + C_4) + R_2 (C_2 + C_3 + C_4) + R_3 (C_3 + C_4) + R_4 C_4$$

The relation shows that the delay depends on the geometrical dimension of the transistor. Consider first the nMOS transistor closest to the output node, i.e. M_4 . If the $\left(\frac{W}{L}\right)$ ratio of this transistor is reduced by a certain factor, two effects are seen: one tends to increase delay, the other tends to decrease delay. First, the current driving capability will decrease, i.e. the equivalent resistance of the nMOS transistor will increase. Second, the parasitic drain capacitance associated with this transistor will decrease. If the length of the nMOS chain is sufficiently long, the increase in resis-

tance has little influence upon the combined RC delay time, whereas a reduction of the capacitance significantly decreases the delay. In other words, when the $\left(\frac{W}{L}\right)_4$ is decreased by a fraction of Δk , in equivalent circuit, C_4 is replaced by $\frac{C_4}{1 + \Delta k}$, and R_4 by $R_4(1 + \Delta k)$; then t_d decreases.

This implies that we are able to reduce the time delay by scaling the device sizes in the entire chain. Therefore, the overall delay time can be reduced by decreasing the size of the nMOS transistor closest to the output node. Progressive sizing of transistors in a transistor chain copes with the extra load of internal capacitances, and scaling of the transistors while keeping the total transistor area constant gives better results than simply reducing the transistor size. The best performance results when we successively reduce the nMOS transistor's aspect ratios starting from the top (output) transistor working down towards the ground. As a rule, reduction of about 25% at each stage usually is a good compromise to speed up a typical circuit based on transistor chain designs. Table 4.1 lists transistors sizes for the nMOS transistors of Fig. 4.2 when implemented in 0.5- μm CMOS technology operating under a 3.3 V power supply.

Table 4.1: Transistors' sizes in the nMOS chain

Transistor	M_1	M_2	M_3	M_4
$\left(\frac{W}{L}\right)$ size (μm)	$\frac{10.8}{0.6}$	$\frac{8}{0.6}$	$\frac{6}{0.6}$	$\frac{4.8}{0.6}$

4.4 Design Simulation Results

We have developed a detailed transistor-level design of a BDN unit with four input ports and four output ports, where each port is 8-bit wide. We employed a 0.5- μm triple-metal/single-poly CMOS technology¹ to obtain the layout shown in Fig. 4.6. The area of the layout is $975 \times 1125 (\mu\text{m})^2$.

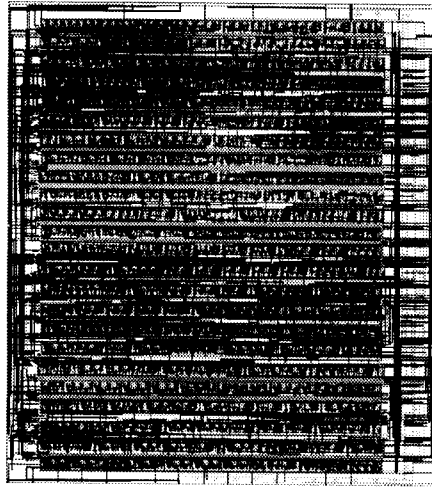


Fig. 4.6: Layout diagram of a 4x4 BDN

Full custom design of the BDN based on extensive use of dynamic circuit techniques resulted in a clock period of 3 ns (about 330 MHz) with acceptable margins. Fig. 4.7 shows four input signals (a, b, c, and d) and four output signals (A, B, C, and D) of the BDN when the clock is running at about 333 MHz. Note that only one bit (out of 8 bits) of each input/output signals along with CLK , I_{in} , and P_{in} signals are shown in Fig. 4.7. At this clock speed, a 4×4 BDN with an 8-bit wide data-path, delivers cells (or packets) at a maximum rate of 10.6 Gb/s. Achieving this performance required close integration of logic and microarchitecture optimization, employing high-perfor-

1. Provided by Canadian Microelectronics Corporation (CMC), <http://www.cmc.ca>.

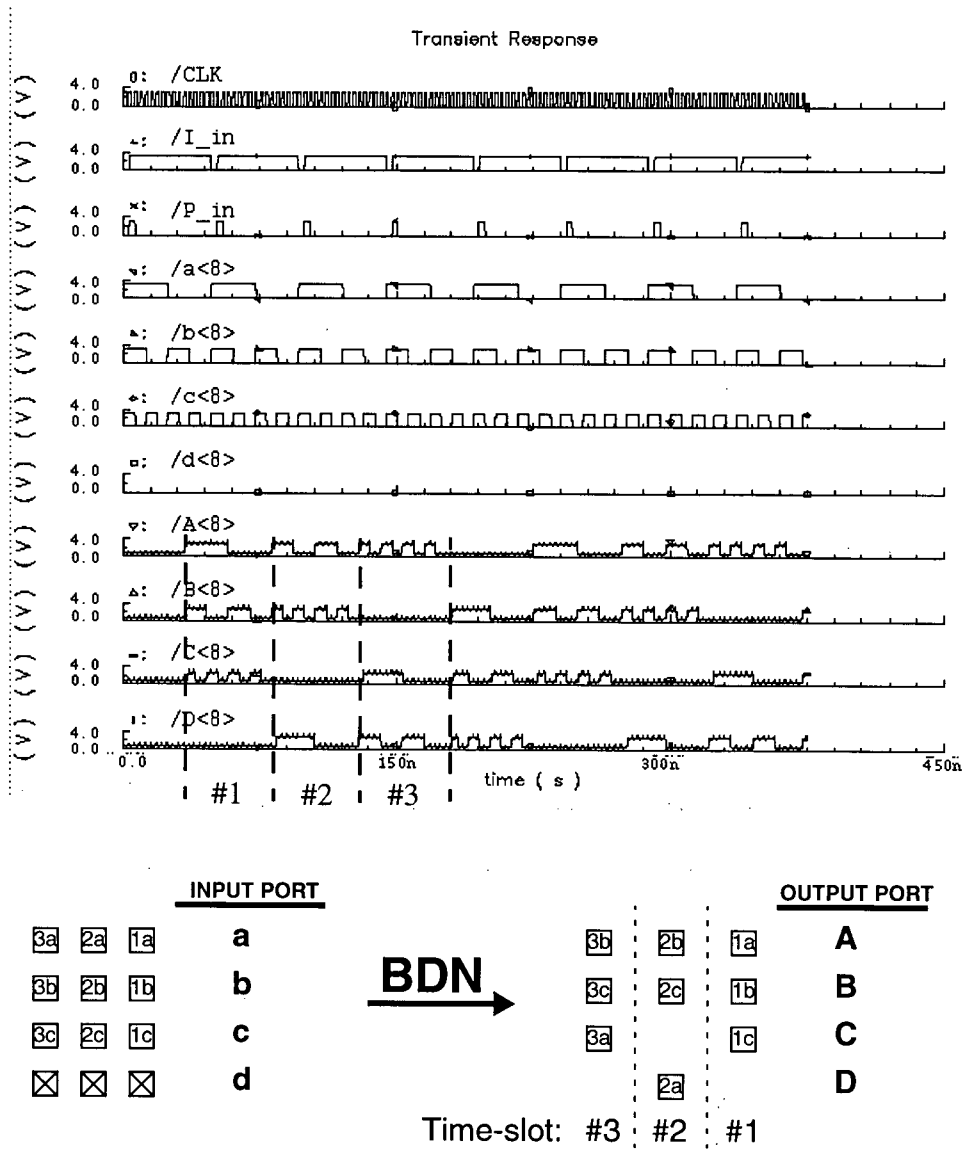


Fig. 4.7: Simulation result of the 4 x 4 BDN during three time-slots (clock cycle is 3 ns)

mance CMOS circuit techniques, and the use of advanced custom design tool suites.

Fig. 4.7 shows the timing simulation of the BDN during three time-slots. For simplicity, the validity bits of the input signals a, b, and c have been asserted in all three phases, and empty cells are always sent on input port d. Therefore, the first time-slot illustrates the situation when the first three columns of the primary array of the BDN must be used to route the incoming cells. In time-slot #2, in addition to the last column of the primary array, the first two columns of the secondary array are also employed, while in the third time-slot, columns 2, 3, and 4 of the primary array are utilized. As shown, the outcomes of these three test sets match the expected results.

4.5 Summary

This chapter presented hardware realization of the parallel MIMO buffer structure. The combined use of pipelined architecture and dynamic CMOS circuits resulted in significant reduction in design complexity and substantial performance gains in speed and chip area. A small 4-input 4-output prototype buffer has been designed using a triple-metal/single-poly 0.5- μ m CMOS technology to demonstrate our concepts. The buffer can attain a rate of 10.6 Gb/s which is more than adequate to support a Sonet OC-192 link.

A major problem in the design of VLSI crossbar networks is the simultaneous activation of a large number of line-drivers at the output leads of a silicon structure. The density of interconnection lines, their associated drivers, and connectors, pose major performance limitation in VLSI implementations. Fortunately, multilevel metallization (e.g. triple-metal in our 0.5- μ m process) has provided the needed technology to achieve the low-resistance, crossing interconnections. Also, distributed implementation variations can be considered to avoid select lines altogether. For example, the control planes (VEC, and HOC) can be used solely as a central contention arbitrator. On receiving the acknowledgment signals, active cells from input ports are self-routed independently to their destinations. Another way is a distributed control scheme that avoids a centralized control plane. This variation gives a modular structure, and has been chosen in our design. Therefore, the overall switch capacity can increase by simply adding more arrays leading to a truly scalable realization. Also, speed optimization of the design required the use of progressive scaling of the transistors in the series chain such that the nMOS transistor closest to the output node has the smallest width-to-length ratio.

CHAPTER 5

A Systolic Parallel Priority Queue for High Speed Packet Networks

5.1 Overview

Priority-based scheduling is one of the most widely used techniques in high-speed packet networks. Normally, a packet arriving to a buffer, in a router or a packet switch, is assigned a priority number which is appended to the packet as an additional field. Packets in the buffer are then ordered and delivered according to their priority value by the scheduler. A major challenge is maintaining the correct operation of the priority queue at very high link speeds, e.g. at 2.4 or even 40 Gb/s. This chapter generalizes the priority queue concept to a *parallel priority queue* (PPQ) which can be scaled to meet the requirements of ultra high-speed links using standard CMOS technology.

5.2 Scheduling Networks and Priority Queues

Providing quality-of-service (QoS) guarantees for multimedia transport applications over packet-switched networks, such as the Internet, relies crucially on the scheduling and buffer management capabilities of the network switches and routers. Scheduling is required to order packets, for example, to satisfy delay bounds of real-time sessions or to enforce fair bandwidth allocation for sessions sharing a link. In priority-based scheduling, a packet is assigned an attribute which becomes its priority number. For example, the priority number may be a finish-time computed by a weighted fair queuing (WFQ) algorithm [4], [5], or a class index assigned by a class-based queuing (CBQ) algorithm [6], [7], etc.

In any case, accessing the highest-priority packet requires the use of a priority queue structure. By definition a *priority queue* is a queuing structure for which the dequeue operation always removes the highest (or lowest) priority element in the queue. In this chapter we will associate higher priority with smaller positive values. This is because in a packet scheduler, highest priority is given to packets with the smallest departure time-stamp.

Fig. 5.1 shows a typical system containing a priority queue (PQ), its associated packet-memory (PM), and a scheduler. We assume that the priority numbers (P) that enter the PQ have fixed-point representation and are of equal length. Each entry in the PQ contains the pair (P,A) , where P is as defined above and A is an address pointer to a packet in the packet memory. Basically, when a new packet arrives at the packet memory its priority number P is stripped off and pushed into the PQ together with the packet address (which is the pointer A). The pair (P,A) will be referred to as a *data cell*. Priority number P consists of an empty-cell indicator bit e , and priority value which itself is arranged from the most significant bit (MSB) to the least significant bit (LSB). Therefore,

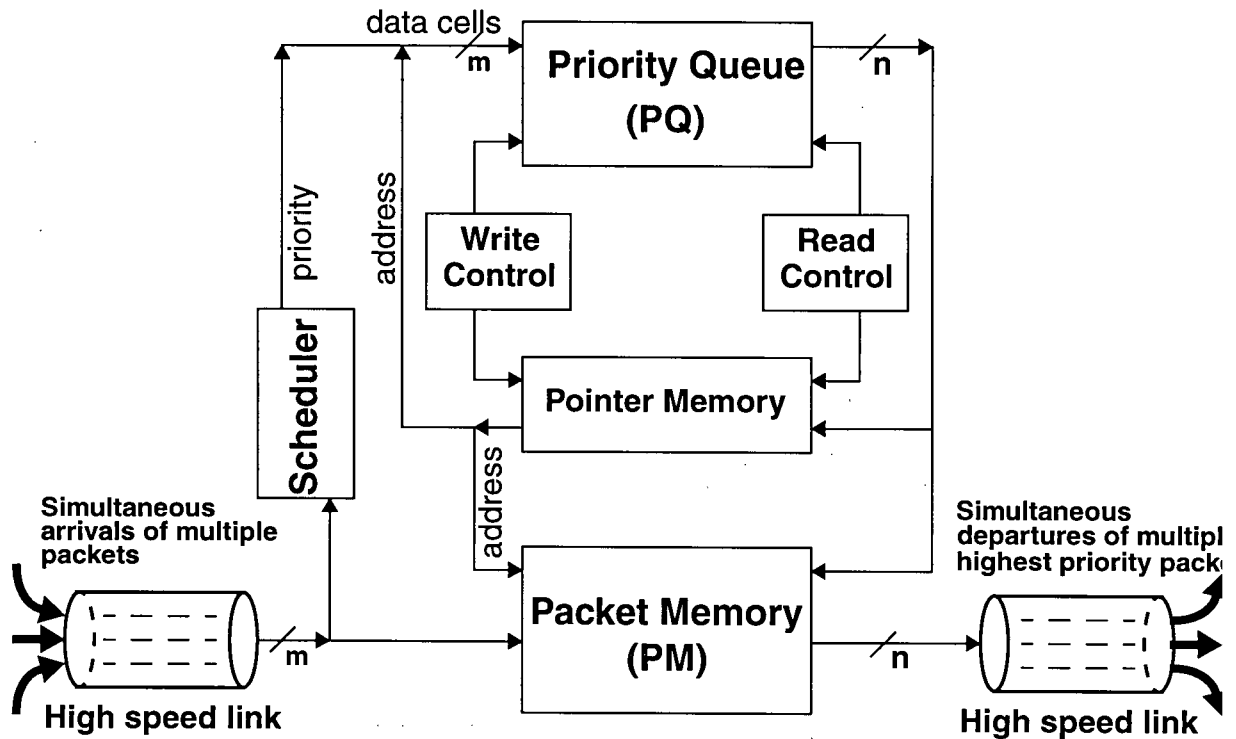


Fig. 5.1: A typical scheduling network

empty data cells (i.e. for which $e=1$) can be considered to have the lowest priority. The PQ maintains a sorted list of data cells according to their P numbers. The result can be easily extended to the multi-priority case [35]. The next section outlines the basic word-model of our proposed systolic parallel priority queue. The queue architecture in details is also presented in Section 5.3. Then, a retimed design of the queue is described in Section 5.4.

5.3 Systolic Parallel Priority Queue

A PPQ maintains prioritized access to the data it contains at all times. Incoming data cells are injected into the PPQ from each of its *input ports* and the highest priority data cells are read from a set of *output ports*. In a non-systolic (or non-pipelined) implementation, the priority queue con-

tains, at any given time, a ranked set of data cells. In a systolic implementation, ranking data may be overlapped with data access as long as we assure that the data cells accessed in any time are the highest priority ones. This conforms with the general definition of a priority queue that was given in Section 5.1. Overlapping ranking with data access enhances throughput substantially. The ultimate goal of the systolic PPQ design is to allow $O(1)$ -time access to the head data elements (cells) in the PPQ. In other words, the access time to the head cells in the PPQ is fixed and independent of the PPQ size.

The PPQ generalizes the operation of a sequential PQ in the following manner. Whenever a set of m new data cells is inserted into the PPQ, the queue outputs the n data cells with the n smallest priority P values among the stored and incoming data cells. We are interested in the realization of a priority queue that inputs m steady streams of data cells and outputs n steady streams of data cells in $O(1)$ -time. For simplicity, we focus mainly on the $m = n$ case in this chapter, but the PPQ architecture can be easily generalized to the case where $m > n$ as will be discussed in Chapter 6.

Fig. 5.2 shows a priority queue $PPQ(m, m, w)$ (or for simplicity $PPQ(m, w)$) which denotes a systolic PPQ with m inputs, m outputs, and depth w , i.e. it can store up to $m \times w$ data cells. In a single unit of time, $PPQ(m, w)$ is capable of reading m data cells, and delivering the m highest priority data cells at its output ports. In Fig. 5.2, I_j^i denotes the data cell received on input port j in time-slot i . Similarly, O_j^i denotes the data cell delivered on output port j in time-slot i . Note that here a cell period is equal to one time-slot.

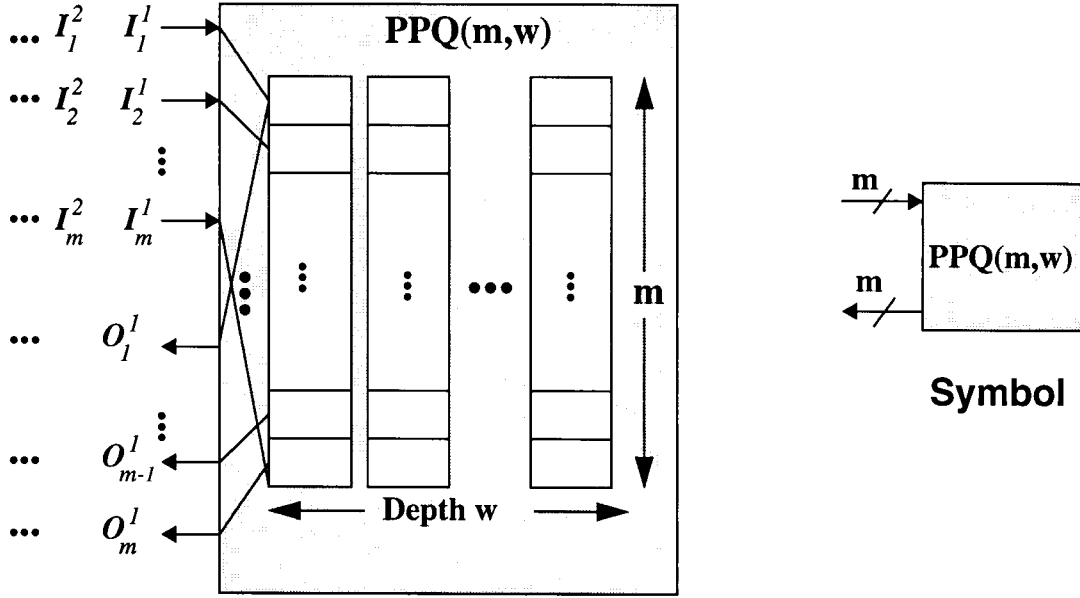


Fig. 5.2: A PPQ contains $m \times w$ cells

This chapter assumes a synchronous VLSI model [41]. In a systolic PPQ, all data cells are processed and transferred in a pipelined fashion. This implies that the PPQ consists of combinational logic stages separated by clocked registers. Note the difference between a clock period which defines one *time-unit* versus a cell period which defines one *time-slot*. Each combinational stage, with its associated registers, form a *pipeline stage*. As will be described later, each pipeline stage of the PPQ consists mainly of some registers and a *parallel sorter*. The PPQ accommodates complete data cells in these *registers*. The incoming m data cells find their right places within the PPQ through a series of sorting steps. Through this chain of sorting processes, not only do the new data cells find their positions, but also the entire PPQ rearranges itself. Subsequent arrivals of new data cells trigger a new series of concurrent sorting procedures and reordering of the PPQ as described in the next section.

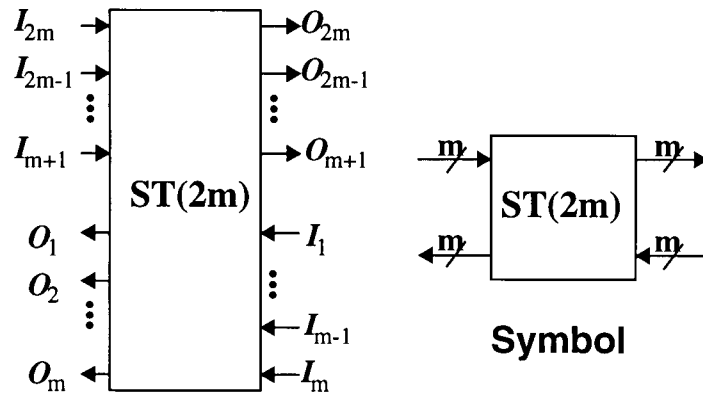
5.3.1 VLSI Word-Model of a Parallel Sorter

As mentioned above, each pipeline stage of a PPQ consists mainly of a parallel sorter and register stages. This section describes the basic functionality of a parallel sorter in a VLSI word-model [41], [56], where the l bits representing one data cell are input or output through the same set of I/O ports (i.e. l parallel data-paths). Under this model, one time-slot is equal to one time-unit (i.e. the cell period is the same as a clock period), and it is possible to store complete l -bit data cells in the registers of a sorter, and to sort a fixed number of l -bit data cells in one time-unit [25], [57], [58]. Details on the iterative construction and systolic timing of parallel sorters in a VLSI bit-serial model are covered in detail in Chapter 6, where it will be shown that the parallel sorter itself has a systolic structure to maintain high throughput.

The input-output (I/O) configuration of a $2m$ -cell sorter $ST(2m)$ is illustrated in Fig. 5.3. At time-slot $t = i$, the data cells incoming on the m inputs of the sorter from the left and the m data cells from the right inputs are compared. Then the m highest priority data cells are output to the left, and the m lowest priority data cells are output in the proper order to the right at time $t = i + \Delta(2m)$, where $\Delta(2m)$ is the *delay* of $ST(2m)$ and is assumed to be smaller than a single time-slot.

