Crossbar Arbitration in Interconnection Networks
for Multiprocessors and Multicomputers

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Computer Science

by

Hsin-Chou Chi

1994
To my parents,

for their love and understanding.
# Table of Contents

## Chapter One - Introduction

1.1. Crossbar Arbitration ................................................................. 5

1.2. Organization of the Dissertation .............................................. 8

## Chapter Two - Previous Work

2.1. Arbitration Problems in Computer Systems .............................. 11

2.1.1. Conventional Bus Arbitration .............................................. 12

2.1.2. Multiple Bus Arbitration ..................................................... 14

2.1.3. Segmented Ring Bus Arbitration .......................................... 16

2.2. Buffered Communication Switches ........................................... 18

2.2.1. Performance Impact of Switch Buffers ................................. 19

2.2.2. Communication Switch Designs ........................................... 21

2.3. The ComCoBB Project .............................................................. 25

2.3.1. High-Performance Multi-Queue Buffers ............................... 26

2.3.2. Support for High-Priority Traffic ........................................ 29

2.3.3. Dynamic Virtual Circuits .................................................... 32

## Chapter Three - Symmetric Crossbar Arbiters

3.1. Symmetric Crossbar Arbitration .............................................. 39

3.2. Symmetric Crossbar Arbiters .................................................. 43

3.2.1. The Importance of the Arbitration Scheme .......................... 44
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.2.</td>
<td>The Skewed Two-Step Arbiter</td>
<td>49</td>
</tr>
<tr>
<td>3.2.3.</td>
<td>The Wave Front Arbiter</td>
<td>50</td>
</tr>
<tr>
<td>3.2.4.</td>
<td>The Wrapped Wave Front Arbiter</td>
<td>52</td>
</tr>
<tr>
<td>3.3.</td>
<td>Performance Evaluation</td>
<td>54</td>
</tr>
<tr>
<td>3.3.1.</td>
<td>Simple Probabilistic Analysis for a 2×2 Switch</td>
<td>54</td>
</tr>
<tr>
<td>3.3.2.</td>
<td>Probabilistic Analysis for an n×n Switch</td>
<td>56</td>
</tr>
<tr>
<td>3.3.3.</td>
<td>Simulation of Synchronous Networks</td>
<td>65</td>
</tr>
<tr>
<td>3.3.3.1.</td>
<td>Performance Measurement</td>
<td>65</td>
</tr>
<tr>
<td>3.3.3.2.</td>
<td>Performance Evaluation of a 4×4 Switch</td>
<td>69</td>
</tr>
<tr>
<td>3.3.3.3.</td>
<td>Performance Evaluation of a 64×64 Network</td>
<td>71</td>
</tr>
<tr>
<td>3.3.3.4.</td>
<td>Impact of Buffer Size and Switch Size</td