5.3.2 Recursive Construction of Systolic PPQs

Fig. 5.4 shows the structure of $PPQ(m, 1)$, a parallel priority queue of depth one. $PPQ(m, 1)$ is constructed from a single $2m$ -cell sorter $ST(2m)$ whose right-side outputs are recirculated to its right-side inputs through a single stage of m registers in a parallel arrangement. The left-side out-



$$O_1 \leq O_2 \leq \dots \leq O_m \leq O_{m+1} \leq \dots \leq O_{2m}$$

Fig. 5.3: The I/O configuration of a $2m$ -cell sorter



puts of $ST(2m)$ are also delivered to the output ports of $PPQ(m, 1)$ through a single stage of m registers. Note that in Fig. 5.4, I_j^i denotes the data cell received on input port j in time-slot i , and \underline{I}^i denotes a vector of m data cells received on the m input ports of the priority queue in time-slot i . In each time-slot, the m data cells in the PPQ and the m data cells incoming from the left-side inputs are compared, and the m highest priority data cells are output to the left and are saved in the m L-registers on the left-side. The m lowest priority data cells are output to the right where they become the new contents of the priority queue (i.e. the contents of the m R-registers in the right-side). Note that this implies the use of edge-sensitive or master-slave type memory devices as registers (or delay elements D). The $ST(2m)$ block shown in Fig. 5.4 is a combinational sorter that can sort (or merge) two sets of m input cells according to their priority value. The sorter pro-

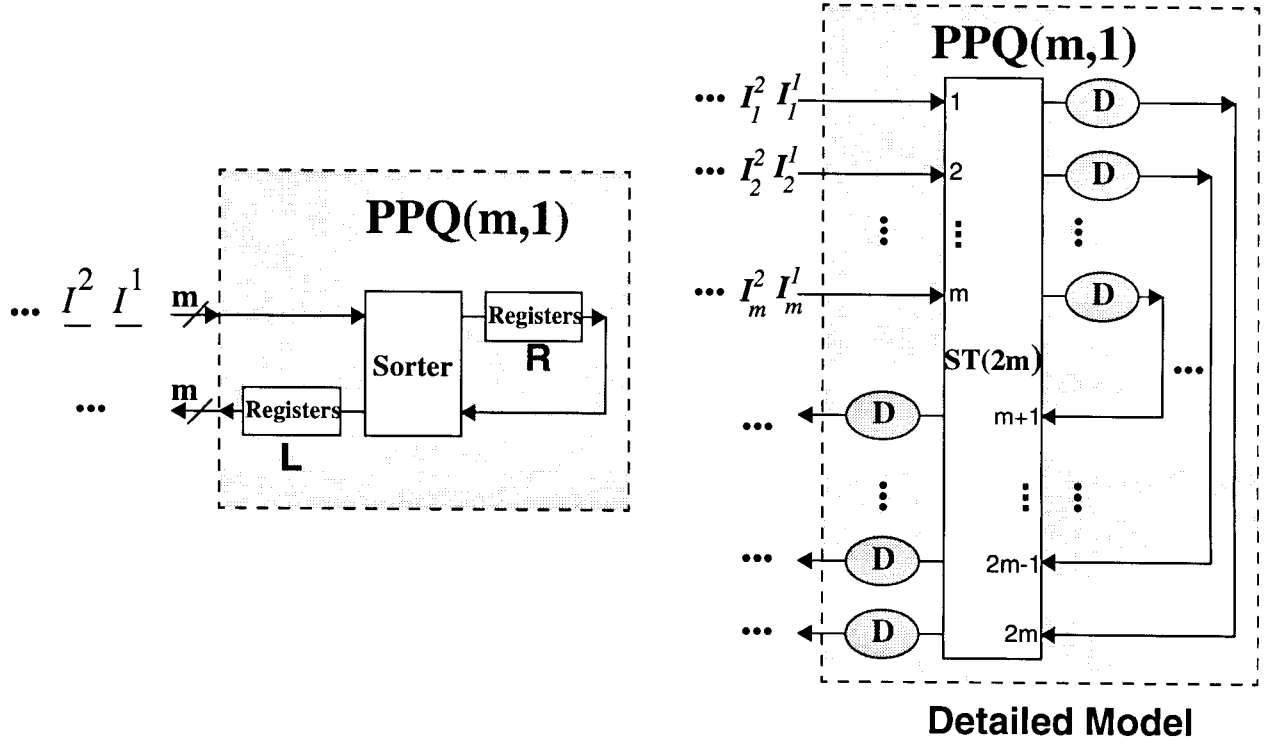


Fig. 5.4: A priority queue with m inputs, m outputs, and depth l ($PPQ(m, 1)$)

duces two sets of m outputs, the left-ward outputs produce the m highest priority cells, while the right outputs produce the m lowest priority cells.

In the following we provide a proof of correctness for the operation of a $PPQ(m, w)$. We start by observing that a $PPQ(m, 1)$ sorts data in the right order and maintains the m highest priority cells in its output L-registers as explained above. Now assuming that $PPQ(m, w - 2)$ operates correctly, we proceed to show (by induction) that $PPQ(m, w)$ will also maintain the correct order of data.

Observe that a parallel priority queue $PPQ(m, w)$ of depth w can be constructed recursively by

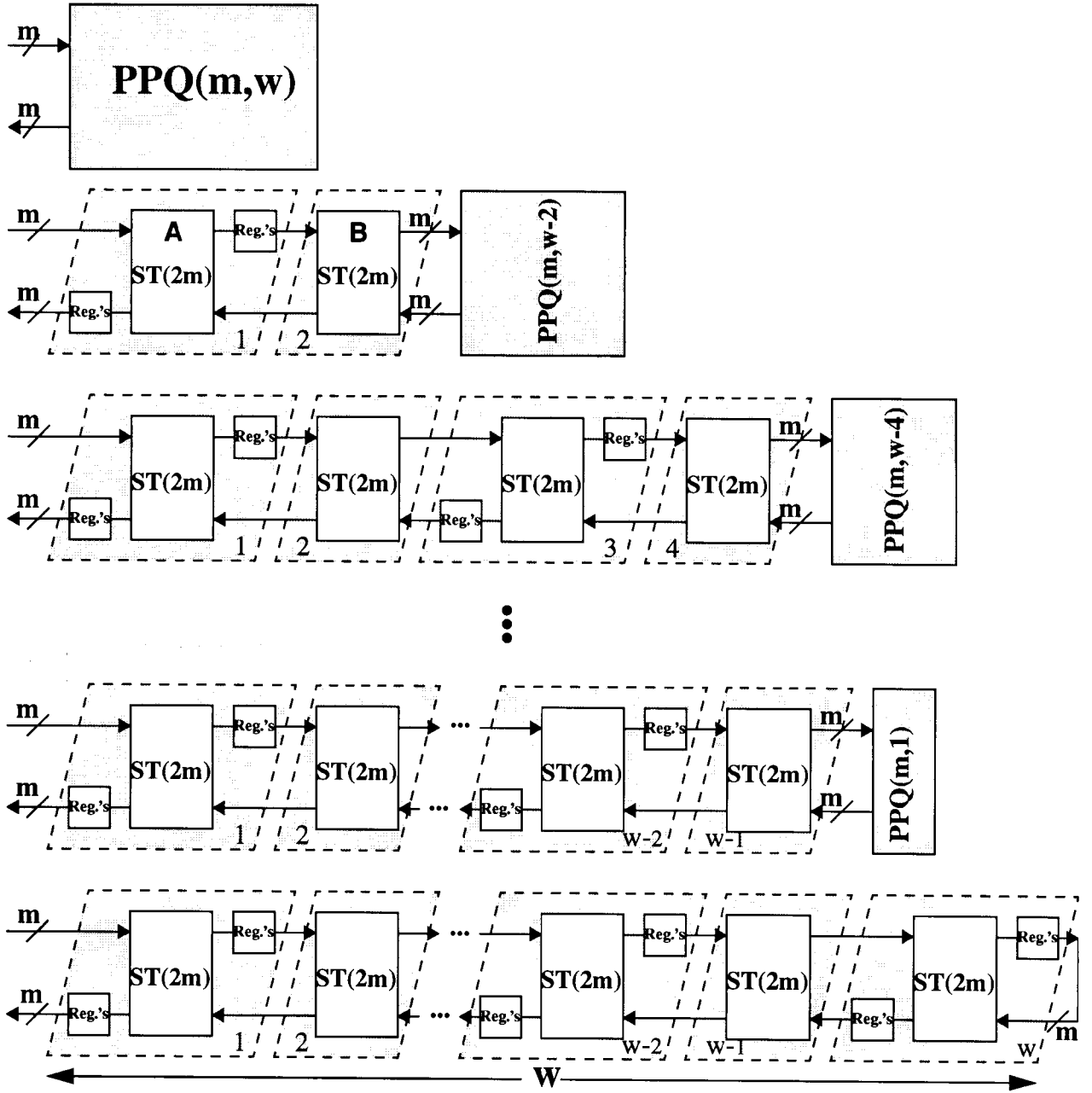


Fig. 5.5: Recursive construction of the parallel priority queue $PPQ(m, w)$

cascading a front block consisting of two $2m$ -cell sorters (labeled A and B) and two banks of m registers with $PPQ(m, w-2)$, a priority queue of depth $w-2$, as shown in Fig. 5.5. Our con-

struction method requires that w be an odd integer. The operation of the queue is as follows: first, the m highest priority data cells coming to the B sorter (in stage 2) from $PPQ(m, w - 2)$ are compared to the m data cells in the registers on the right-side of the A sorter (in stage 1). After sorting, the m highest priority cells from sorter B are sent to be compared with the m incoming data cells on the input ports of the A sorter in stage 1. At the same time, the m lowest priority data cells from the B sorter are passed to the right-side where they are pushed into $PPQ(m, w - 2)$. Pipeline timing ensures that incoming data cells from the left-side of the B sorter in stage 2 rendezvous with data cells coming to the sorter from $PPQ(m, w - 2)$ where they are compared. In the same time-slot, the A sorter in stage 1 compares m data cell received from stage 2 with m incoming data cells on its input ports. The A sorter, then outputs the m highest priority data cells to the left, and the m lowest priority data cells to the right where they are saved in the registers. The above explanation completes an induction proof of correctness for the operation of $PPQ(m, w)$.

Priority queue $PPQ(m, w - 2)$ can itself be realized recursively from two $2m$ -cell sorter $ST(2m)$ and associated registers, and a priority queue $PPQ(m, w - 4)$, a priority queue of depth $w - 4$. The recursive construction will result in a realization of $PPQ(m, w)$ from $w - 1$ $2m$ -cell sorters $ST(2m)$, and a $PPQ(m, 1)$ priority queue realized by a sorter $ST(2m)$ and two banks of m registers, as was shown in Fig. 5.5. Essentially, the realization is a bidirectional pipeline array of w consecutive $2m$ -cell sorters $ST(2m)$ and associated registers, as shown at the end of Fig. 5.5.

The architecture can be described in another manner. Each pair of consecutive sorter stages oper-

ate on $2m$ inputs and produces $2m$ outputs in one time-slot. The outputs of every other sorter are latched, and the registers are clocked so that when several of these elements are interconnected, the changing output of one will not interfere with the input to another. Fig. 5.5 shows how these stages are interconnected to make priority queues of depth w . Note that the total latency in the PPQ is limited to the latency in the first sorter, which selects the departing (highest priority) data cells from the PPQ just after the arrival of the new data cells in the PPQ.

5.4 Retimed Design

Further scrutiny of the $PPQ(m, w)$ design of Fig. 5.5 reveals that even though the PPQ is systolic, it actually consists of $\frac{w-1}{2}$ *semisystolic processors* or *Mealy-machines* (in stages 2, 4, ..., $w-1$), and $\frac{w+1}{2}$ *systolic processors* or *Moore-machines* (in stages 1, 3, ..., w) [59]. In a systolic processor, the output of the combinational logic of the parallel sorter is directly latched into registers within the processor [59]. In other words, the outputs of the processor emanate directly from registers. In a semisystolic processor, the outputs of the processor are allowed to flow directly from combinational logic too.

The flexibility afforded by this semisystolic processor model simplifies the task of designing implementations of the PPQ. The price to pay is that the time required to carry out any step of the sorting algorithm will be at least as long as the time required for data cells to ripple through a pair of sorters between two registers. The minimum time-slot period is lower bounded by this ripple delay. In this section, we describe a retimed PPQ design in which each stage of the PPQ is a systolic processor. In the retimed design the time-slot is lower bounded by propagation delay through

a single sorter.

The semisystolic model can be lagged (or retimed) to form an equivalent systolic network [59].

The retiming problem can be solved by inserting another register (i.e. a delay element D) along each connection from an output to an input of the $ST(2m)$ sorter to construct $PPQ(m, 1)$ as shown in Fig. 5.6, where I_j^i and \underline{I}^i are as defined in Section 5.2. As a result of inserting the

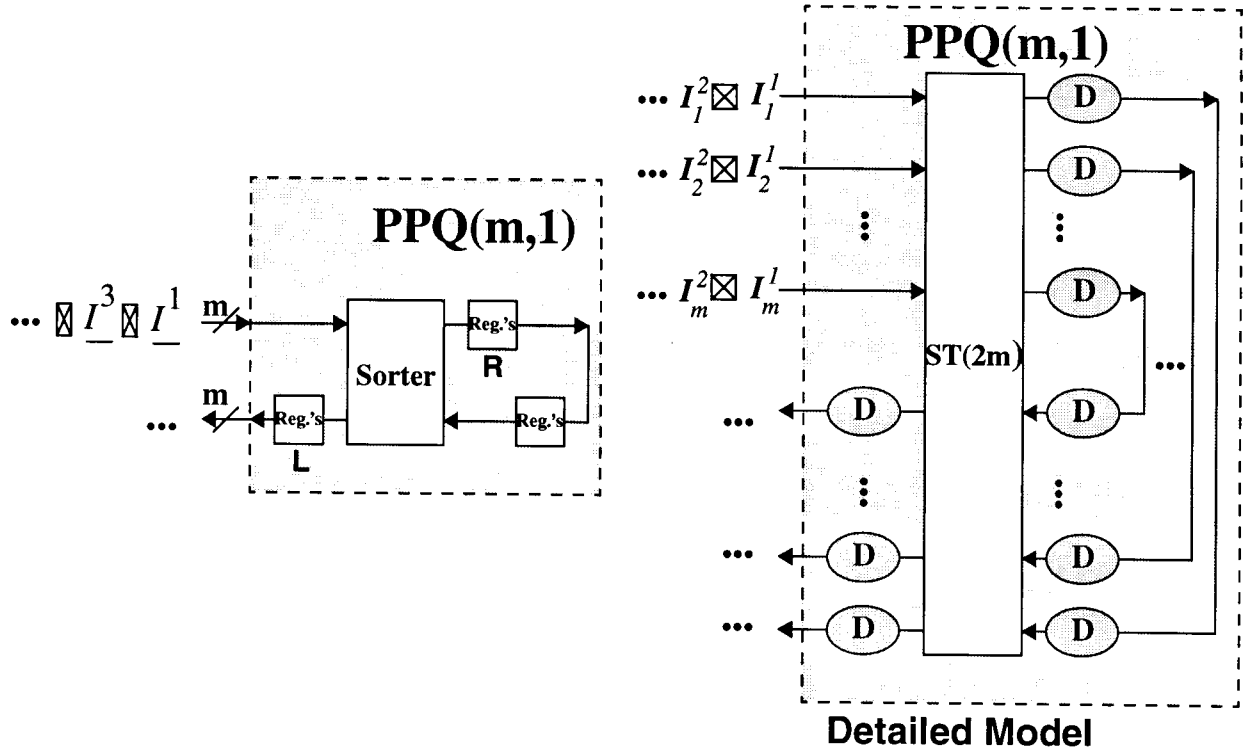


Fig. 5.6: Retimed design of $PPQ(m, 1)$

delay elements (i.e. registers), the input arrivals must be rescheduled so that the correct cells are compared by the sorter in each step. In particular, the incoming data cells are now inserted to the

network in every other time-slot (instead of in each time-slot).

The retimed realization of the $PPQ(m, 1)$ which operates on a parallel stream of width m data cells separated by one time-slot as shown in Fig. 5.6. The factor-of-two slow-down in the input rate is due to the retiming transformation that was applied to drive an equivalent systolic network (consisting of only systolic processors). As suggested in [59], we can recover the loss in efficiency by realizing two priority queues simultaneously on the same systolic architecture. Although, there is no extensive difference between the realizations shown in Fig. 5.4 and Fig. 5.6, it can be easily shown that only the latter one can be used for the recursive construction of a systolic priority queue with systolic processors, as shown in Fig. 5.7. Note that all stages (processors) of the $PPQ(m, w)$ of Fig. 5.7 are Moore-machines. A proof of correctness for the operation of the queue can be completed by induction (similar to the proof in Section 5.2).

Note that in the retimed design, at any given time-slot (i.e. step) half of the sorters are not active because they receive empty (or invalid) data cells at the registers connected to their inputs. The other half becomes inactive in the following step. This alternating operation occurs often in the design of systolic systems as a result of retiming, and is an unavoidable result of the slow-down design technique [59]. Note that the queue rearranges itself after the arrival of each wave of new data cells.

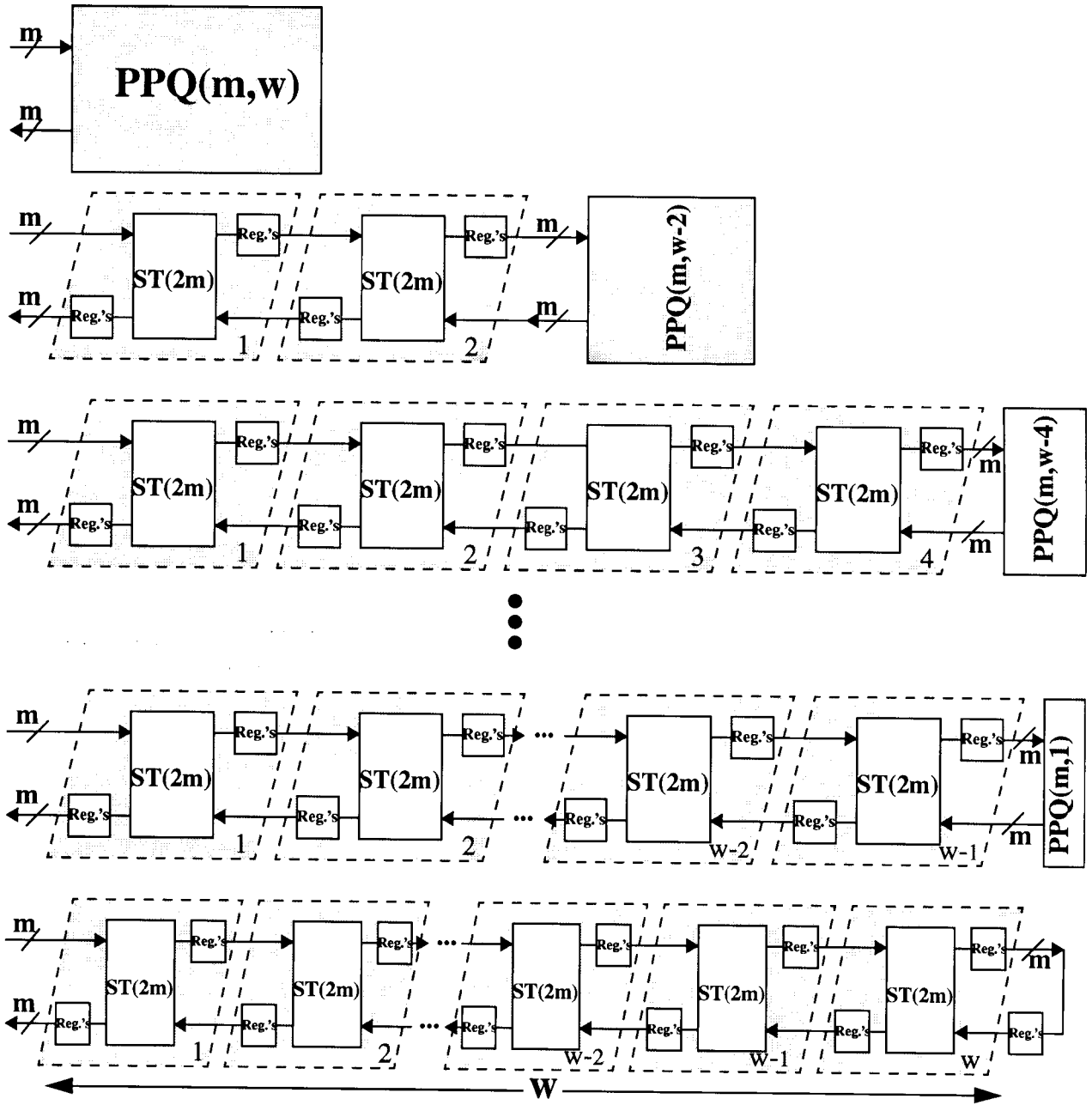


Fig. 5.7: Recursive construction of the systolic parallel priority queue $PPQ(m, w)$

5.5 Summary

This chapter presented a parallel priority queue (PPQ) which can be scaled to meet the requirements of ultra high-speed links using standard CMOS technology. Incoming data cells are injected into the PPQ from each of its *input ports* and the highest priority data cells are read from a set of *output ports*. The PPQ maintains prioritized access to the data it contains at all times, and the access time to the data is fixed and independent of the PPQ size, i.e. $O(1)$ -time access.

The presented design in this chapter is actually the generalization of the priority queue concept with steady streams of parallel inputs and outputs, i.e. the same input and output rates. The next chapter develops a novel rate-adaptation feature to allow different input and output rates. This is an important feature because in practice the output rate of the queue is controlled by the available link bandwidth which may vary (or even becomes zero) independent of the packet arrival rate. The next chapter also presents an area-efficient, systolic design of the PPQ for VLSI implementation.

CHAPTER 6

Scalable PPQs with Output Rate-Control

6.1 Introduction

The proposed systolic parallel priority queue (PPQ) in Chapter 5 is the realization of a queue that inputs and outputs steady streams of data cells. However, in many real applications, it is desirable to decouple the input process to the PPQ from the output process. In this chapter we have included an output rate-adaptation feature to allow the PPQ bandwidth (measured in packets per second) to be varied dynamically; for example based on the congestion in the upstream link. The chapter also presents an area-efficient, systolic design of the modified PPQ for VLSI implementation. We show how the operations of the systolic PPQ can be considered as composite processes which are realized in bit-serial pipelined circuits [27].

The modified PPQ is rate-adaptive in the sense that the PPQ operates correctly even when the queue input rate and output rate are different. This is an important feature because in practice the output rate of the queue is controlled by the available link bandwidth which may vary (or even becomes zero) independent of the packet arrival rate. This decoupling of the input and output packet flow rates is a distinguishing feature of our PPQ concept and has not been addressed in previous literature.

Rate-adaptation is realized by an inhibit control signal that blocks output ports while allowing injection of the data cells into the PPQ. This means that the output rate can be controlled by back pressure signals from other stages in the network due to lack of buffer space or due to link congestion. It is shown that because the operation of the systolic PPQ is pipelined, no degradation occurs even when the output ports are blocked. The inhibit feature is dictated by an INHIBIT control signal which disables the read PPQ operation. If the INHIBIT signal is asserted during a clock cycle, the PPQ will accept m incoming cells but will not flush out the m highest priority cells (see Fig. 5.2). INHIBIT is a global control signal which dictates the inhibit configuration inside the PPQ.

The rest of the chapter is organized as follows. The queue architecture with output-rate control and its main features in details are presented in Section 6.2. Section 6.3 describes how output rate-adaptation can be incorporated in the retimed PPQ. Section 6.4 addresses the case when the number of parallel inputs to the PPQ is larger than the number of outputs. The bit-serial realization of the queue is investigated in Section 6.5. Then, we elaborate on the basic core elements of the queue, i.e. parallel sorters, in Section 6.6. The timing details for a primitive sorter during normal operation are presented in Section 6.7. In Section 6.8 VLSI design issues on the implementation of the queue are discussed, followed by concluding remarks in Section 6.9.

6.2 A PPQ Design with Output Rate-Control

In many real applications, it is desirable to decouple the input process to the PPQ from the output process. Specifically, the PPQ outputs may be intermittently blocked from sending cells due to a link congestion. This process is usually called output rate-adaptation, and it requires significant modifications to the PPQ architecture. To provide rate-adaptation capability to the PPQ, we strategically place a bank of m multiplexers (MUXs) and m demultiplexers (DMUXs) between pipeline stages. The MUXs and DEMUXs are controlled by a global INHIBIT signal. The resulting design is illustrated in Fig. 6.1 for $PPQ(m, 1)$. If the output ports are blocked (i.e. where

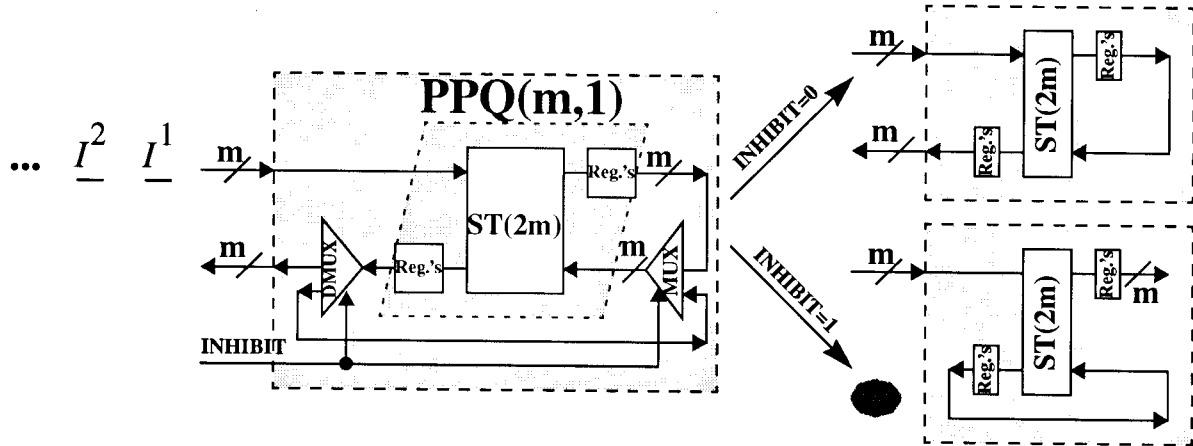


Fig. 6.1: $PPQ(m, 1)$ with inhibit feature

$INHIBIT=1$), while cells are still arriving at the PPQ, then the m highest priority data cells are routed to the right where they become the new contents of the PPQ, and the m lowest priority data cells are discarded.

For larger size PPQs, as illustrated in Fig. 6.2, the INHIBIT signal indicates that the queue should not move forward (i.e. leftwards in Fig. 6.2), while still implementing the sorting functions required to insert incoming cells in the proper positions and maintain the appropriate ordering in the queue. Depending on the value of the INHIBIT signal (0 indicates no inhibit, and 1 indicates inhibit), the structure of the PPQ is reconfigured using MUXs and DMUXs as shown in the figure. Note that because of the pipeline operation of the PPQs, no slow-down occurs even when the output ports are inhibited. Note that a proof of correctness for the operation of a $PPQ(m, w)$ with output-inhibit control can be also provided by induction (as in the previous case).

We will explain the operation of the PPQ by means of an example. Several steps of the operation of a $PPQ(1, 5)$ are shown in Fig. 6.3. In this example, we select the simple one input, one output priority queue of depth 5. As shown, each incoming data cell is tagged by its priority number. Fig. 6.3 illustrates data propagation between consecutive sorters and the status of the queue after the arrival of each new data cell. Each *step* represents the queue structure right after a rising edge clock signal until right before the next rising edge. The small numbered-squares represent the priority numbers of data cells which are stored in the PPQ registers, while crossed squares indicate empty registers. The registers are edge-sensitive flip-flops; they latch the data with the rising-edge of the clock signal at the beginning of each time-slot. The numbered circles represents the priority number of data cells which are delivered to the output link by the queue. If the queue receives an INHIBIT=1 signal, it will be reconfigured by MUXs and DMUXs (not shown in Fig. 6.3), so as to prevent the delivery of any data cells to the output until INHIBIT is deactivated, i.e. INHIBIT=0. Note that the state of the INHIBIT signal must be received by the queue before the rising-edge of the clock. The solid circles indicate the blocking situation dictated by the network.

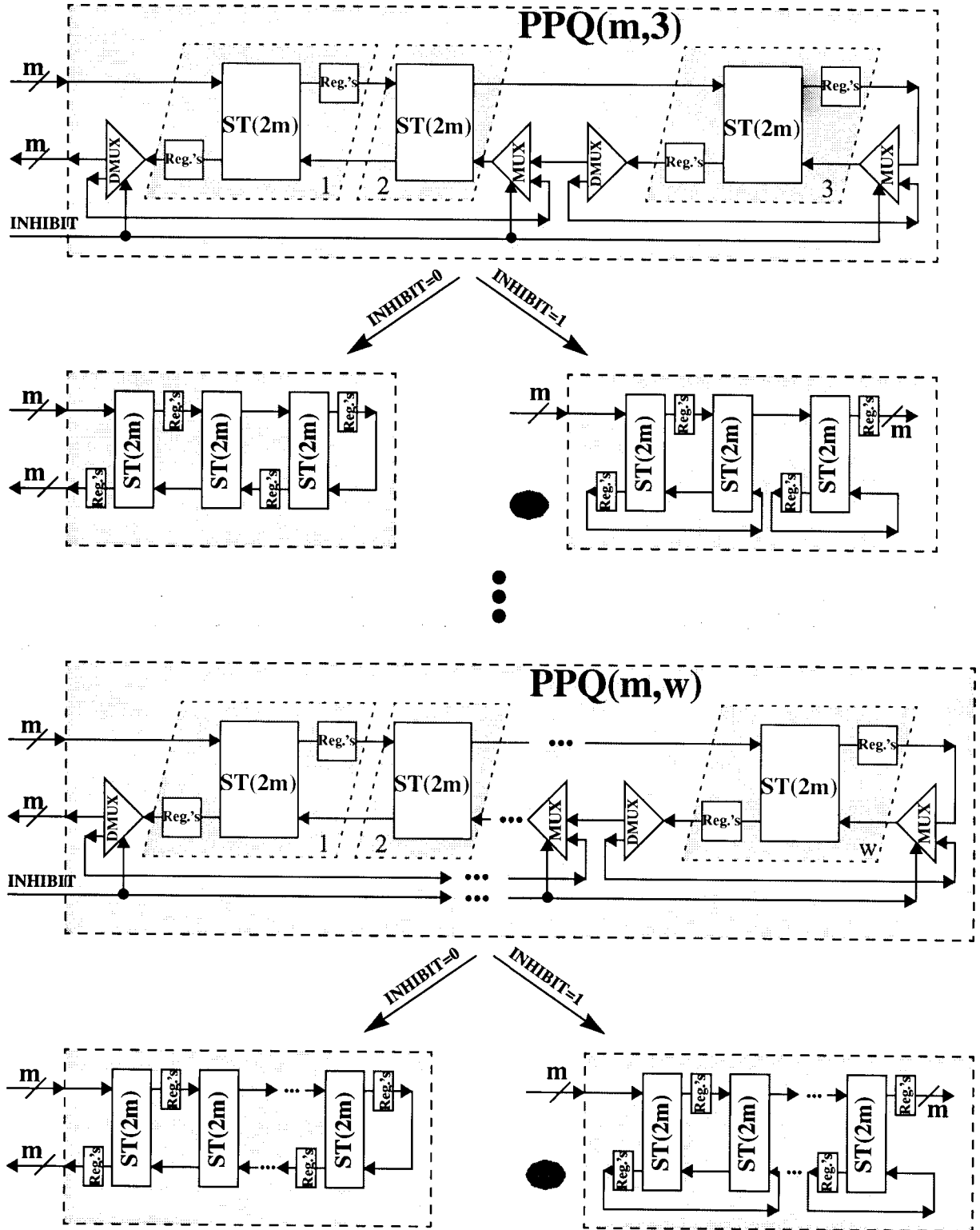


Fig. 6.2: Building recursively priority queues with different depth sizes and inhibit feature

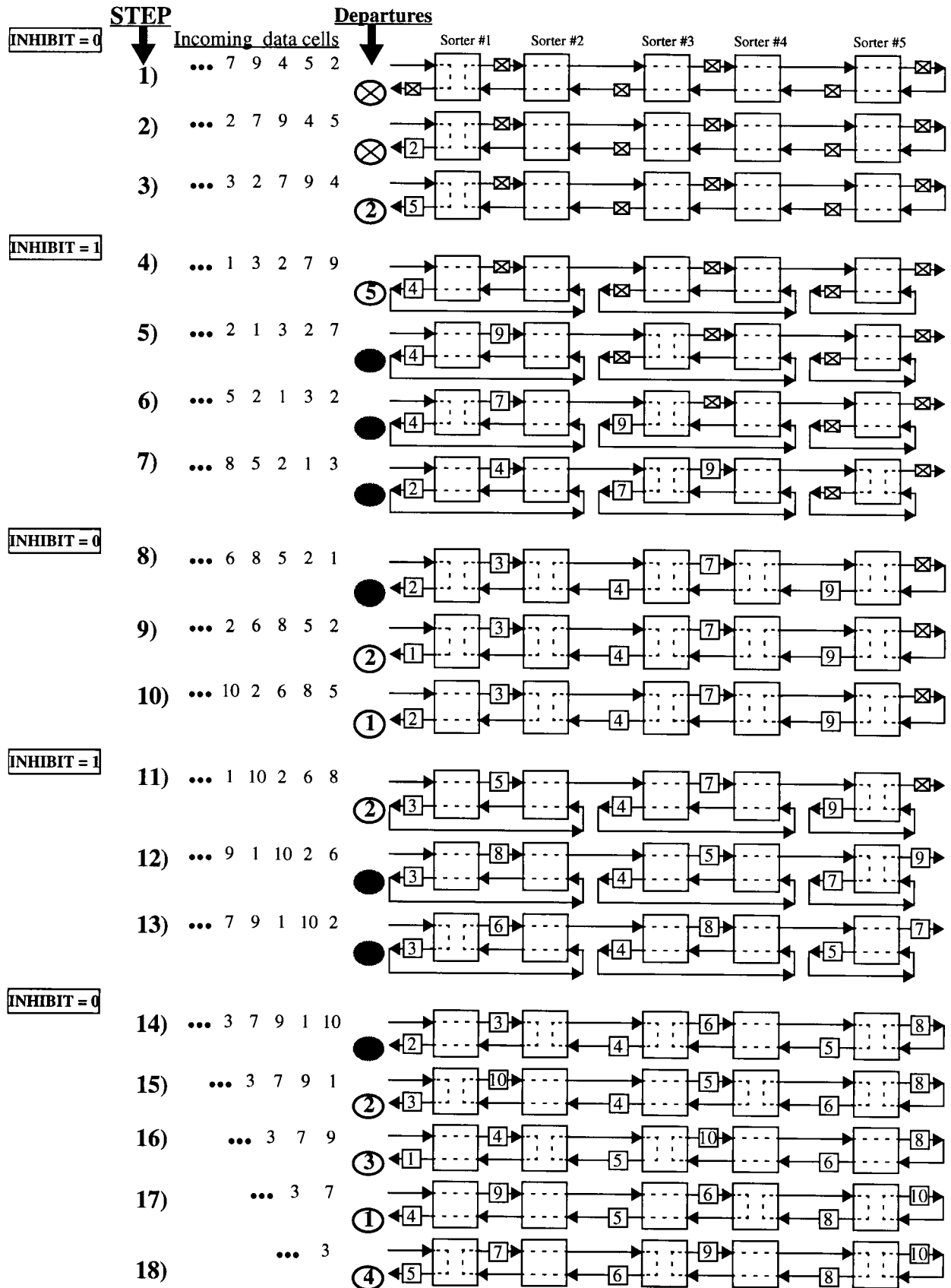


Fig. 6.3: Several steps of the operation of a $PPQ(1, 5)$

Initially, it is assumed that the queue is empty and its output is not blocked. As long as $\text{INHIBIT}=0$, the data cells are delivered to the output port and then to the following stage of the network. In any given time-slot, data is exchanged between two consecutive sorters. For example, in each time-slot, sorter #2 sorts its inputs from the two registers connected to its input ports, and passes the higher priority data cell to left. Then, this data cell is compared with a newly arriving data cell in sorter #1. The same sorting procedure is applied by sorters #2, and #3, by sorters #3, and #4, and by sorters #4, and #5.

In step 4 (just before the rising-edge of the clock), an $\text{INHIBIT}=1$ signal is issued indicating that no data cells can be output from the queue in the next step (step 5). This causes the queue to reconfigure itself through MUXs and DMUXs after receiving the INHIBIT signal. Note that the queue rearranges itself after the arrival of each new data cell and keeps the new configuration as long as its output is blocked (step 4-7). Because up to this point the blocking does not persist too long, the queue becomes just half full during these steps and no overflow (discard) happens.

In step 8 (just before the rising-edge of the clock), the queue receives an $\text{INHIBIT}=0$ signal and subsequently reconfigures itself for normal operation. In step 9, the queue delivers the highest priority data cell which has been stored in the queue with priority number 2 to the network.

In step 11 (just before the rising-edge of the clock), another $\text{INHIBIT}=1$ signal is issued to the queue and the queue outputs data cell with priority number 2 after the rising-edge of the clock. Now, the queue is reconfigured again because of its blocked output. In steps 12 and 13, while the network still injects new data cells in the queue, the queue has to discard two data cells with priority numbers 9, and 7. It is worth mentioning that cell discard can be avoided if traffic shaping or congestion control is enforced in the network.

In step 14, an INHIBIT=0 signal is issued and the queue gets back to its normal configuration and gives access to highest priority data cell to the network in each step. The above example shows that the PPQ rearranges itself after arrival of new data cells. This rearrangement spreads into the queue like a wave.

6.3 The PPQ Retimed Design with Output Rate-Control

Output rate-adaptation can be also incorporated in the retimed PPQ using the modified design of Fig. 6.4 for $PPQ(m, 1)$. Depending on the value of the INHIBIT signal (0 or 1), the structure of

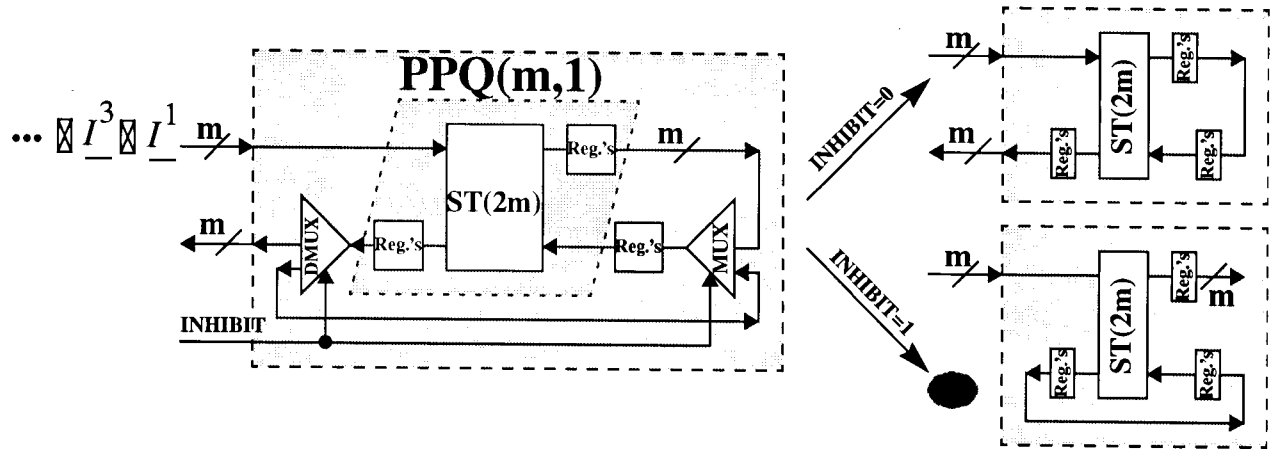


Fig. 6.4: Retimed design of $PPQ(m, 1)$ with inhibit feature

the PPQ is reconfigured using MUXs and DMUXs. For larger size PPQs, a global INHIBIT signal is employed as before to prevent the PPQ from forwarding data cells through its output. The recursive construction of such a PPQ is illustrated in Fig. 6.5.

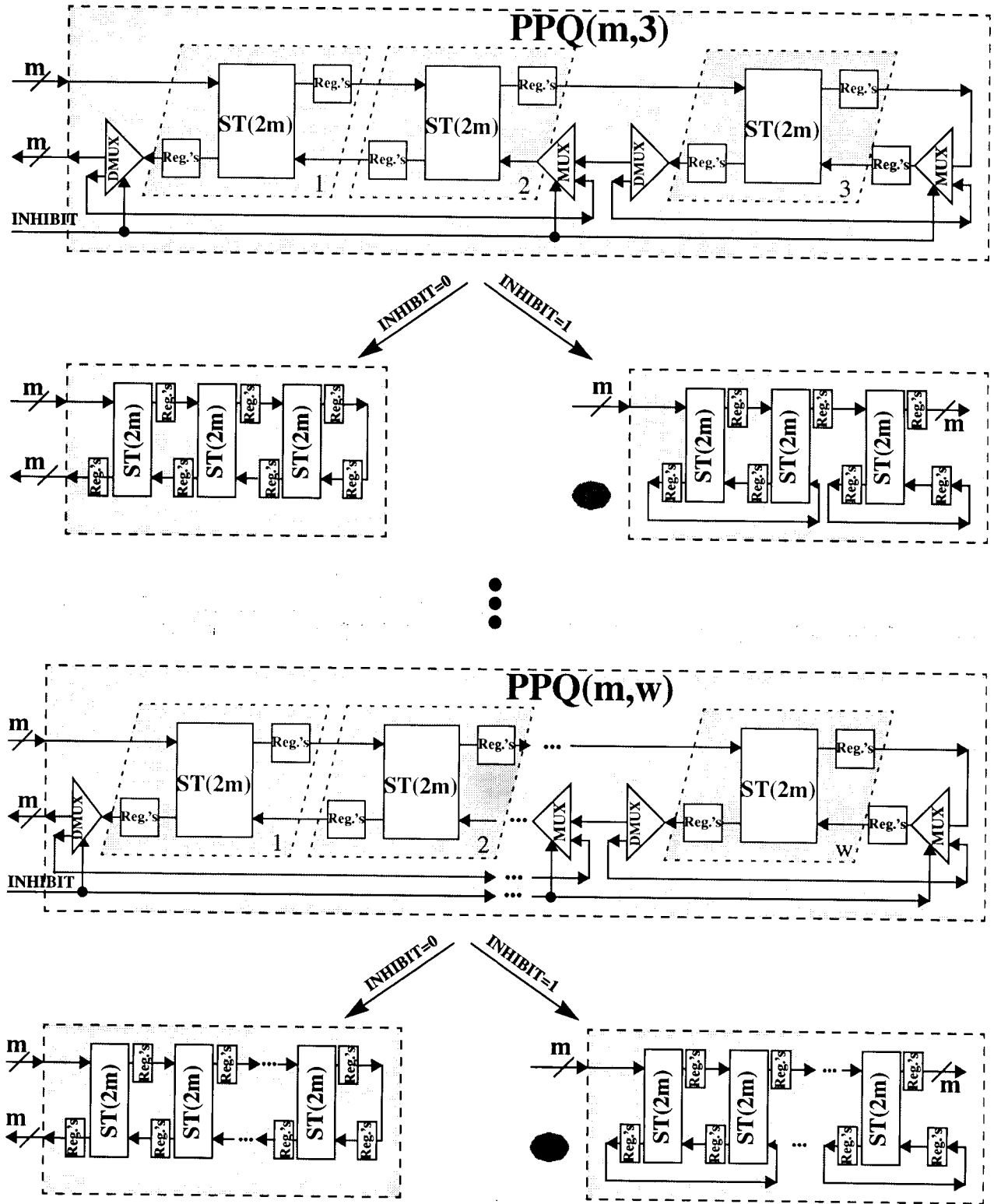


Fig. 6.5: PPQs with different depth sizes and inhibit feature

The dynamic reconfiguration of the retimed PPQ can be illustrated by means of an example. Several steps of the operation of a two inputs, two outputs priority queue of depth 3, i.e. $PPQ(2, 3)$, are shown in Fig. 6.6 using the same notations as the example of Fig. 6.3. Note that in the retimed design, at any given time-slot (i.e. step) almost half of the sorters (viz. shaded sorters) are not active because they receive empty (or invalid) data cells at the registers connected to their inputs. The other half becomes inactive in the following step. This alternating operation occurs often in the design of systolic systems as a result of retiming, and is an unavoidable result of the slow-down design technique [59]. Note that the queue rearranges itself after the arrival of each wave of new data cells and keeps the new configuration as long as its output is blocked (steps 5-10). Because up to this point the blocking does not persist too long, the queue becomes half full during these steps with no overflow.

In step 11 (just before the rising-edge of the clock), the queue receives an $INHIBIT=0$ signal and subsequently reconfigures itself for normal operation. In step 15 (just before the rising-edge of the clock), another $INHIBIT=1$ signal is issued, and the queue will be reconfigured because of its blocked outputs. Because the PPQ is full, the injection of any new data cells will cause some older lower priority cells at the queue tail to be discarded. In steps 16 and 18, data cells with priority numbers 12, 13, 9, 10 are discarded. In step 19, an $INHIBIT=0$ signal is received and the queue goes back to its normal configuration.

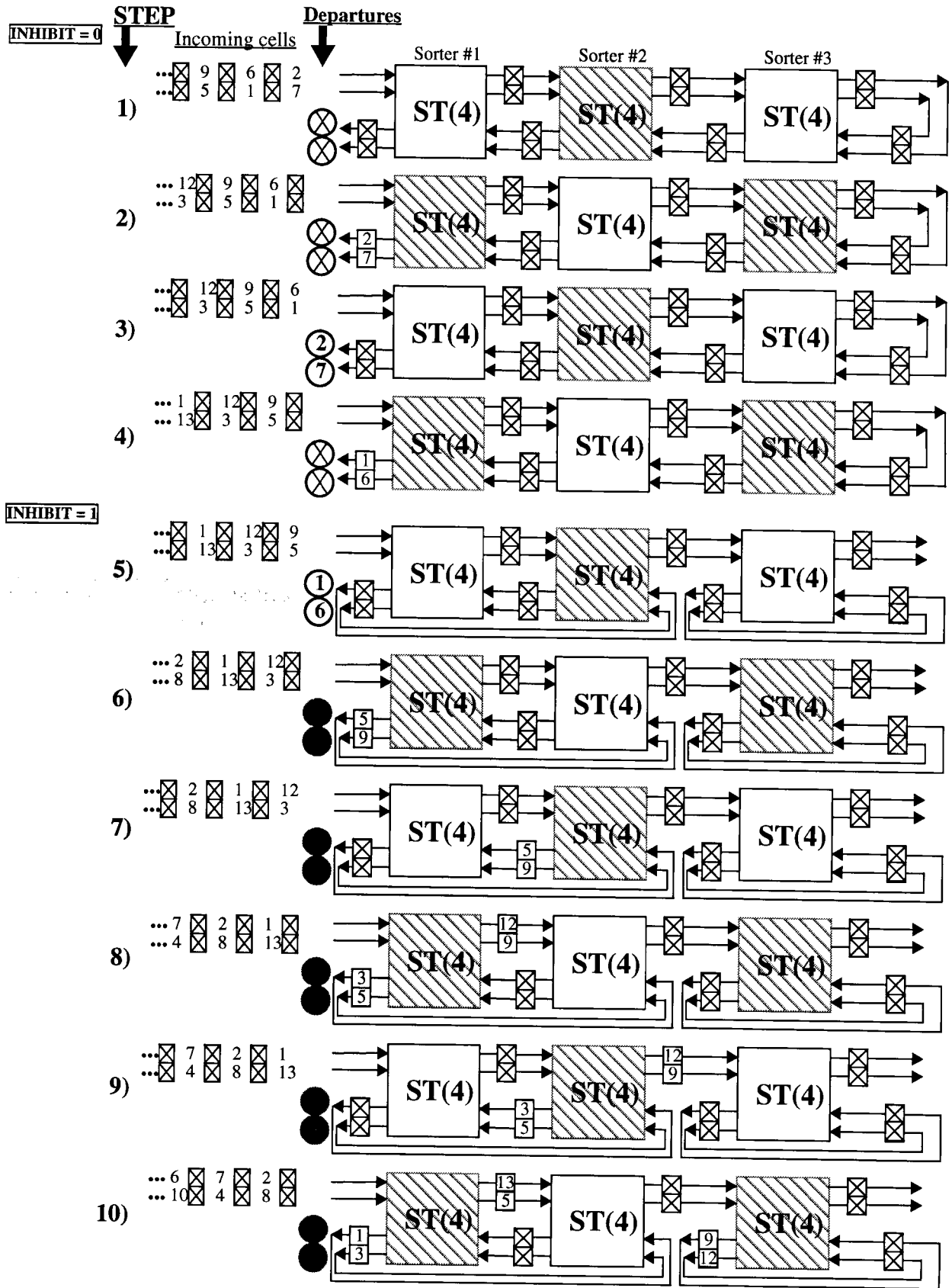


Fig. 6.6: Several steps of the operation of a systolic $PPQ(2, 3)$

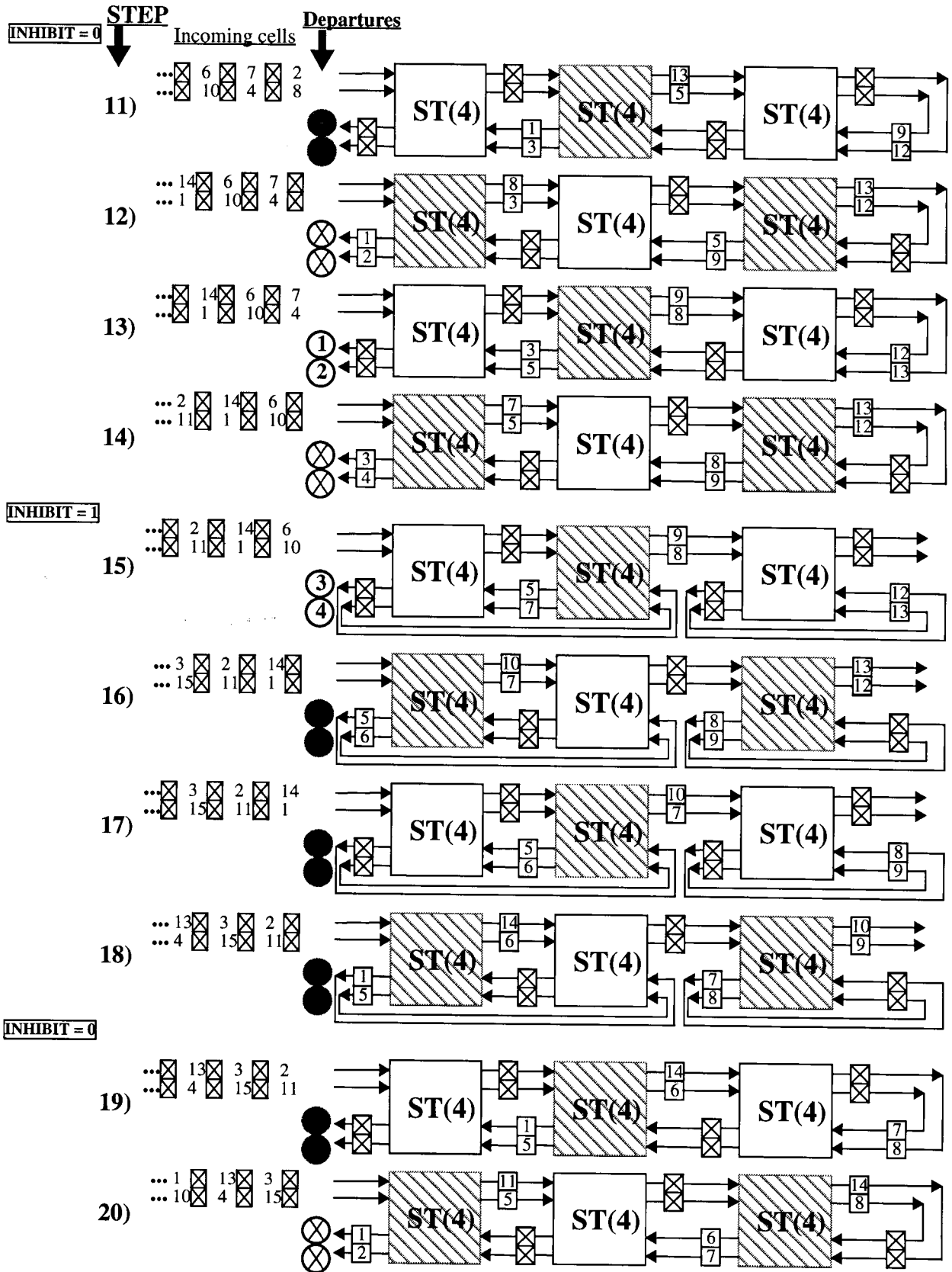


Fig. 6.6 (cont'): Several steps of the operation of a systolic $PPQ(2, 3)$

6.4 Systolic PPQs with an Unequal Number of Inputs and Outputs

We now discuss the case when the number of parallel inputs to the PPQ is larger than the number of outputs. Fig. 6.7 shows a $PPQ(m, n, w)$ which is a systolic PPQ with m inputs and n outputs and depth w , where $m > n$, and w is an odd integer. Instead of using $2m$ -cell sorters $ST(2m)$, we employ $ST(m + n)$ sorters in this structure. Note that each $ST(m + n)$ has $m + n$ inputs and $m + n$ outputs. In each step, data cells incoming on the m inputs of the sorter from the left and the n data cells from the right-side inputs are compared. Then the n highest priority data cells are output to the left, and the m lowest priority data cells are output to the right.

As illustrated in Fig. 6.7, the basic $PPQ(m, n, 1)$ is constructed from a single sorter $ST(m + n)$ whose n highest priority left-side outputs are delivered to the output ports through n registers. From the m lowest priority right-side output, n outputs are recirculated to its right-side inputs through n registers, and the remaining $m - n$ lowest priority cells are discarded. In other words, the PPQ discards lowest priority cells before blocking occurs.

For larger sizes PPQs, each stage operates on $m + n$ inputs and produce $m + n$ outputs in a single step. Note that as soon as the queue gets full, the $m - n$ data cells with lowest priority numbers have to be discarded in every other step. This cell rejection takes place in addition to the discarding process that occurs due to the blocking feature.

6.5 Bit-Serial Realization of the Systolic PPQ

The previous sections focused on the operation of the PPQ under a word-level VLSI model which assumes that a fixed number of elements can be sorted in $O(1)$ -time. Although such circuits are realizable in current VLSI technology, they may require excessive silicon area [60]. Bit-serial sorters can be employed instead to reduce area at the expense of longer sorting time. In this section, we show how the operations of the systolic PPQ can be considered as composite processes which are realized in bit-serial pipelined circuits.

The sorting techniques of this chapter are implemented on a suitable bit-level VLSI model [41], [56], for which the following assumption holds: operations are performed under the bit-serial model of parallel computation and must be completed in one *time-slot*. Under this model it is not possible to store the entire bits representing one data cell in one node, nor is it possible to sort serial-bit data cells in one *time-unit*. However, the buffers at the outputs of each sorter must be large enough to contain entire data cells. In general, any fast bit-serial sorter can be used in our PPQ provided that it can be interfaced properly to the registers of each stage. In the following we provide delay analysis for a PPQ using such circuits.

PPQ Timing and Delay Analysis: As discussed in Chapter 5, the priority number of a data cell is assumed to be $(k + 1)$ -bit tuple $(e, p_1, p_2, \dots, p_k)$, with e being the empty-cell indicator, p_1 the most significant priority bit (MSB), and p_k the least significant priority bit (LSB). Thus, empty data cells (i.e. for which $e=1$) are considered as the lowest priority (largest P value) data cells.

The core element in a bit-serial realization of a sorter is a simple bit-serial *primitive sorter* ST , which operates on two inputs. A primitive sorter, shown in Fig. 6.8, is a comparator device with

two inputs, I_1 and I_2 , and two outputs, O_1 and O_2 , that performs the following functions:

$$O_1 = \text{MIN}(I_1, I_2) \text{ and } O_2 = \text{MAX}(I_1, I_2).$$

Two basic geometries (i.e. pictorial representations) are shown in Fig. 6.8 for the primitive sorter

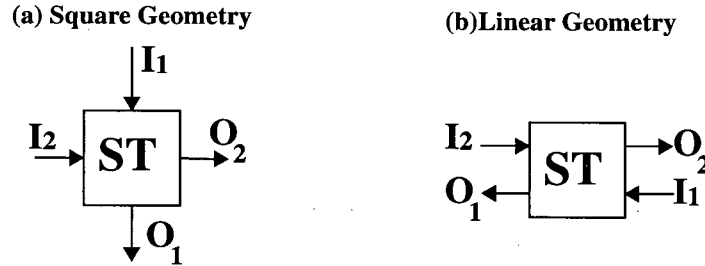


Fig. 6.8: Two pictorial representations of a primitive sorter

ST . The “square” geometry (Fig. 6.8-a) will be used in constructing parallel sorters with larger dimensions, while the “linear” geometry (Fig. 6.8-b) will be employed in constructing priority queues of various depths. Note that the internal structures of the two geometries are identical.

The primitive sorter compares the two data cell serialized in time starting with the most significant bit of the priority number P , and moving to the next less significant bit in the next clock cycle. The realization of such sorter is well known and can be found in many text books [25], [61]. For our analysis it is important to realize that the delay of a primitive sorter is equivalent to that of a single flip-flop (or *primitive delay element* D). Thus, in each input-output path, the combined delay of a primitive sorter with its one-bit register is fixed and is defined as *sorting cycle* of a primitive sorter (i.e. $\Delta(2)$ or simply Δ) which is equal to one clock period or one unit of time.

Under the assumption that each primitive sorter has a unit time sorting cycle, i.e. $\Delta = 1$, we can define $\Delta(2m)$ as the time taken by a $ST(2m)$ sorter to complete a sorting operation on all of its input data cells. Thus, $\Delta(2m)$ is the delay of a $2m$ -cell sorter $ST(2m)$ measured in terms of unit delays (or Δ s). This period is the time which it takes to sort the incoming data cells and make them available at the outputs of the sorter, and is determined by the maximum delay path that an input data cell encounters as it passes through the sorter.

It is important to observe that the correct operation of the proposed systolic PPQ requires that we employ a stage of delay elements with a delay equal to the sorting cycle of $ST(2m)$. Such a delay element will be denoted by $D(2m)$ and its total delay is equal to $\Delta(2m)$. Note that a delay element $D(2m)$ can be realized using $\Delta(2m)$ primitive delay elements D (which itself has delay equal to $\Delta=1$) in series. The function of a primitive delay element D is mainly that of a basic edge-triggered flip-flop where output transitions take place following the rising edge of the clock tick.

6.6 Structure of Parallel Sorters

Chapter 5 presented a realization of a systolic parallel priority queue $PPQ(m, w)$ based on a linear interconnection of $2m$ -cell sorters, and additional registers. The feasibility and efficiency of a PPQ therefore relies crucially on the sorter realization. In this section, we show that equally modular and recursive realizations can be derived for the sorters. We focus mainly on small sorter design, since for most practical realizations we expect the number of inputs to a sorter will be less than 8.

This section presents a method for the recursive construction of small sorters from a VLSI synthesis perspective. The authors are aware of the wide body of literature on the design of sorters [25], [58], [61]. Our main concern here is optimizing the design of small sorters. Although this method may not result in sorters with a minimal number of gates, it has advantage of having a very compact recursive specification of the sorter in terms of the parameter m . This can be advantageous from high-level synthesis (using hardware description languages such as VHDL or Verilog) perspective. In the following, two different methods are proposed for implementing a $2m$ -cell sorter.

6.6.1 Recursive Construction of Parallel Sorters

In the first approach, a $2m$ -cell sorter $ST(2m)$ can be constructed recursively from m -cell sorters $ST(m)$. Note that this assumes that $m = 2^k$, $k = 1, 2, \dots$. For $m = 2$, Fig. 6.9 shows that 5

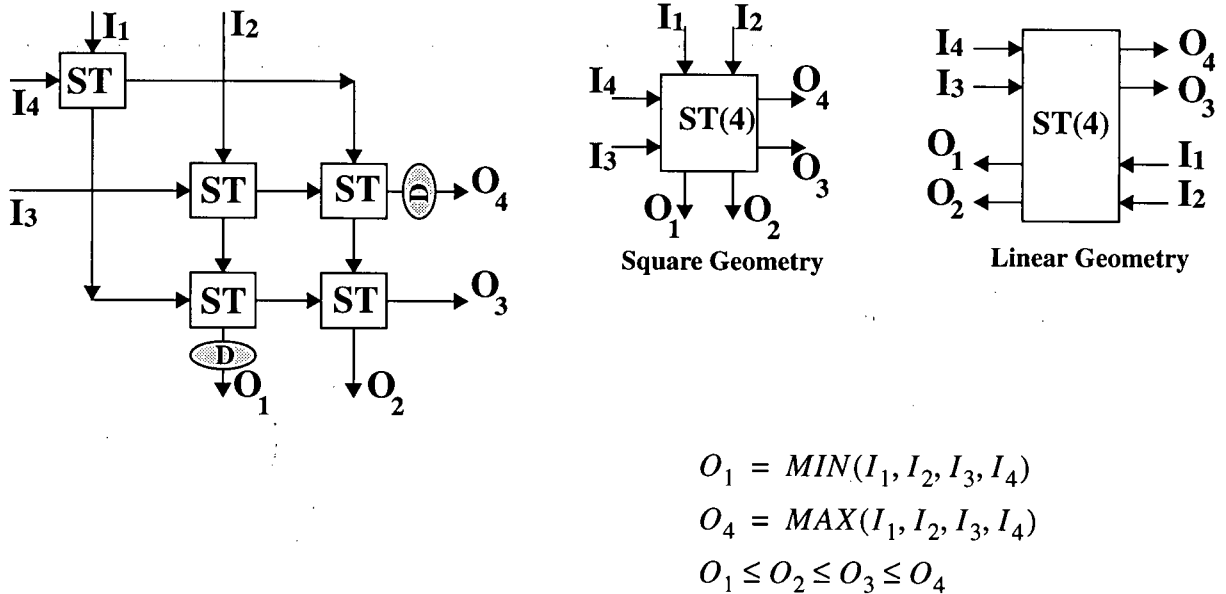


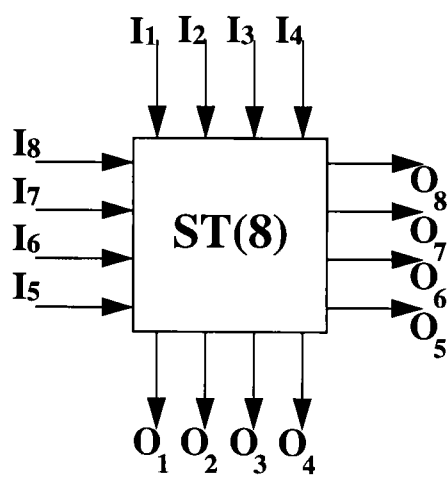
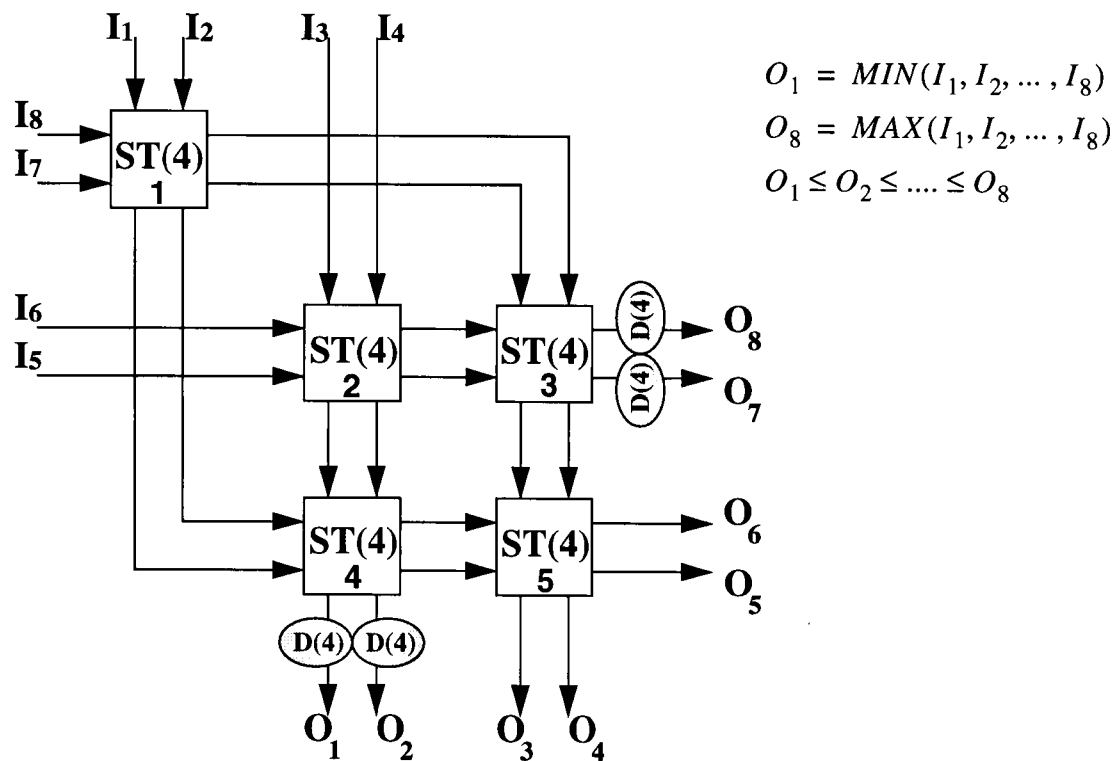
Fig. 6.9: Configuration of $ST(4)$ and its two pictorial representations

primitive sorters ST (i.e. 2-cell sorters) are required to construct a 4-cell sorter $ST(4)$. The equivalent five comparator sorting network is well known and can be found in [25], [61]. Two primitive delay elements D are also employed to assure the proper synchronization of the incoming data cells. Essentially, the delay elements equalize the sorting cycle, or delay, along all input-output paths of the sorter, so that all data cells experience the same delay inside the sorter.

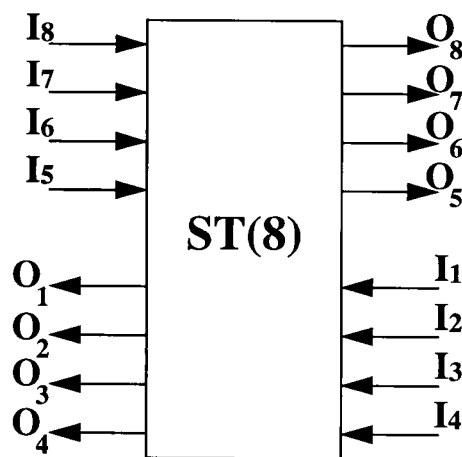
Two basic geometries (i.e. pictorial representations) are also shown in Fig. 6.9 for $ST(4)$. The “square” geometry will be used in constructing larger sorters, while the “linear” geometry will be used in constructing a priority queue $PPQ(2, w)$, where w is an odd integer.

Fig. 6.10 illustrates how an 8-cell sorter $ST(8)$ can be constructed from five 4-cell sorters and four $D(4)$ delay elements. Delay chains in the parallel sorters are represented by shaded ovals. Again, the delay elements guarantee that all incoming data cells in one time-slot leave the sorter in the same time. Two equivalent geometries for the 8-cell sorter $ST(8)$ are also demonstrated in Fig. 6.10.

This procedure can be applied recursively to realize a systolic $2m$ -cell sorter $ST(2m)$ from five m -cell systolic sorters $ST(m)$ and m delay elements $D(m)$, as shown in Fig. 6.11. Let $n_{(2m)}^{ST}$, and $n_{(2m)}^D$ denote, respectively, the number of primitive ST sorters and primitive D delay elements used to construct $ST(2m)$. Also note that $\Delta(2m)$ denotes the delay of $ST(2m)$, as defined in Section 6.5. Then, the following results can be easily derived from the recursive construction of $ST(2m)$:



Square Geometry



Linear Geometry

Fig. 6.10: Realization of an 8-cell sorter

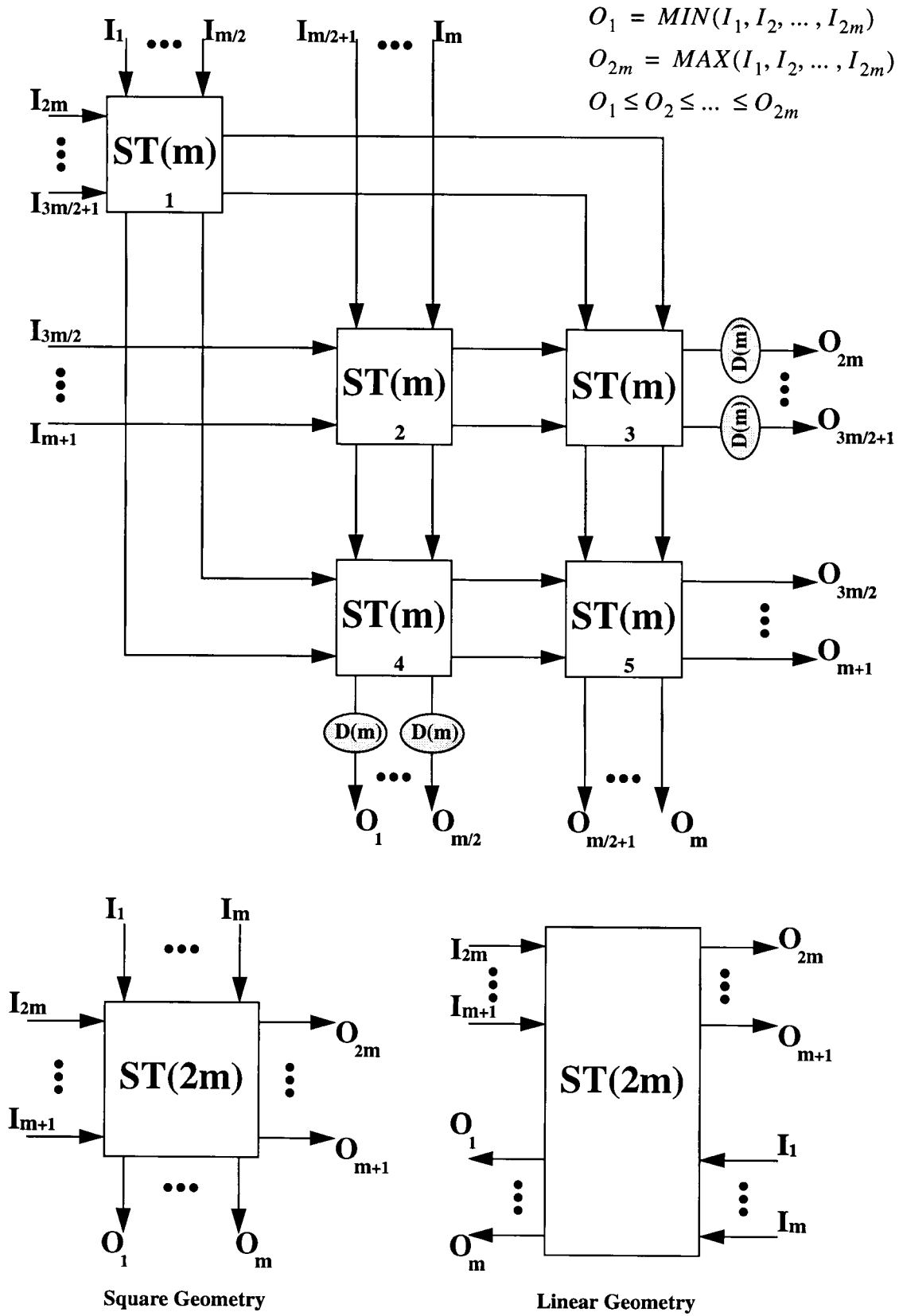


Fig. 6.11: Recursive construction of a $2m$ -cell sorter and its two geometries

$$n_{(2m)}^{ST} = 5^{\log_2 m} = m^{\log_2 5}$$

$$n_{(2m)}^D = 2(6^{\log_2 m} - 5^{\log_2 m}) = 2(m^{\log_2 6} - m^{\log_2 5})$$

$$\Delta(2m) = 3^{\log_2 m} = m^{\log_2 3}$$

It is worth mentioning that in Fig. 6.10 4-cell sorters 3, 4, and 5 (and similarly in Fig. 6.11 m -cell sorters 3, 4, and 5) can be replaced by three *merger* units [25]. Although, this realization might be preferable because the structure of a merger is simpler than that of a parallel sorter (e.g. a 4-cell merger can be implemented using 4 comparators [25]), this mixed implementation would sacrifice the true modularity of the sorter. Also, since PPQs normally employ small sorters, e.g. $m \leq 4$, the hardware saving from employing mergers would be small.

6.6.2 Direct Construction of Parallel Sorters

The recursive construction method described above, although simple to specify, can result in higher complexity implementations, especially as m gets larger. In this section, we explore an alternative construction that reduces both the size and the sorting cycle of the sorter. In this approach, the $2m$ -cell sorter $ST(2m)$ is realized directly using primitive sorters and delay elements. This method can be used to construct any $ST(2m)$ sorter, where $m = 2k$, $k = 1, 2, \dots$, i.e. m is an even integer.

Fig. 6.12 shows the structure of a 6-cell sorter $ST(6)$. The delay elements are used in the sorter to ensure that both inputs arrive at the same time to each primitive sorter and to ensure that all data cells which arrive in one time-slot leave the sorter at the same time. This structure can be

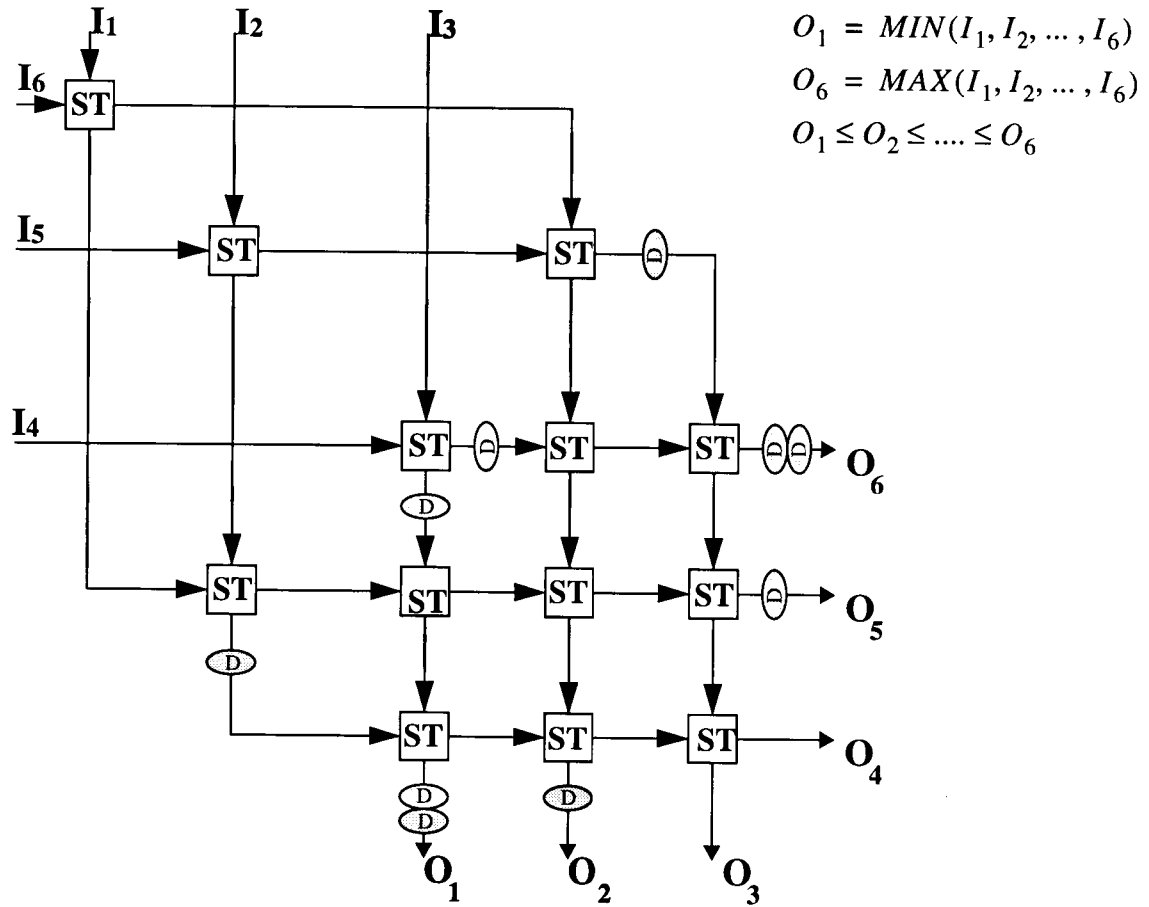


Fig. 6.12: Configuration of a 6-cell sorter using primitive elements

expanded for more input ports as shown in Fig. 6.13 for an 8-cell sorter $ST(8)$, and in Fig. 6.14 for the general case of a $2m$ -cell sorter $ST(2m)$. The systolic sorter of Fig. 6.14 is comprised of a two dimensional systolic array with two triangular sub-arrays and one square array of sorters containing m^2 primitive sorters. Each triangular array also includes $\frac{1}{2}(m-1)(m-2)$ primitive sorters. For this approach, the following bounds can be derived:

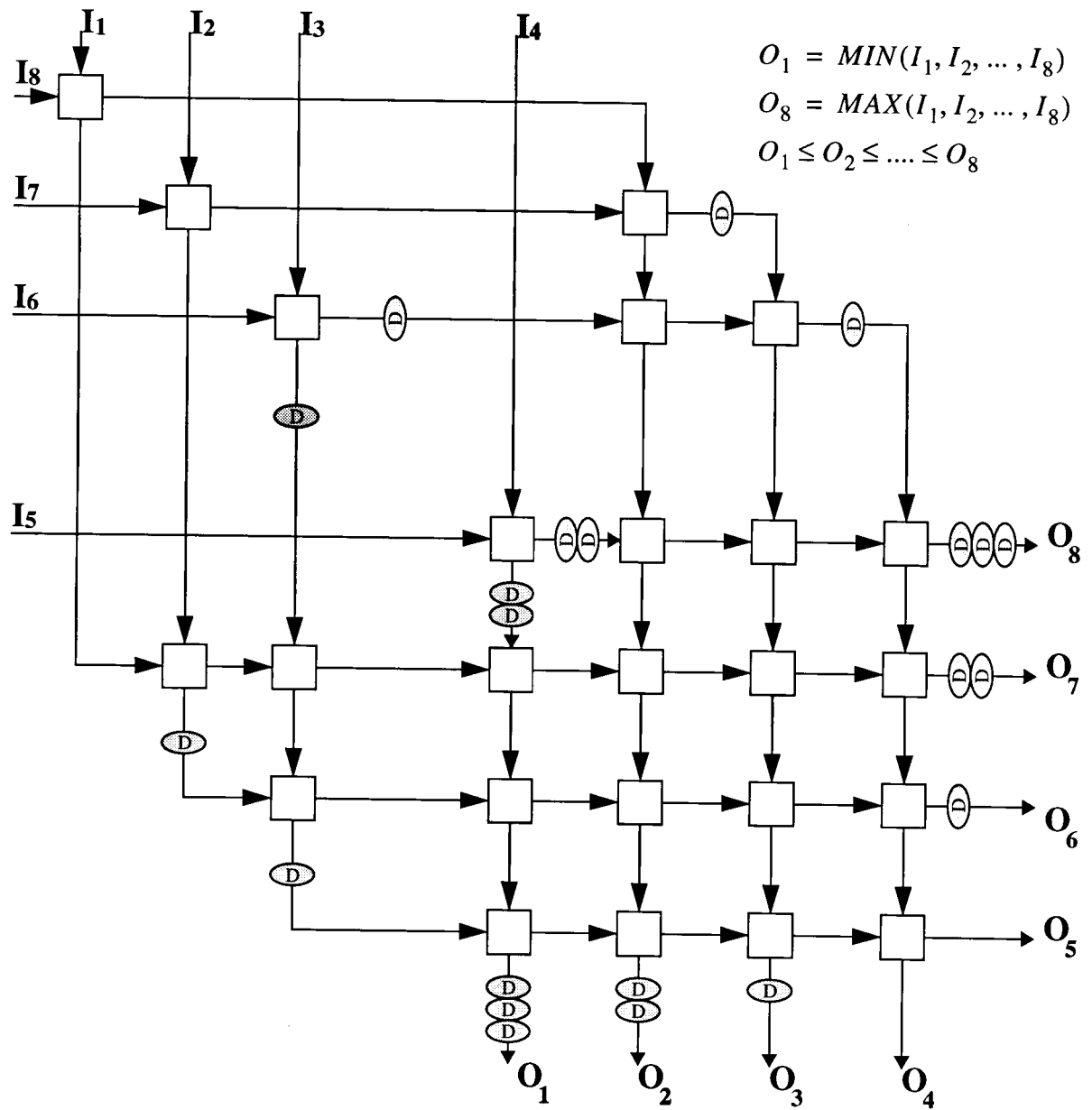


Fig. 6.13: Construction of an 8-cell sorter $ST(8)$ using primitive elements

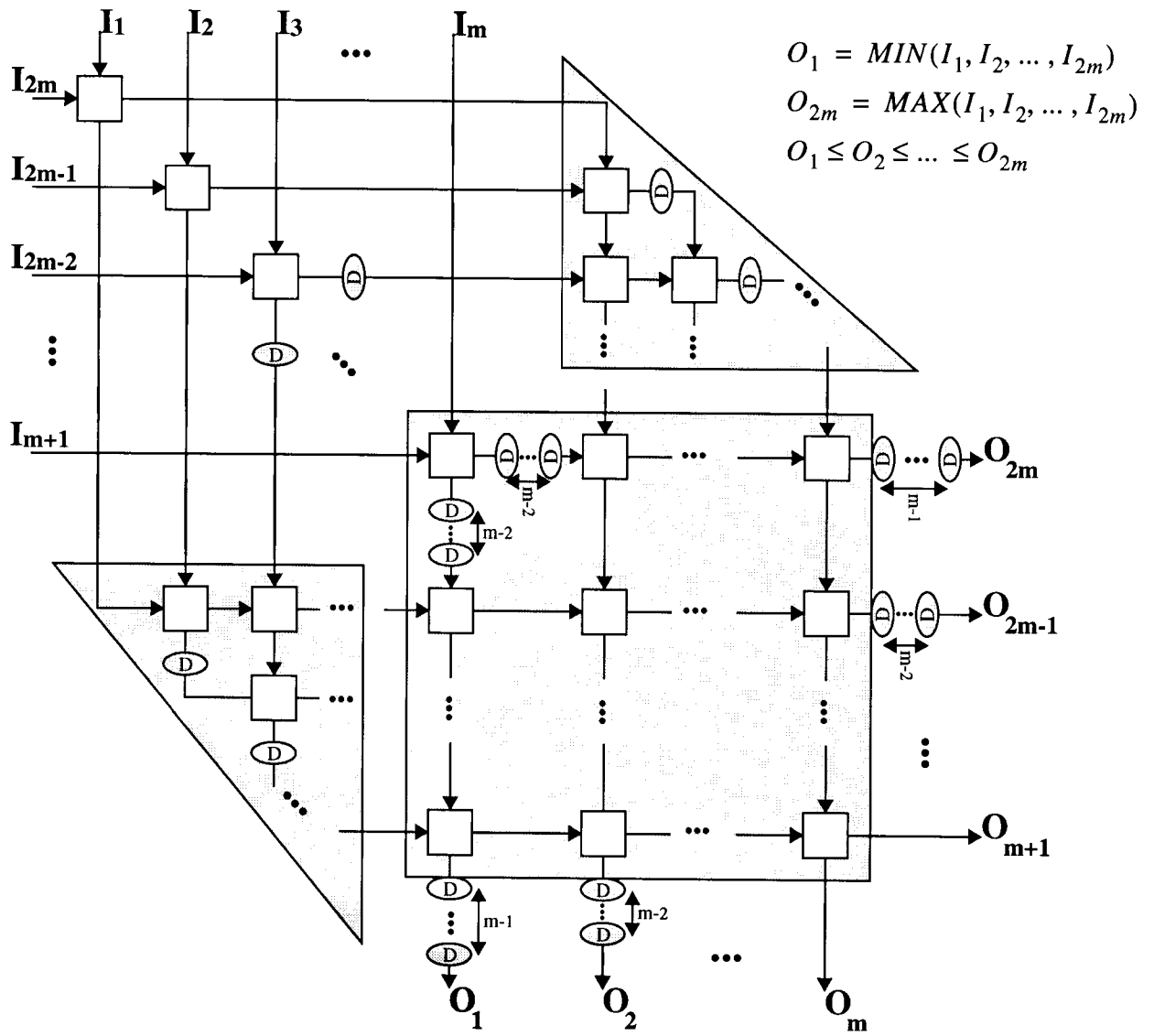


Fig. 6.14: Realization of a $2m$ -cell sorter $ST(2m)$ using primitive elements

$$n_{(2m)}^{ST} = 2m^2 - 2m + 1$$

$$n_{(2m)}^D = 2m^2 - 2m - 2$$

$$\Delta(2m) = 3(m - 1)$$

where $n_{(2m)}^{ST}$, $n_{(2m)}^D$, and $\Delta(2m)$ are as defined before (Section 6.6.1).

Table 6.1 compares the number of elements and the sorting cycle to construct a $2m$ -cell sorter $ST(2m)$ with both approaches for specific values of m .

Table 6.1: Comparing two methods of implementation of a $2m$ -cell sorter

m	Approach 1			Approach 2		
	$n_{(2m)}^{ST}$	$n_{(2m)}^D$	$\Delta(2m)$	$n_{(2m)}^{ST}$	$n_{(2m)}^D$	$\Delta(2m)$
2	5	2	3	5	2	3
4	25	22	9	25	22	9
8	125	182	27	113	110	21
16	625	1342	81	481	478	45

In Section 6.8, implementation issues of primitive sorter and delay elements are presented. But before moving on to further discussion of the implementation issues, it is important to clarify the timing details for a primitive sorter during normal operation in Section 6.7.

6.7 Primitive Sorter Timing

A primitive sorter ST receives data cells from two inputs, and steers these data cells depending on their priority values. The sorter has a set of locally controllable switches which enable the four input/output ports to be connected internally in various configurations. While enable signal is on, depending on data cells' priority values, each primitive sorter ST has two states: *pass (or forward) mode* and *switched (or return) mode*.

The ST and its two possible states are shown in Fig. 6.15-a. Normally, ST is at the pass mode,

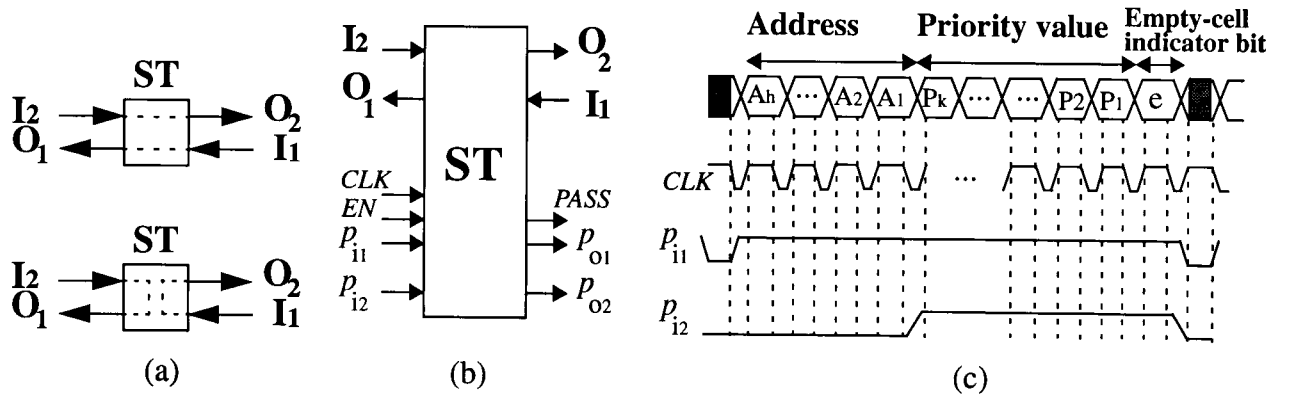


Fig. 6.15: Primitive sorter: a) pass and switched modes, b) More detailed I/O diagram, c) Timing and format of input signals.

i.e. data cells from the left-side are routed to the right-side, and data cells from the right-side propagate to the left-side. While the switching mode is enabled in the sorter, if the priority level of the data cell arriving from the west is higher than that of the data cell arriving from the east, i.e. the P value of I_2 is smaller than that of I_1 , then the ST state is toggled so that the data cell from the

left-side is routed back to the west, and the data cell from the right-side is routed back to the east. In other words, higher priority (smaller P value) data cells always propagate westwards. In the other case, the data cell from the left-side propagates to the east, and that from the right-side propagates to the west.

A more detailed diagram of ST and its data cell input signals format are also shown in Fig. 6.15. In addition to its four ports (I_1 , I_2 , O_1 , and O_2), a SWT has a clock signal (CLK), an *enable* signal (EN), and 5 additional control signals: two input signals p_{i1} , and p_{i2} , and three output signals p_{o1} , p_{o2} , and $PASS$. The p_{i1} signal is used to indicate the beginning of a data cell time-slot, while the p_{i2} signal is used to indicate the timing of incoming data cells priority field (i.e. both empty-cell indicator bit and priority value). $PASS$ is a special signal which shows the mode of ST (pass or switched). The correct operation of a ST assumes that two bit-serial aligned data cells arrive simultaneously at the I_1 and I_2 ports and also synchronized with the p_{i1} and p_{i2} signals.

Signals p_{o1} and p_{o2} are two delay versions of p_{i1} and p_{i2} . As demonstrated in the previous section (see figures 6.9-6.14), data cells are propagated in a general $2m$ -cell sorter $ST(2m)$ similar to a wave propagating along the diagonal direction toward the bottom right corner. The control signals which communicate between adjacent ST s ease the synchronization problem. This requires the same phase to the signal arriving at each ST . For this reason, the p_{i1} and p_{i2} signals are applied from the top left ST s of the $ST(2m)$, and each ST distributes these signals through the p_{o1} and p_{o2} output signals to its east and south neighbors. These signals have to pass through some delay elements in each ST as will be shown later. In order to increase speed and minimize

the silicon area of the primitive sorters, a modified version of domino circuits [38] has been employed. The design strategy and implementation issues of the PPQ will be discussed in the next section.

6.8 VLSI Design Strategy and Implementation Issues

Two different design strategies have been used to implement our systolic PPQ. The first strategy involved developing a synthesizable VHDL model of the queue. Our VHDL model is parametrized in the number of input/output ports of the queue as well as the depth of the queue so that PPQs of different sizes can be synthesized from the same VHDL code. The VHDL model has been also used for timing and functional verification. However, because our main target was to operate the PPQ at the maximum possible clock rate, our second design strategy was focused on transistor-level circuit development for the basic blocks rather than synthesizing the VHDL code.

The first step in our design flow involved full-custom circuit design of the basic blocks using the 0.5- μ m CMOS technology, followed by the same design procedure as described in Section 4.2. Then, using a recursive structural VHDL program which uses the custom design basic blocks as its core elements, a systolic PPQ is realizable. Again, our structural VHDL model is parametrized in the number of input/output ports of the queue as well as the depth of the queue so that PPQs of different sizes can be synthesized from the same VHDL code. Circuit simulation with Hspice assured that the PPQ unit function correctly up to about 330 MHz under a 3.3 V power supply in 0.5- μ m CMOS technology.

The circuit diagram of control part of a *ST* is depicted in Fig. 6.16. The modified domino CMOS circuit technique uses two different clock signals, a (system) *clock* and a *cell clock* (i.e. *CLK*, and

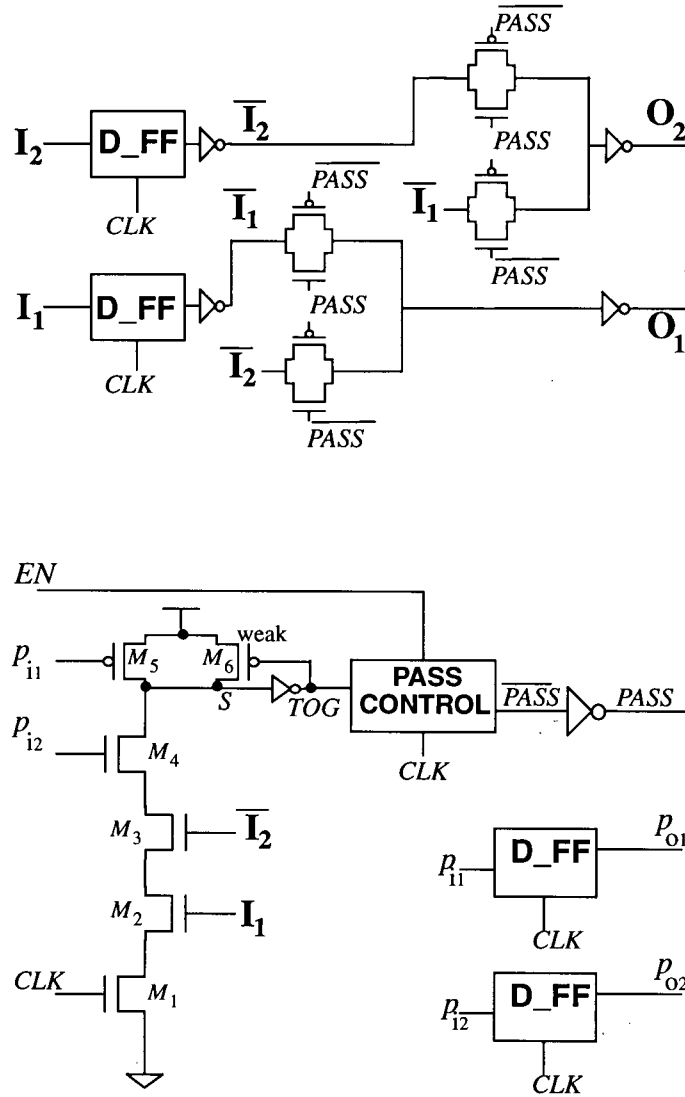


Fig. 6.16: Circuit diagram of the ST

p_{i1}). Similar to the circuit diagram of the switch element in Chapter 4, the use of such dynamic CMOS circuits achieves significant reduction in the number of transistors and, more importantly, a higher switching speed compared to static circuits. The circuit operation is based on first pre-

charging the output node capacitance and subsequently, evaluating the output level according to the applied inputs [38]. A progressive scaling of the transistors in the series chain of our design is beneficial. As discussed in Chapter 4, a graded sizing of nMOS transistors in series structures, where the nMOS transistor closest to the output node has the smallest width-to-length ratio, yields the best performance [40].

The complete signal and data-path of the *ST* is also shown in Fig. 6.16. Data inputs I_1 , and I_2 , and control signals p_{i1} , and p_{i2} are first latched by four D_FF's. The *PASS* signal specifies the direction of data-path in each cell period. The control signals p_{i1} , and p_{i2} are passed as two output signals p_{o1} and p_{o2} to the neighbor *ST*s, to the east and south, after one clock cycle delay.

The data words I_1 , and I_2 are also passed to neighboring *ST*s in the same fashion. Based on a triple-metal/single-poly 0.5- μm CMOS technology, the area of the primitive sorter, which consists of 64 transistors and is depicted in Fig. 6.16, is about $4040 (\mu\text{m})^2$, while the area of the primitive delay element is about $592 (\mu\text{m})^2$.

6.9 Summary

A systolic Parallel Priority Queue (PPQ) for the packet networks was designed and its high-speed operation performance was verified. The proposed PPQ is rate-adaptive in the sense that the PPQ operates correctly even when the queue input rate and output rate are different. This is an important feature because in practice the output rate of the queue is controlled by the available link bandwidth which may vary (or even becomes zero) independent of the packet arrival rate. This decoupling of the input and output packet flow rates is a distinguishing feature of our PPQ concept and has not been addressed in previous literature.

This chapter also described how output rate-adaptation can be also incorporated in the retimed PPQ. Then, the case when the number of parallel inputs to the PPQ is larger than the number of outputs was addressed. Finally, the chapter presented an area-efficient, systolic design of the PPQ for VLSI implementation. In this design, the combined use of a systolic structure, and dynamic CMOS circuits facilitated the balancing of design complexity and performance return. Circuit simulation with Hspice assured that the PPQ unit function correctly up to about 330 MHz under a 3.3 V power supply in 0.5- μ m CMOS technology.

CHAPTER 7

Conclusion & Future Directions

The explosion in data traffic volumes in recent years and the consequent fiber expansion are forcing system designers to rapidly increase packet network capacity. A critical issue in designing of these packet network is the realization of high speed data buffers. The main thrust of this thesis is the development of two scalable queuing structures in VLSI for very high speed packet-switched networks. The main motivation for this work is the emerging need for scalable architectures for high speed interfaces and packet scheduling systems. However, all architectures and circuits presented in the thesis can be employed in other applications (for example see Section 3.5 in Chapter 3).

A methodology for the systematic design of order-preserving multi-input multi-output (MIMO) buffers was presented in Chapters 3 and 4. These buffers are capable of inputting and/or outputting multiple packets while maintaining their FIFO order, and adaptively manage the available buffer space for statistically multiplexed input traffic. Our approach employs a systolic routing network and bank of parallel FIFO buffers to yield a load-balanced buffer realization with increased bandwidth [8]. Using this methodology we derived scalable parallel MIMO FIFO buffer structures that can be designed to match the rate of ultra high-speed links using current memory technology that uses moderate clock rates. Using a 0.5- μ m CMOS technology, a small prototype of the MIMO buffer was designed which attains a rate of 10.6 Gb/s which is more than adequate to support a Sonet OC-192 link [9]. The combined use of pipelined architecture and dynamic CMOS circuits resulted in significant reduction in design complexity and substantial performance gains in speed and silicon area. Although the idea of using a parallel arrangement of FIFO buffers to realize a faster shared FIFO buffer has been around for many years, our systolic routing network provides significant advantages over previously proposed banyan/butterfly networks since the systolic network eliminates the need for parallel-prefix adders that compute packet ranks before concentrating them on the output ports. In our designs, load-balancing is done using arithmetic-free circuits. Our design approach is scalable and amenable to simple VLSI implementation.

In Chapter 5, an area-efficient, systolic design of the parallel priority queue (PPQ) for VLSI implementation was presented [21]. The PPQ maintains prioritized access to the data it contains at all times, and the access time to the data is fixed and independent of the PPQ size, i.e. $O(1)$ -time access. A major challenge is maintaining the correct operation of the priority queue at very high link speeds. The PPQ can be scaled to meet the requirements of ultra high-speed links using

standard CMOS technology. The presented design in Chapter 5 is the generalization of the priority queue concept with steady streams of parallel inputs and outputs, i.e. the same input and output rates.

Chapter 6 developed a novel rate-adaptation feature to allow different input and output rates for the PPQ [22]. This is an important feature because in practice the output rate of the queue is controlled by the available link bandwidth which may vary (or even become zero) independent of the packet arrival rate. In many real applications, it is desirable to decouple the input process to the PPQ from the output process. Specifically, the PPQ outputs may be intermittently blocked from sending cells due to a link congestion. The proposed PPQ in Chapter 6 is rate-adaptive in the sense that the PPQ operates correctly even when the queue input rate and output rate are different. To provide rate-adaptation capability to our PPQ, we strategically employed data steering blocks, between pipeline stages of PPQ, which are controlled by a global “go/stop” signal. This means that the output rate can be controlled by back pressure signals from other stages in the network due to lack of buffer space or due to link congestion. This decoupling of the input and output packet flow rates is a distinguishing feature of our PPQ concept and has not been addressed in previous literature. Chapter 6 also presented an area-efficient, systolic design of the modified PPQ for VLSI implementation, and addressed how the operations of the systolic PPQ with output rate-control can be considered as composite processes which are realized in bit-serial pipelined circuits.

Future Directions

Recently, there has been much interest in using more complex, active queue management algorithms [112] such as multiclass RED (random early detection) [113]-[115] and WFQ (weighted fair queuing) [116]-[118]. To use the proposed PPQ for more sophisticated scheduling algorithms of these queue managements, future improvements are necessary to provide some new features to the architectures. Moreover, the PPQ architectures presented in Chapters 5 and 6 consist of several identical pipeline stages, where each pipeline stage of the PPQ includes mainly some registers and a parallel sorter as its own processing element. One of the immediate improvements is to apply more advanced processing elements instead of parallel sorters for other promising applications [62]. For example, the processing elements can accumulate certain values while sorting packet headers.

It has been reported that although the global clocking schemes simplifies hardware design, the self-timed approach has definite advantages [63], [64] (see Appendix B). Systolic architectures are typically designed under the strict synchronous model of computation. However, for our PPQ architecture it is possible to redesign the sorters as asynchronous processing elements, whereby each computes its output values only when all its inputs are available resulting in a self-timed design [62]. The asynchronous design approach eliminates the need for a global clock and circumvents the problems arising from clock skew. Nevertheless, our systolic PPQ designs are heavily pipelined and retimed; critical paths are properly balanced and little room is left to obtain an asynchronous benefit. Moreover, an asynchronous benefit of this kind must be balanced against a possible overhead in completion signaling and asynchronous control [64]. A self-timed design also has the potential for reduced power consumption relative to its synchronous counterpart. A

synchronous circuit is either quiescent (i.e., the clock is turned off) or active entirely (i.e., clock on). An asynchronous circuit, by contrast, consumes energy only when is active. In other words, self-timed circuits can potentially achieve low power consumption because unused circuits parts can automatically turn into a stand-by mode. However, it is not obvious to what extent this advantage is fundamentally asynchronous. Synchronous techniques such as clock gating may achieve similar benefits, although they have their limitations. In general, the potential of asynchronous design for low-power depends on the application [64]. Additional savings in power can be accomplished by using self-timed circuits with a mechanism that adaptively adjusts the supply voltage to the smallest possible while maintaining the performance requirements [65]. An important consideration that may limit the applicability of this technique is that the supply voltage must vary at a slow rate relative to the internal operational speed of the circuit, otherwise it may interfere with the operation of the circuit. In summary, the issue of clocking in our designs and possible self-timed versions of the PPQ architectures need further investigation [62].

BIBLIOGRAPHY

- [1] S. E. Butner, and R. Chivukula, "On the Limits of Electronic ATM Switching," IEEE Network, vol. 10, no. 6, pp. 26-31, Nov./Dec. 1996.
- [2] S. E. Butner, and D. A. Skirmont, "Architecture and Design of a 40 Gigabit per second ATM Switch," in Proc. Int'l. Conf. Comp. Design, Austin, TX, pp. 352-357, Oct. 1995.
- [3] Richard Crisp, "Direct Rambus Technology: The New Main Memory Standard," IEEE Micro, vol. 17, no. 6, pp. 18-28, Nov./Dec. 1997.
- [4] A. Demers, S. Keshav, and S. Shenker, "Design and Analysis of a Fair Queuing Algorithm," ACM Sigcomm '89, Austin, Sep. 1989.
- [5] A. K. Parekh, "Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," Technical Report LIDS-TH-2089, MIT, Cambridge, MA 02139, Feb. 1992.

- [6] D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", ACM Sigcomm '92, pp. 14-26, Aug. 1992.
- [7] S. Floyd, and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," IEEE/ACM Trans. on Networking, vol. 3 no. 4, pp. 365-386, Aug. 1995.
- [8] M. Kazemi-Nia, and H. Alnuweiri, "VLSI Design of Parallel FIFO Buffers for Gigabit Packet Networks," accepted for publication in the IEEE Trans. on VLSI Systems.
- [9] M. Kazemi-Nia, and H. Alnuweiri, "Parallel Buffer Design for High Speed Interfaces," in proc. CCBR '99, pp. 102-113, Ottawa, Nov. 1999.
- [10] Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora, "The knockout switch: A simple architecture for high-performance packet switching," IEEE J. on Select. Areas in Commun., vol. 5, no. 8, pp. 1274-1283, Oct. 1987.
- [11] H. S. Kim, "Design and Performance of Multinet Switch: A Multistage ATM Switch Architecture with Partially Shared Buffers," IEEE/ACM Trans. on Networking, vol. 2, no. 6, pp. 571-580, Dec. 1994.
- [12] H. S. Kim, "Multinet Switch: Multistage ATM Switch Architecture with Partially Shared Buffers," Infocom '93, pp. 4c.3.1 - 4c.3.8, 1993.
- [13] K. Eng, M. J. Karol, and Y. Yeh, "A Growable Packet (ATM) switch Architecture Design Principles and Applications," IEEE Trans. on Communications, vol. 40, no. 2, pp. 423-430, Feb. 1992.
- [14] K. Y. Eng, M. J. Karol, and Y. Yeh, "A High-Performance Prototype 2.5 Gb/s ATM Switch For Broadband Applications," Globecom' 92, pp. 111-117, 1992.
- [15] K. Y. Eng, M. J. Karol, and Y. Yeh, "Concentrator-based growable packet switch," U. S. Patent 5-256-958, Oct. 26, 1993.
- [16] H.J Chao, and B. Choe, "Design and Analysis of a Large-Scale Multicast Output Buffered ATM Switch," IEEE/ACM Trans. on Networking, vol. 3, no. 2, pp. 126-138, April 1995.
- [17] H. J. Chao, "A Recursive Modular Terabit/Second ATM Switch," IEEE J. on Selec. Areas in Commun., vol. 9, no. 8, pp. 1161-1172, Oct. 1991.
- [18] K. L. E. Law, and A. Leon-Garcia, "A Large Scalable ATM Multicast Switch," IEEE J. on Selec. Areas in Commun., vol. 15, no. 5, pp. 844-854 June 1997.

- [19] I. Widjaja, and A. Leon-Garcia, "The Helical Switch: A Multipath ATM Switch Which Preserves Cell Sequence," *IEEE Trans. on Communications*, vol. 42, no. 8, pp. 2618-2629, Aug. 1994.
- [20] A. Pattavina, "Nonblocking Architectures for ATM Switching," *IEEE Communication Magazine*, vol. 31, no. 2, pp. 38-48, Feb. 1993.
- [21] M. Kazemi-Nia, and H. Alnuweiri, "A Parallel Priority Queue (PPQ) with Rate Adaptation for High Speed Networks," in *proc. CCBP '99*, pp. 90-101, Ottawa, Nov. 1999.
- [22] M. Kazemi-Nia, and H. Alnuweiri, "A Systolic Parallel Priority Queue (PPQ) with Output Rate-Control for High Speed Networks," submitted for publication in the *IEEE Trans. on VLSI Systems*, June 1999.
- [23] H. J. Chao, and N. Uzun, "A VLSI Sequencer Chip for ATM Traffic Shaper and Queue Manager," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1634-1643, Nov. 1992.
- [24] S.-W. Moon, J. Rexford, and K. Shin, "Scalable hardware priority queue architectures for high-speed packet switches," *IEEE Real-Time Tech. and Applic. Symp.*, pp. 203-212, June 1997.
- [25] F. T. Leighton, *Introductions to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*, San Mateo, CA: Morgan Kaufmann, 1992.
- [26] M. Kazemi-Nia, and H. Alnuweiri, "Balanced Multiport Buffer Design in Silicon," *IEEE ATM '96 Workshop*, pp. HW.1-6, San Francisco, CA, Aug. 1996.
- [27] M. Kazemi-Nia, and H. Alnuweiri, "VLSI Implementation of Highly-Optimized Scalable ATM Switches," *18th Symp. on Communications*, pp. 101-104, Kingston, Canada, June 1996.
- [28] H. T. Kung, "Why systolic architectures," *Computer*, vol. 15, no. 1, pp. 37-46, Jan. 1982.
- [29] C. E. Leiserson, *Area-Efficient VLSI Computation*, Ph.D. dissertation, Dept. Computer Science, Carnegie-Mellon University, 1981; published in a book form, Cambridge, Massachusetts: MIT Press, 1983.
- [30] C. E. Leiserson, "Systolic Priority Queues," *Caltech. Conf. on VLSI*, pp. 200-214, Jan. 1979.
- [31] M. Hashemi and A. Leon-Garcia, "A General Purpose Cell Sequencer/Scheduler for ATM Switches", *IEEE Infocom '97*, Kobe, Japan, April 1997.

- [32] H.J Chao, "A Novel Architecture for Queue Management in the ATM Networks," IEEE J. on Selec. Areas in Commun., vol. 9, no. 7, pp. 1110-1118, Sep. 1991.
- [33] H.J Chao, and N. Uzun, "A VLSI Sequencer Chip for ATM Traffic Shaper and Queue Manager," in Proc. Globecom '92, pp. 1276-1281, 1992.
- [34] H. J. Chao, and N. Uzun, "A VLSI Sequencer Chip for ATM Traffic Shaper and Queue Manager," IEEE Journal of Solid-State Circuits, vol. 27, no. 11, pp. 1634-1643, Nov. 1992.
- [35] H.J Chao, and N. Uzun, "An ATM Queue Manager Handling Multiple Delay and Loss Priorities," IEEE/ACM Trans. on Networking, vol. 3, no. 6, pp. 652-659, Dec. 1995.
- [36] G.C. Heinze, R. palmer, I. dresser, and N. Leister, "A three chip set for ATM switching," in IEEE Custom Integr. Circ. Conf., pp. 14.3.1-14.3.4, May 1992.
- [37] D. Picker, and R.D. Fellman, "A VLSI Priority Packet Queue with Inheritance and Overwrite," IEEE Trans. on VLSI systems, vol. 3, no. 2, pp. 245-253, June 1995.
- [38] N. Weste, and K. Eshraghian, Principles of CMOS VLSI Design - a Systems Perspective, Addison-Wesley, 2 edition, 1993.
- [39] J.A. Pretorius, A.S. Shubat, and C.A. Salama, "Charge redistribution and noise margins in domino CMOS logic," IEEE Trans. Circ. Syst., vol. 33, no. 8, pp.786-793, Aug. 1986.
- [40] M. Shoji, "FET Scaling in Domino CMOS Gates," IEEE J. of Solid-State Circuits, vol. sc-20, no. 5, pp. 1067-1071, Oct. 1985.
- [41] J. D. Ullman, Computational Aspects of VLSI, Rockville, MD: Computer science press, 1984.
- [42] H.M. Alnuweiri, and R. Beraldi, "A General Class of Highly-Optimized Scalable ATM Switches," Globecom '95, Singapore, Nov. 1995.
- [43] W. Marcus, "A CMOS Batchter and Banyan Chip Set for B-ISDN Packet Switching," IEEE J. of Solid-State Circuits, vol. 25, no. 6, pp. 1426-1432, Dec. 1990.
- [44] M. D. Marco, " Distributed Routing Protocols for ATM Extended Banyan Networks," IEEE j. on Selec. Areas in Commun., vol. 15, no. 5, June 1997.
- [45] P. C. Wong, and M. S. Yeung, "Design and Analysis of a Novel Fast Packet Switct -- Pipeline Banyan, IEEE/ACM Trans. on Networking, vol. 3, no. 1, pp. 63-69, Feb. 1995.

- [46] F. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks," *Proc. of the IEEE*, vol. 78, no. 1, pp. 133-167, Jan. 1990.
- [47] J. S. Turner, "Design of a Broadcast Packet Switching Network," *IEEE Trans. Commun.*, vol. 36, no. 6, pp. 734-743, June 1988.
- [48] M. Alimuddin, H. M. Alnuweiri, and R. W. Donaldson, "The Fat-Banyan ATM Switch," *IEEE Infocom '95*, pp. 659-666, April 1995.
- [49] E. T. Bushnell, and J. S. Meditch, "Dilated Multistage Interconnection Networks for Fast Packet Switching," *IEEE Infocom '91*, pp. 1264-1273, 1991.
- [50] C. P. Kruskal, and M. Snir, "The performance of Multistage Interconnection Networks for Multiprocessors," *IEEE Trans. on Computers*, vol. c-32, no. 12, pp. 1091-1098, Dec. 1983.
- [51] T. Szymanski, and S. Shaikh, "Markov Chain Analysis of Packet-Switched Banyans with Arbitrary Switch Sizes, Queue Sizes, Link Multiplicities, and Speedups," *IEEE Infocom '89*, pp. 960-971, 1989.
- [52] Y. Mun, and H. Y. Youn, "Performance Analysis of Finite Buffered Multistage Interconnection Networks," *IEEE Trans. on Computers*, vol. 43, no. 2, pp. 153-162, Feb. 1994.
- [53] H. Yoon, K. Y. Lee, and M. T. Liu, "Performance Analysis of Multibuffered Packet-Switching Networks in Multiprocessor Systems," *IEEE Trans. on Computers*, vol. 39, no. 3, pp. 319-327, March 1990.
- [54] C. M. Chu, H. Tayar, and H. M. Alnuweiri, "Enhanced Packet Switching on a Dilated Banyan Switch with Back Pressure," in *Proc. ISCA '99*, pp. 130-134, Mexico, 1999.
- [55] *IEEE Standard VHDL Language Reference Manual*, std 1076-1993, New York: IEEE, 1993.
- [56] C. D. Thompson, "Area-time complexity for VLSI," *Proc. ACM Symp. Theory Computation*, May 1979.
- [57] D. E. Knuth, *The Art of Computer Programming. Vol. III: Sorting and searching*, Reading, MA: Addison-Wesley, 1973.
- [58] H. M. Alnuweiri, "A New Class of Optimal Bounded Degree VLSI Sorting Networks", *IEEE Trans. on Computers*, vol. 42, no. 6, pp. 746-752, June 1993.
- [59] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introductions to Algorithms*, Cambridge, MA: The MIT Press, 1990.

- [60] C. D. Thompson, "The VLSI complexity of sorting," IEEE Trans. on Computers, vol. C-32, no. 12, pp. 1171-1184, Dec. 1983.
- [61] D. E. Knuth, The Art of Computer Programming. Vol. III: Sorting and searching, Reading, MA: Addison-Wesley, 1973.
- [62] M. Kazemi-Nia, and H. Alnuweiri, "A Self-timed Parallel Priority Queue (PPQ) with Output Rate-Control," in preparation.
- [63] S. Hauck, "Asynchronous Design Methodologies: An Overview," Proc. of the IEEE, vol. 83, no. 1, pp. 69-93, Jan. 1995.
- [64] C. H. Van Berkel, M. B. Josephs, and S. Nowick, "Applications of Asynchronous Circuits," Proc. of IEEE, vol. 87, no. 2, pp. 223-233, Feb. 1999.
- [65] L. Nielsen, C. Niessen, J. Sparso, and K.V. Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," IEEE Trans. on VLSI Systems, vol. 2, no. 4, pp. 391-397, Dec. 1994.
- [66] Craig Partridge, Gigabit networking, Addison-Wesley Pub., 1991.
- [67] T. Koinuma, and N. Miyaho, "ATM in B-ISDN Communication Systems and VLSI realization." IEEE J. of Solid-State Circuits, vol.30, no. 4, pp. 341-347, April 1995.
- [68] M. Hlucchyj, and M. Karol, "Queueing in high-performance packet switching," IEEE J. on Select. Areas in Commun., vol. 6, no. 9, pp. 1587-1597, Dec. 1988.
- [69] H. Kuwahara et al., "A shared buffer memory switch for an ATM exchange," in Proc. ICC '89, Boston, MA, June 1989, pp. 118-122.
- [70] T. Kozaki, N. Endo, Y. Sakurai, O. Matsubara, M. Mizukami, and K. Asano, "32 x 32 Shared Buffer Type ATM Switch VLSI's for B-ISDN's," IEEE J. on Selec. Areas in Commun., vol. 9, no. 8, pp. 1239-1247, Oct. 1991.
- [71] M. Katevenis, P. Vatsolaki, and A. Efthymiou, "Piplelined Memory Shared Buffer for VLSI Switches," Proc. of Sigcomm '95, Cambridge, MA, pp. 39- 48, 1995.
- [72] Notani et al., "An 8x8 ATM Switch LSI with Shared Multi-buffer Architecture," Proc. of Sym. on VLSI Circuits Digest of Technical Papers, pp. 74-75, 1992.
- [73] C. Zukowski, and T.B. Pei, "VLSI implementation of ATM buffer management," Proc. of IEEE Int. Conference on Communications, pp. 716-720, 1991.
- [74] M. Katevenis, P. Vatsolaki, and A. Efthymiou, "Piplelined Memory Shared Buffer for VLSI Switches," Proc. of Sigcomm '95, Cambridge, MA, pp. 39- 48, 1995.

- [75] G. Kornaros, C. Kozyrakis, P. Vatsolaki, and M. Katevenis, "Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow Control," in Proc. 17th of ARVLSI' 97, Ann Arbor, MI, USA, pp. 127-144, Sept. 1997.
- [76] R. O. Onuvural, Asynchronous Transfer Mode Networks: Performance Issues, Artech House, 1994.
- [77] C. Fayet, A. Jazques, and G. Pujolla, "High speed switching for ATM: the BSS," Computer Networks and ISDN Systems, pages 1225-1233, 1994.
- [78] M.J. Karol, M.G. Hlucchyj, and S.P. Morgan, "Input versus output queueing on a space-division packet switch," IEEE Trans. Commun., vol. 35, no. 12, pp. 1347-1356, Dec. 1987.
- [79] Y. Watanabe, Y. Nakasha, Y. Kato, K. Odani, and M. Abe, "A 9.6-Gb/s HEMT ATM Switch LSI with Event-Controlled FIFO," IEEE J. Solid-State Circuits, vol. 28, no. 9, pp. 935-940, Sep. 1993.
- [80] M.J. Karol, M.G. Hlucchyj, and S.P. Morgan, "Input versus output queueing on a space-division packet switch," IEEE Trans. Commun., vol. 35, no. 12, pp. 1347-1356, Dec. 1987.
- [81] H. Shi, and O. Wing, "A Novel Design Approach of ATM Switches for VLSI Implementations," Proc. of Elec. & Comp. Eng. Canadian Conference, pp. 684-687, 1993.
- [82] Y. Watanabe, Y. Nakasha, Y. Kato, K. Odani, and M. Abe, "A 9.6-Gb/s HEMT ATM Switch LSI with Event-Controlled FIFO," IEEE Journal of Solid-State Circuits, vol. 28, no. 9, pp. 935-940, Sep. 1993.
- [83] S. Li, "Theory of periodic contention and its application to packet switches," Infocom '88, pp. 320-325, 1988.
- [84] N. McKeown, T. E. Anderson, "A Quantitative Comparison of Scheduling Algorithms for Input-Queued Switches," Infocom '96.
- [85] C.L. Seitz, System Timing. In Introduction to VLSI Systems, C.A. Mead, and L.A. Conway, Eds., Addison-Wesley, 1980.
- [86] E. Brunvand, N. Michell, and K. Smith, "A Comparison of Self-timed Design using FPGA, CMOS, and GaAs Technologies," Proc. of IEEE Intl. Conf. on Computer Design, pp. 76-80, 1992.
- [87] E. Brunvand, "The Non-Synchronous Rise (NSR) Processor," Proc. Annual Intl. Conf. on System Sciences, Hawaii, pp. 428-435, 1993.

- [88] J. Novak, and E. Brunvand, "Using FPGAs to Prototype a Self-Timed Floating Point Co-Processor," Proc. of IEEE Custom Integrated Circuits Conf., pp. 85-88, 1994.
- [89] Tierno et. al., "A 100-MIPS GaAs Asynchronous Microprocessor," IEEE Design and Test of Computers, vol. 11, no. 2, pp. 43-49, Summer 1994.
- [90] G. M. Jacobs, and R. W. Brodersen, "Self-Timed Integrated Circuits for Digital Signal Processing Applications," IEEE VLSI Signal Processing III, pp. 197-208., 1989.
- [91] G.M. Jacobs, and R.W. Brodersen, "A Fully Asynchronous Digital Signal Processor Using Self-Timed Circuits," IEEE Journal of Solid-State Circuits, vol. 25, no. 6, pp. 1526-1536, Dec. 1990.
- [92] C. H. Lau, D. Renshaw, and J. Mavor, "Data Flow Approach to Self-Timed Logic in VLSI," Proc. of IEEE Intl. Symp. on Circuits and Systems, pp. 479-482, 1988.
- [93] V. Akella, and G. Gopalakrishnan, "SHILPA: A High-level Synthesis System for Self-Timed circuits," Proc. Intl. Conf. Computer-Aided Design, pp. 587-591, 1992.
- [94] D. Lloyd, and S. Jones, "Improved self-timed circuit design method," Elec. Letters, vol. 28, no. 5, pp. 492-493, 27th Feb. 1992.
- [95] S.L. Lu, "Design of hardware efficient self-timed circuits," Elec. letters, vol. 29, no. 1, pp. 6-7, 7th Jan. 1993.
- [96] N. R. Poole, "Self-timed logic circuits," IEEE Elec. & Commu. Eng. Journal, vol. 6, no. 6, pp. 261-270, Dec. 1994.
- [97] Komori et. al., "The Data-Driven Microprocessor," IEEE Micro, vol. 9, no. 3, pp. 45-59, June 1989.
- [98] Yamasaki et. al., "VLSI Implementation of a Variable-Length Pipeline Scheme for Data-Driven Processors," IEEE Journal of Solid-State Circuits, vol. 24, no. 4, pp. 933-937, Aug. 1989.
- [99] I.E. Sutherland, "Micropipelines," Commun. Assoc. Comput. Mach., vol. 32, no. 6, pp. 720-738, June 1989.
- [100] S.L. Lu, "Implementation of Micropipelines in Enable/Disable CMOS Differential Logic," IEEE Trans. on VLSI, vol. 3, no. 2, pp. 338-341, June 1995.
- [101] Spaso, J. Staunstrup, and M. D. Sorensen, "Design of delay insensitive circuits using multi-ring structures," Proc. European Design Automation Conf., Hamburg, pp. 15-20, 1992.

- [102] J. Spaso, and J. Staunstrup, "Design and Performance Analysis of Delay Insensitive Multi-Ring Structures," Proc. Annual Intl. Conf. on System Sciences, Hawaii, pp. 349-358, 1993.
- [103] T. H. Meng, Synchronization Design for Digital Systems, Kluwer Academic publishers, 1991.
- [104] L. G. Heller, and W. R. Griffin, "Cascode voltage switch logic: A differential CMOS logic family," in ISSCC Dig. Tech. Papers, pp. 16-17, New York, Feb. 1984.
- [105] T. Wu, and S. Vrudhula, "A Design of a Fast and Area Efficient Multi-Input Muller C-element," IEEE Trans. on VLSI, vol. 1, no. 2, pp. 215-219, June 1993.
- [106] S.L. Lu, "Improved design of CMOS multiple-input Muller C-element," Elec. letters, vol. 29, no. 19, pp. 1680-1682, 16th Sep. 1993.
- [107] M. Shams, J. Ebergen, and M. Elmasry, "Modeling and Comparing Implementations of the C-Element," IEEE Trans. on VLSI, vol. 6., no. 4, Dec. 1998.
- [108] K.V. Berkel, "Beware the isochronic fork," The VLSI Journal "Integration", no. 13, pp. 103-128, 1992.
- [109] I. M. Filansky, and H. Baltes, "CMOS Schmitt Trigger Design," IEEE Trans. on Circuits and Systems--I: Fundamental Theory and Applications, vol. 41, no.1, pp. 46-49, Jan. 1994.
- [110] D. L. Jackson, R. Kelly, and L. E. M. Brackenbury, "Differential register bank design for self-timed differential bipolar technology," IEE Proc.-Circuits Devices Syst., vol. 144, no. 5, Oct. 1997.
- [111] R. Kelly, and L. E. M. Brackenbury, "Design and modeling of a high performance differential bipolar self-timed microprocessor," IEE Proc. -Comput. Digit. Tech., Vol. 144, No. 6, November 1997.
- [112] B. Braden, D. Clark, J. Crowcroft, B. Davie, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," Internet Draft, March 1997.
- [113] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Trans. on Networking, vol. 1, pp. 397-413, Aug. 1993.
- [114] S. Floyd, and V. Jacobson, "On Traffic Phase Effects in Packet Switched Gateways," Internetworking: Research and Experience, vol. 3, pp. 397-413, Aug. 1993.

- [115] D. Lin, and R. Morris, "Dynamics of Random Early Detection," in Proc. ACM Sigcomm '97.
- [116] A. K. Parekh, and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control -- The Single Node Case," in Proc. Infocom '92, vol. 2, pp. 915-924, May 1992.
- [117] A. K. Parekh, and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," in Proc. Infocom '93, vol. 2, pp. 521-530, 1993.
- [118] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," Internetworking: Research and Experience, vol. 1, no. 1, pp. 3-26, 1990.
- [119] C. Aras, J.F. Kurose, D.S. Reeves, and H. Schulzrinne, "Real-Time Communication in Packet-Switched Networks," Proc. of IEEE, vol. 82, no. 1, pp. 122-138, Jan. 94.
- [120] H. J. Chao, and N. Uzun, "An ATM Routing and Concentration Chip for a Scalable Multicast ATM Switch," IEEE Journal of Solid-State Circuits, vol. 32, no. 6, pp. 816-828, June 1997.
- [121] H. J. Siegel, Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, McGraw-Hill, second edition 1990.

APPENDIX A

ATM Technology

Asynchronous transfer mode, or ATM, has been standardized by ITU-T (International Telecommunication Union - Telecommunication Standardization Sector, formerly known as CCITT) as the transfer mode for the future multimedia communication systems because of its ability to address the broadband integrated services digital network (B-ISDN) requirements [46]. It allows a single architecture to efficiently support connections carrying traffic at widely different bandwidths, and to handle bursty traffic as expected in some multimedia applications. The ATM technique provides a universal interface, using short fixed-length packets, called *cells*, for all types of media transmission (data, voice, and video). Cells belonging to many different connections can interleave in a cell stream transported over a physical link. It is worth noting that despite its name,

ATM, uses a synchronous physical layer, such as SONET and SDH¹ [46]. In other words, the bit stream in the network is synchronous. The word *asynchronous* refers to the fact that the cell stream allocation for services is not synchronous and channels are signified in the ATM cell header.

ATM systems for B-ISDN should handle a wide range of quality of service (QoS) parameters such as cell delay, cell loss rate, and cells delay-jitter in a multimedia environment [119], because the characteristics of real-time communication applications differ significantly from those that are non real-time. Typically, the desired delivery time for each message across the network is determined by a specific maximum delay or latency, resulting in a deadline being associated with each message. On the other hand, non-real-time applications may have strict cell loss-rate requirements. For example, real-time applications such as voice and interactive video have strict end-to-end delay requirements, and signals must be delivered within a specified period of time, while data transfers such as remote file access require low loss-rates and must be delivered reliably. Another important performance metric for real-time traffic is delay-jitter, commonly defined as the maximum variation in the delay experienced by cells in a single connection. Many real-time applications, particularly those which are interactive, require a bound on delay-jitter, in addition to a bound on the delay. Note that certain real-time applications such as non-interactive television and audio broadcasting may require bounds on delay-jitter but not delay.

1. Synchronous Optical Network, SONET, has been chosen as the standard for the future broadband data communication systems in the North America by the American National Standards Institute (ANSI). In Europe, ITU-T has adopted synchronous digital hierarchy, SDH, as a standard compatible with SONET.

A.1 ATM Networking and Switching

Before the introduction of ATM, two different types of public switching networks, circuit switching and packet switching, have been developed for different types of communication services [66]. Circuit switching networks are used for applications such as telephony in which continuous low rate signal channels with very low delay are needed, while packet switching networks are used for computer data transfers which may require high speed data transfers in a short period of time and the application is not sensitive to the data transfer delay. In ATM networks the benefits of those two networks are combined, and the information which includes voice, data, and motion video is divided into fixed length packets (called *cells*) which are asynchronously multiplexed and transmitted over networks. An ATM cell is composed of 48 bytes of data and 5 bytes header, as shown in Fig. A.1. The single most important ATM standard concerns the definition of the cell

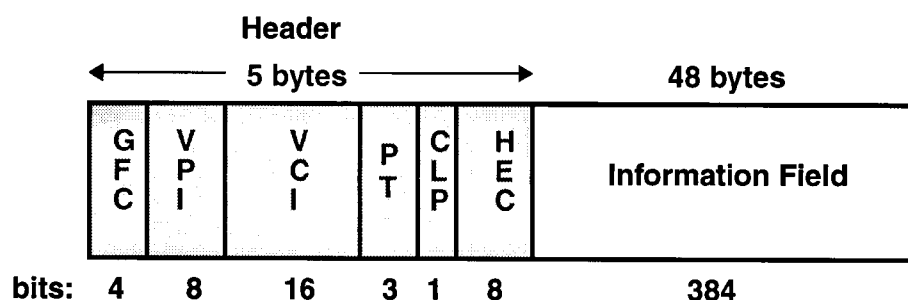


Fig. A.1: ATM cell format and header components

header. The header contains information about the routing and the cell content. The primary use of the header tag is to identify cells belonging to the same virtual channel and to make routing possible.

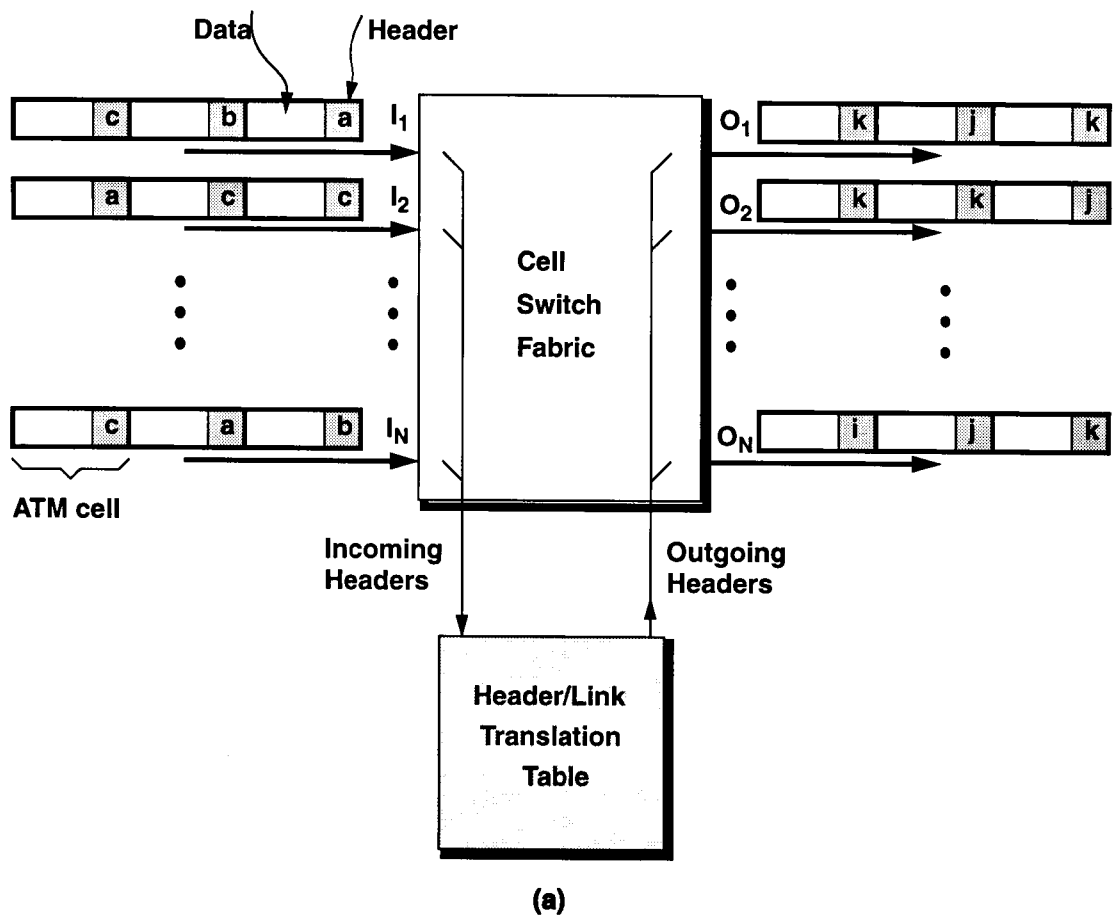
The connection identifier consists of two sub-fields, a 16-bit virtual channel identifier (VCI) and

an 8-bit virtual path identifier (VPI). Together they are used to route the cell to its destination. The main purpose of the PT field is to distinguish between cells containing user data and network information. The cell loss priority (CLP) bit allows a two-level loss priority (CLP = 0 for high, CLP = 1 for low) to be specified explicitly for individual cells. Low loss priority cells should be discarded before cells of high loss priority if congestion occurs and necessitates a loss of cells. Notice that cell priority may be indicated explicitly for each cell. The header error control (HEC) field uses a cyclic redundancy check for error protection of the cell header. It provides single bit-error correction and multiple bit-error detection.

ATM is a connection-oriented technology. It uses the concept of virtual path to make connection between each pair of network nodes. Connections can be established dynamically by the communicating partners themselves via a signalling protocol. This process creates an entry in a table at user network interface (UNI) that is used to map the cell header's VCI label into the internal (manufacturer dependent) information necessary to route the cell to the appropriate physical output port of the switching fabric, as shown in Fig. A.2.

A.2 ATM Switch Architectures

A switch performs three basic functions: routing to connect inputs to their output destinations, scheduling or arbitrations when cells arriving simultaneously at different inputs are destined for the same output, and (optionally) queueing at switch inputs or outputs to hold those cells that lose the arbitration. An important characteristic of a packet switch is its blocking probability. In an ATM switch this characteristic determines the *cell loss rate*. There are two different sources of blocking in a switch. Internal blocking is due to the limited number of routing hardware resources inside the switch. This kind of blocking can be eliminated by providing wider data paths or large



Input Line	Header	Output Line	Header
I_1	a b c	O_x O_y O_z	i j k
• • •	• • •	• • •	• • •
I_N	a b c	O_u O_v O_w	l m n

(b)

Fig. A.2: An ATM switch: a) data flow through the switch fabric, b) translation table

internal buffers. The other source of blocking is due to output contention caused by the simultaneous arrival of several cells at the same output. The degree of output blocking depends on the statistics of the incoming cells. While it may not be totally eliminated, output blocking can be reduced arbitrarily by using larger buffers to temporarily store the extra cells forwarded to a particular output.

Although several buffering disciplines have been reported for fast packet switching [46], not many of these architectures have been implemented. Implementing a design in actual hardware poses real challenges, and many good designs on paper are not feasible for implementation [67]. Based either on the structure of the underlying fabric or the placement of the buffers, switch fabrics can be classified in a number of ways. Based on the fabric structure, there are three basic architectures: the shared-memory, the shared-medium (bus), and the space-division switches [68].

A shared-memory switch consists of a single dual-ported memory shared by all input and output lines. The incoming cells are multiplexed and are written to the shared memory, as shown in Fig. A.3. In the shared-memory architecture [69]-[73], the control logic must be able to handle, in a single time slot, incoming cells from N inputs, enqueue them in the proper addresses in the shared memory, and select up to N outgoing cells for output. If we assume that the speed of each input line is equal to F Mbit/s, shared-memory switches need a memory throughput of $2NF$. If N is large then, the controller speed as well as the access speed of the buffer memory become bottle-neck for the high bit-rate operation. This restricts the number of input-output lines of a single switch and hence its scalability. A new organization for a shared buffer based on multiple memory banks, addressed in a pipelined fashion, has been presented in [74], [75]. The centralized controller remains to be the major bottleneck in the design. Thus, fully-shared memory switches have the advantage of requiring less memory to meet a given loss probability, but they have high control

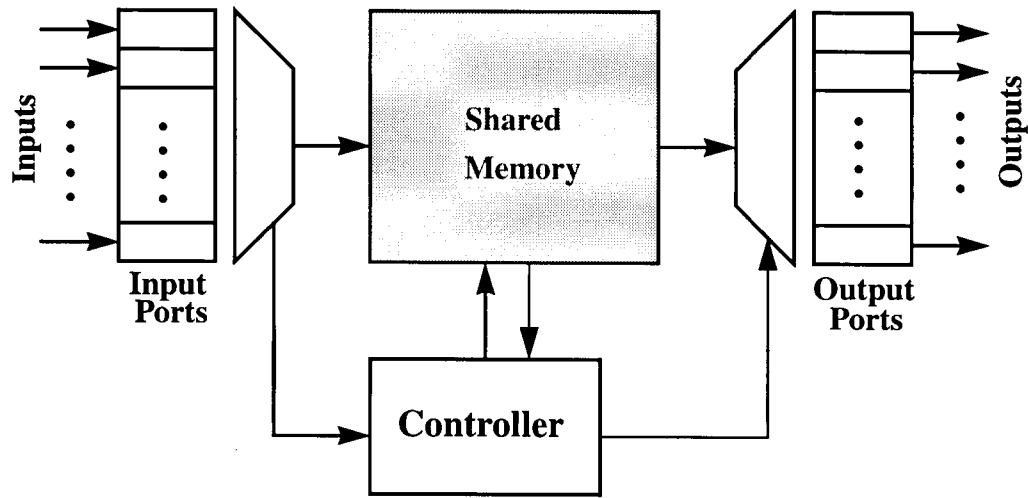


Fig. A.3: Shared buffer switch architecture

complexity, and relatively long access time.

A shared-medium switch consists of a common high-speed medium (typically a high-speed bus) onto which the input lines are synchronously multiplexed. At the output, there is an address filter and a first-in-first-out (FIFO) buffer for each output line. The function of the address filter is to extract the packets which are meant for a particular output and to store them in the FIFO buffer for that output. In an $N \times N$ shared-medium switch, the bandwidth of the bus must be equal to N times the rate of a single input line. Therefore, as the number of links attached to the medium and their speeds increase, the medium speed becomes a bottleneck. The first shared medium switch for ATM was ATOM (ATM output buffer modular) switch [76]. It applied a bit-slice organization to alleviate the bottleneck of the medium speed. Another shared-medium switch, which has been called PARIS (packetized automated routing integrated system), was designed for variable length packets [76]. More recently a bus structure based on sequentially scheduling the inputs and paral-

lel transfer has been reported [77]. However, a parallel bus of 425 wires has been used, which seems to be highly excessive. Because the shared-medium supports broadcasting and multicasting naturally, it is not surprising that multicasting is very easy to implement with a shared bus. However, the shared-medium architecture results in separate buffers for each output, and hence requires more memory in order to achieve a required cell loss probability.

Space-division switches are classified into two major categories based on their routing capabilities [78], non-blocking and blocking switches. In non-blocking switches, internal blocking within a switch will not occur due to the existence of a sufficiently large number of non-overlapping internal paths. However, blocking may still happen at the output ports of the switch. In blocking switches, internal blocking can occur when two or more packets contend for the same link within the switch. Internal blocking and output contention cause degradation in throughput. To reduce throughput degradation, buffers may be provided at the input ports (input queuing), output ports (output queuing), or internally at the switching elements (internal or cross-point queuing). Each buffering method is characterized by different types of drawbacks. Input queueing with FIFO buffers is simple to implement but has the worst performing architecture. Mainly, input buffering has low throughput (about 58.6%) due to head-of-line (HOL) blocking effects [79]. In each routing step, only the front (head-of-line) packet of each input queue is considered for possible routing to its destined output; if it collides with the front packet from another queue, then it has to wait, and so do all the packets in its queue, even if they were destined to currently idle outputs. This is known as head-of-line blocking. It can be increased to 70% by having two cells in each buffer competing for the output ports in each cell time interval [68].

With output queuing, all cells contending for the same output port are stored in the output buffers until they can be read out. Output buffering increases the throughput over input buffering, since

more than one cell can be delivered to the output when output contention occurs, which is not possible with input buffering. Output queueing achieves optimal delay/throughput performance [80] but requires switches (or at least the output buffers) to operate at a faster rate than the effective or peak rate of the incoming traffic. Pure output buffering exerts poor utilization of buffers, because buffers are exclusively dedicated to each output. Because of the above limitations of input or output buffered switches, the combined input/output buffered switch is a good compromise which has been proposed and studied by some researchers [42], [81], [120].

Buffers may also be placed at the cross-points of the crossbar switch (internal buffering) [82]. ATM switches of this type have similar performance to output queueing switches with the difference that the queue for each output is distributed among N buffers (for an $N \times N$ switch). Internal buffering is used to alleviate internal blocking, but it has several limitations: the complexity of the switching elements is increased and the internal buffers introduce random delays within the switching fabric, causing undesired cell-delay variation in the network. This approach also suffers from a rapid nonlinear growth in memory complexity with switch size.

Because HOL blocking degrades performance in the worst case [83], the standard approach has been to abandon input buffering and instead use output buffering. By increasing the bandwidth of the internal interconnect, multiple cells can be forwarded at the same time to the same output, and queued there for transmission on the output link. However, HOL blocking effects can be greatly reduced by using a separate queue for each output [80], where any cell queued at the head of an input queue is eligible for forwarding across the switch. The central problem with random-access input queues is the need for fast scheduling - quickly finding a maximal conflict-free set of cells to be forwarded across the switch, such that each input is connected to at most one output, and vice

versa. Work in [84] has evaluated some alternative approaches to the scheduling of cells in a high bandwidth input-queued ATM switch. These approaches are based on distributed deterministic scheduling with rotating priorities, but the control overhead is high and their actual efficiency remains to be verified.

We should think of speed-up as a logical concept since the implementation can be done by increasing the clock speed, by increasing the parallelism, or by increasing both. As we mentioned earlier, shared buffer switches need a memory throughput of $2NF$ (N : number of input/output lines, F : speed of each input line). However, output buffering switches need speeds of only $(N+1)F$. In this sense, output buffering switches are superior to shared buffer switches due to their simple output queue control, and more relaxed memory access time requirements. On the other hand, in the combined input/output buffer type switch, internal switching must be L times faster than the input or output port speed to reduce the effects of HOL blocking (L : speed ratio within a switch). Therefore, the memory access speed required for the combined input/output buffers is $(L+1)F$. Finally, the required speed for each buffer in internal buffering method is $2F$.

A.3 ATM Buffer Management

In an ATM network, various types of traffic are multiplexed to efficiently utilize the network bandwidth and resources. Therefore, ATM networks have to provide multimedia services with diversified traffic flow characteristics and QoS requirements. In such environments, the bandwidth efficiency can be improved by defining multiple QoS classes and using a flow control method which manages the required QoS of each class individually. A priority buffer manager provides a QoS control method for an ATM switching node. The buffer manager at each output port of an ATM switch, as shown in Fig. A.4, is responsible for scheduling cell transmissions and selectively

discarding cells. Cells with higher delay (and/or delay-jitter) priority will be transmitted first, while cells with higher loss priority will be discarded last when the buffer is full. Note that the assignment of the delay and loss priorities to each service can be completely independent. For instance, voice and interactive video services need higher delay priority but lower loss priorities because they can tolerate higher cell loss rates without noticeable imperfection. Conversely, data traffic requires higher loss priorities but lower delay priorities.

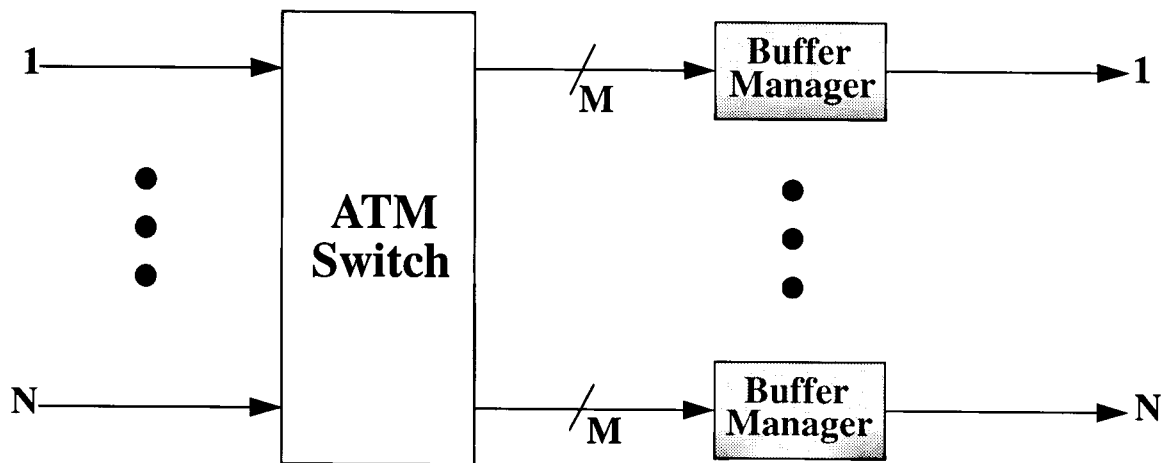


Fig. A.4: The buffer manager at each output port of an ATM switch

APPENDIX B

Asynchronous System Design

During the last decade there has been a revival in research on asynchronous circuits. Synchronous digital design is based on two major assumptions: all signals are binary, and time is discrete. Asynchronous circuits maintain the assumption that signals are binary, but remove the assumption that time is discrete. The asynchronous scheme with an appropriate handshake protocol is a design discipline where the sequencing of events is controlled by the internal delays of system elements rather than by an external clock. In place of a global clock, an asynchronous system only needs a reset signal and external handshaking signals to synchronize its operation. Internally, stages communicate at their own speed. To highlight the difference between the synchronous and asynchronous timing disciplines, the following example was presented [85]. A synchronous sys-

tem works like a scheduled train line. A particular passenger has to synchronize his/her travel plan with the schedule of the train. On the other hand, an asynchronous (or self-timed) system is like traveling with your own car. There is no fear of missing a scheduled departure time, and no waiting in a station for a train to arrive. You may visit a new place whenever you have finished visiting an old location. It is a sequence of events and we are interested only in ordering them.

The potential advantages of self-timed systems over their synchronous counterparts have created a resurgence of interest in asynchronous design methodologies. One important design consideration in synchronous designs is clock skew, which is the difference in arrival times of the clock signal at different parts of the circuit. The asynchronous design approach eliminates the need for a global clock and circumvents the problems arising from clock skew. This is becoming an important consideration when large amounts of both chip area and design time are dedicated to clock circuits and clock distribution. Currently, in some applications, significant silicon area and design effort are required for clock generation and distribution to maintain skew within acceptable limits. These factors become progressively more difficult as feature sizes shrink. With the advances in technology, the silicon area penalty of using asynchronous design is becoming less significant, and the performance penalty is being similarly reduced.

A self-timed design also has the potential for reduced power consumption relative to its synchronous counterpart. A synchronous circuit is either quiescent (i.e., the clock is turned off) or active entirely (i.e., clock on). An asynchronous circuit, in contrast, only consumes energy when and where active. However, it is not obvious to what extent this advantage is fundamentally asynchronous. Synchronous techniques such as clock gating may achieve similar benefits, but they have their limitations. In general, the potential of asynchronous for low-power depends on the application [64].

From a system design point of view, asynchrony means design modularity. As digital systems become complex, it is advantageous to adopt a modular design approach to simplify the design task to local timing considerations only. However, asynchronous circuits are more difficult to design in an ad hoc fashion than synchronous circuits.

Another important advantage of using asynchronous systems is the separation of timing from functionality which means that, the same circuit can be implemented in a variety of technologies without modification to the circuit [86], [89], [110], [111]. These potentials of asynchronous digital system design have stirred the interest of many researchers and industries. Asynchronous design is now progressing from a fashionable academic research topic to a viable solution to a number of digital VLSI design challenges [64]. Researchers have a great liking for creating new terms. As a result, asynchronous system design has also been called self-timed [90]-[96], data-driven [97], [98], micropipelined [99], [100], delay-insensitive [101], [102], or speed-independent [103] system design.

The discipline of timing management is the key to the effective design of any large scale digital system. In a synchronous system, we know when data are valid because the clock signal is asserted. In self-timed circuits, we need another method of determining when data are valid. We require a strategy based on some form of a request-acknowledge handshaking scheme. Seitz [85] illustrates two different request-acknowledge protocol strategies: a four-phase signalling protocol (also known as return-to-zero protocol, or Muller signalling), and a two-phase signalling protocol (also known as non-return-to-zero protocol, or transition-signalling).

The two-phase signalling is the most energy efficient and least time consuming signalling scheme. Fig. B.1 illustrates the working of this transition signalling protocol. Using a transition as a basic

event, a sender can issue a request by causing a transition on the request wire (*Req*). The sending module is responsible for keeping the information on the data wires valid for as long as the receiver desires. After a certain time, depending on the delay of the element, the receiver can acknowledge that the data is no longer needed by causing a transition on an acknowledge wire (*Ack*). We observe that there are two transitions made in one data transaction and it takes two transfers to go back to the original state. The term two-phase is used because of these two distinguishable states in each transaction.

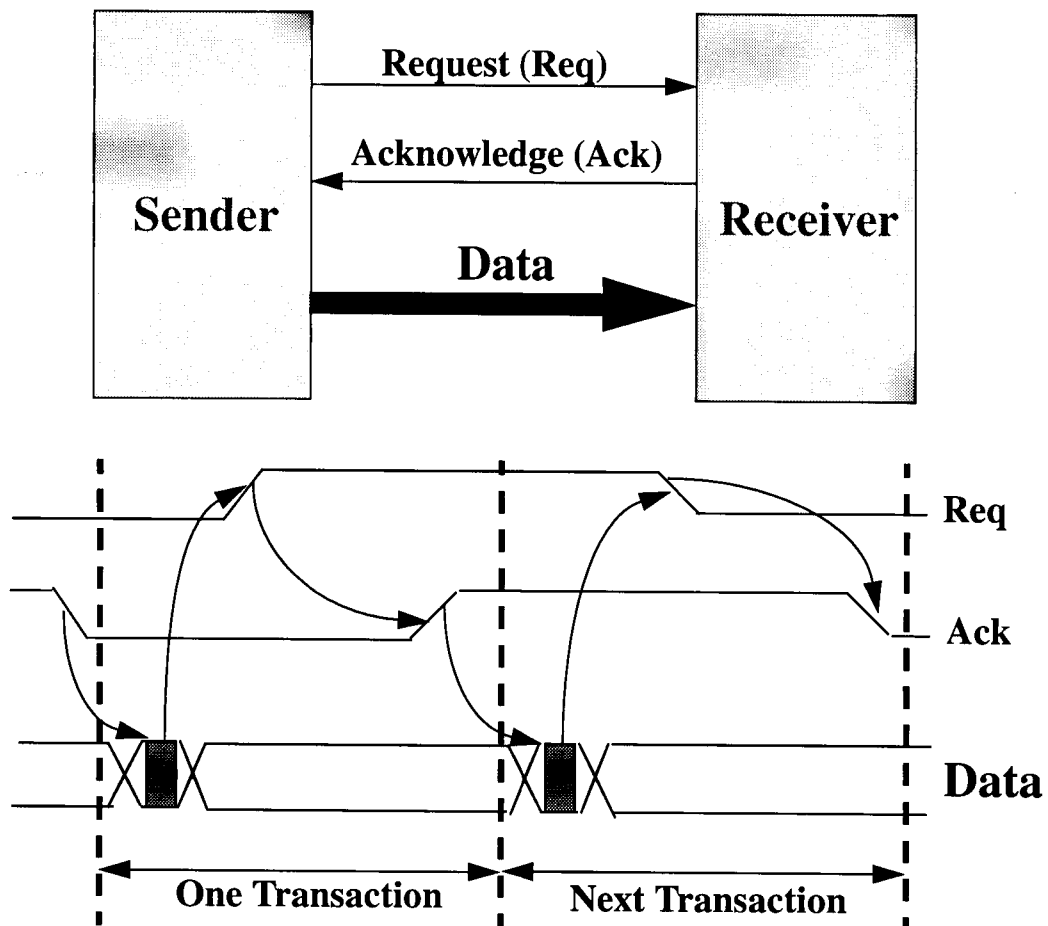


Fig. B.1: Two-phase signalling scheme

In four-phase signalling scheme, there are total of four transitions made to the request and acknowledge signals in one transfer. Fig. B.2 shows a complete four-phase transaction with each of its four distinct phases. The four-phase handshake protocol always uses the rising transitions to initiate operation and the falling transitions to reset. It is common to initialize the signal wires to a low state. This means that after a complete four-phase transaction the signal wires again both be low. The four phases are:

1. Quiescent: *Req* low and *Ack* low. This indicates that no transaction is in progress.
2. Requesting: *Req* high and *Ack* low. The sender has issued a request, and the receiver has not yet acknowledged.
3. Acknowledging: *Req* high and *Ack* high. The receiver has acknowledged the request.
4. Clearing: *Req* low and *Ack* high. The sender has initiated the clearing phase of the protocol by dropping its request signal. The receiver has not yet responded by lowering its acknowledgment.

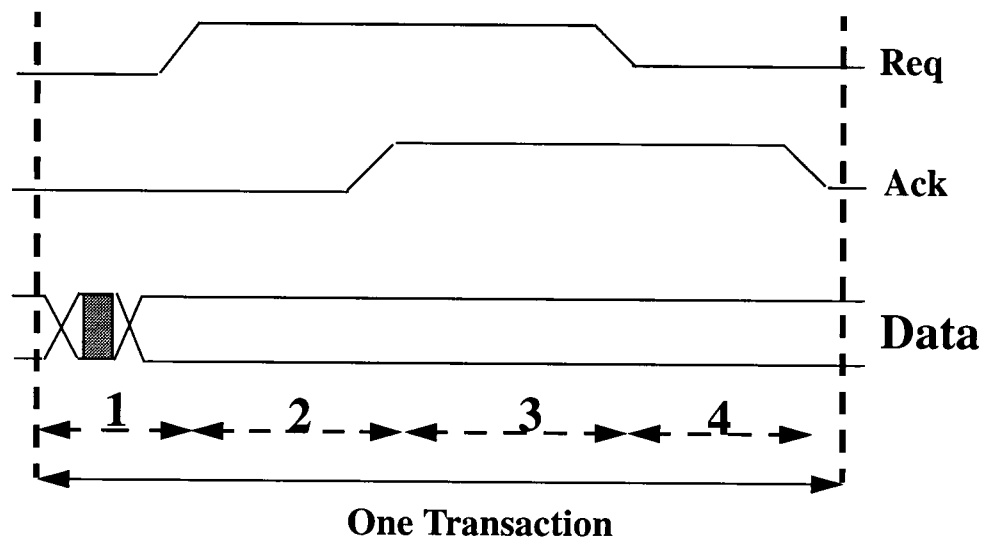


Fig. B.2: Four-phase signalling scheme

Two-phase signalling means that both the rising and falling edges of the signals can flag an action. In contrast, the four-phase scheme uses only the rising or the falling transition to flag actions. Both the two-phase and the four-phase protocols are delay-insensitive and thus are very useful for communication between asynchronous modules. Two-phase signalling can be faster because it requires only half the number of the transitions that four-phase signalling requires. Obviously, systems designed using the two-phase signalling convention will have twice as much throughput than systems designed using the four-phase signalling scheme. Additionally, less power is consumed when fewer transitions are required. However, two-phase signalling may require special circuits not normally available as packaged parts, and more strict design constraints than systems employing the four-phase signalling convention. In VLSI, four-phase signalling are more suited to modules that deal with data rather than just control signals. Since only a rising edge initiates a communication, four-phase circuit structures generally are simpler than their two-phase counterparts.

Sutherland introduced a timing discipline called the transition-signalling framework, based on what was called micropipelines [99]. Micropipelines use the two-phase signalling (also called bundled data convention [99]) as proposed by Seitz [85]. For the successful operation of micropipelines using the two-phase bundled data convention, the delays in data transmission across stages should be less than the delays encountered in transmitting control signals. The work reported in [99] uses static CMOS to implement the data processing blocks of the pipeline. This suggests that in general a delay element needs to be introduced in the control wires of the bundle to satisfy the timing requirements. Jacobs, Broderson [90], [91], and Meng [103] have described another approach to design using the transition signalling framework and a four-phase signalling scheme. They use dynamic cascaded voltage switch logic (DCVSL) to implement the logic processing

blocks [104].

Control circuitry for transition signalling is necessary to perform logical combination of events besides the self-timed elements. Control circuits for the above request-acknowledge protocols are built out of modules that form various logical combinations of events. The exclusive OR circuit acts as the OR element for events. A Muller C-element acts as the AND element for events. A C-element is a bi-stable element whose output goes high only when all inputs are high and whose output returns low only when all inputs are low; otherwise the output stays unchanged.

The C-element is one of the basic units in the early approaches to designing self-timed circuits, and many researchers have come up with its circuitry [105]-[107]. The conventional implementation of the C-element, however, suffers from *early threshold* problems which are a prime source of hazards and should be avoided in asynchronous circuits [108]. The term early threshold refers to low logic thresholds for up-going transitions and high logic thresholds for down-going transitions. To avoid such hazardous behavior, a new design for the C-element has been developed to overcome the early threshold voltage problem [62]. Hysteresis is the property of this circuit in which the input threshold level changes as a function of the output level. In particular, when the input level passes the threshold level, the output level changes and the input threshold level is subsequently reduced so that the input level must return beyond the previous threshold level before the C-element's output changes state again. This modification eliminates the early threshold voltage problem. The proposed C-element circuit is of hysteretic nature with its implementation based on the Schmitt trigger circuit [109].

Since, the clocked-logic conceptual framework is poorly matched to FIFO memory design [99], self-timed FIFO presented in a number of applications [26], [64], [79]. In self-timed approach,

each stage of FIFO operates at its own pace, using control information only from adjacent stages. There are two control wires between stages and data flow is synchronized with the signals on the control wires, but not with an external clock signal. The data signals automatically cascade through the stages and the propagation gate delay is asynchronous with any external signal. External control signals are needed only at the FIFO entrance and exit. The time taken at each stage, which depends on the interstage propagation delay, determines the upper limit of the FIFO's operating frequency. Besides, the self-timed FIFO does not require complex clock distribution and the storage capacity can be easily expanded.

There are two versions of the self-timed FIFO: a two-phase FIFO, and a four-phase FIFO. For both types, a string of Muller C-elements interspersed with inverters is the only logic required to control the pipeline. In two-phase FIFO shown in Fig. B.3, a set of event-controlled storage registers (two-phase registers) connected in series serve as the FIFO data path, while a string of Muller C-elements serve as its control. The event-controlled storage element (two-phase register) is required to respond to both the rising and falling voltage transitions, such that a new value is stored in the register on each transition of the control signals.

A 4-cycle FIFO can be developed in a similar discussion. A very simple circuit can be derived for the event control of a 4-cycle FIFO, as shown in Fig. B.4. Note that now two C-elements are required to implement the four-phase handshake between FIFO stages. A four-phase handshake can be also viewed as two, two-phase, handshakes executed in sequence. There is also a NOR gate to assure that input data are not allowed to pass through the register until the final falling acknowledge signal of the next stage.

A significant difference between the storage elements (registers) used in the two FIFO's is that the

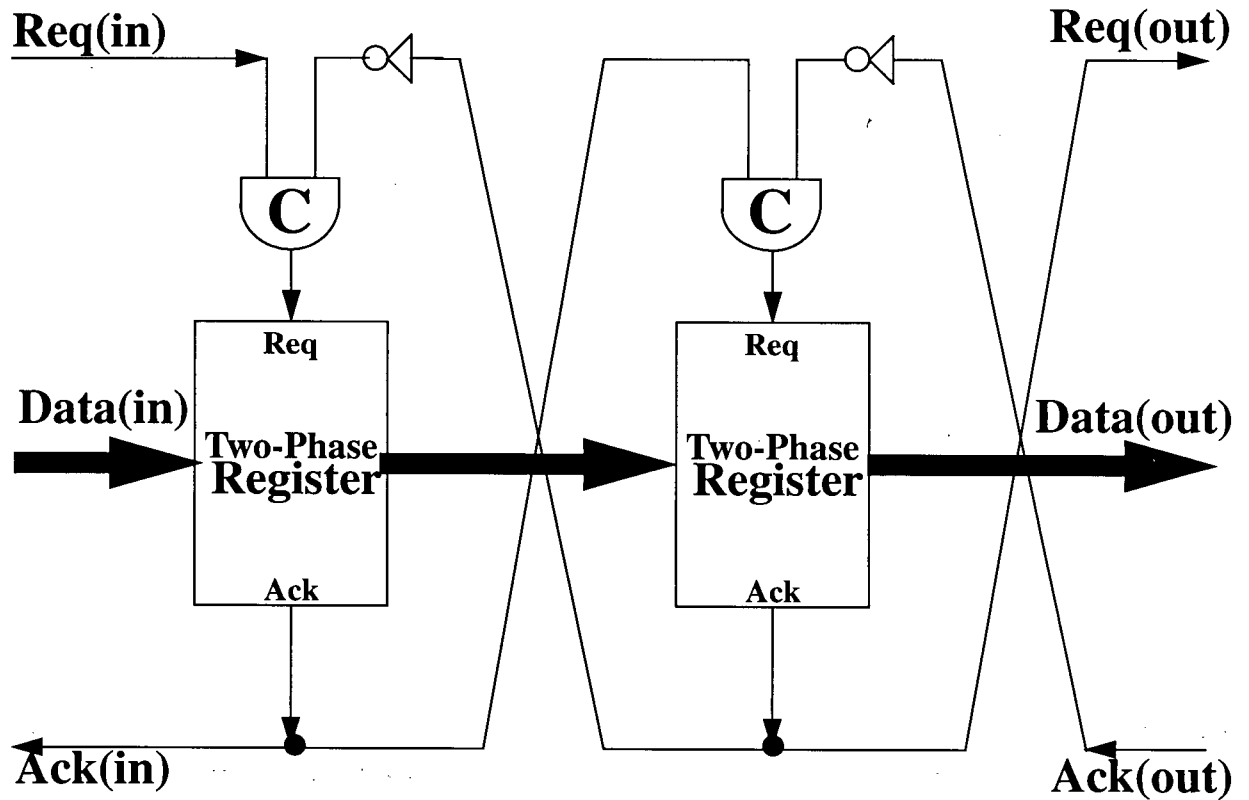


Fig. B.3: Two-phase FIFO structure

4-cycle storage elements will respond only on the rising transitions, or if the level of the request input is high. Naturally, data must propagate through a two-phase self-timed FIFO faster than the control events propagate through its control. Sutherland [99] assures this as follows. First, the C-elements used in the control circuit are more complex than the storage element used in the data path, and are inherently slower. Second, the layout of the circuit ensures that the zigzag path of the control signals has longer wires than those in the data path. If above condition can not be guaranteed, the request (*Req*) signal arrival at a stage must be delayed by an amount equivalent to the computation time of the current block.

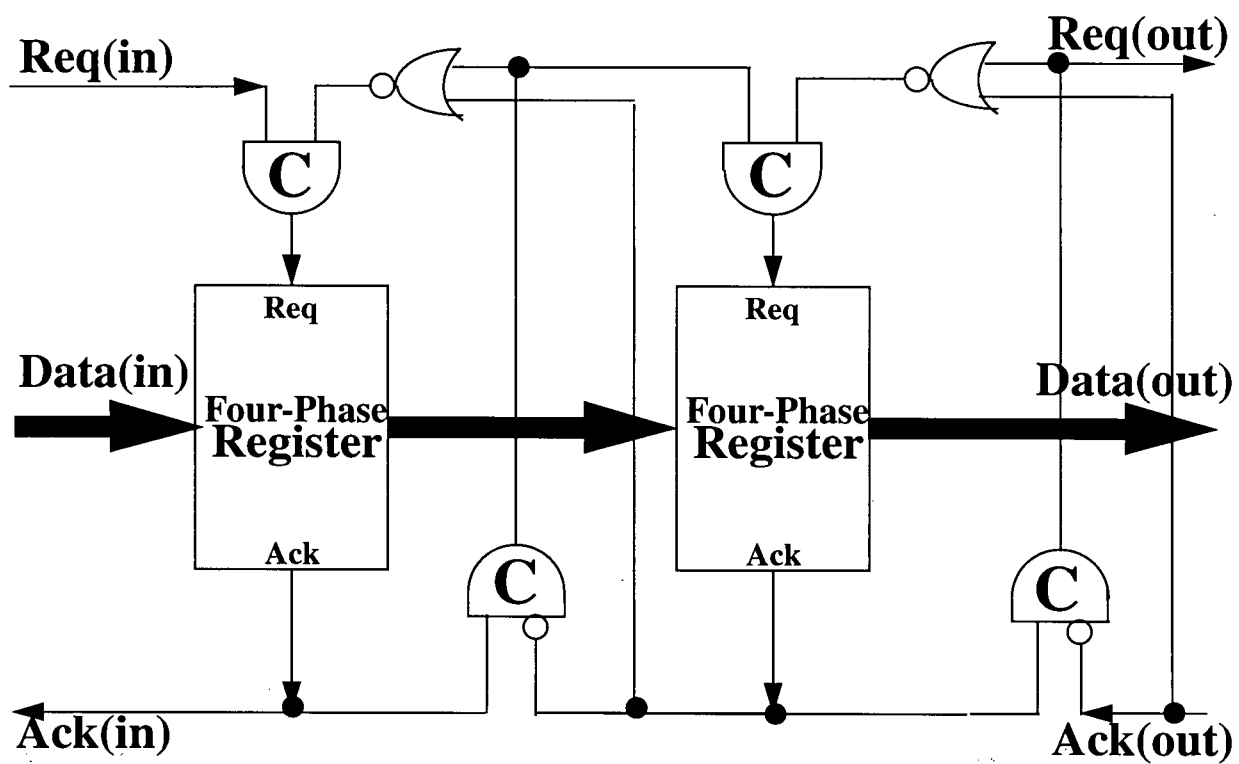


Fig. B.4: Four-phase FIFO structure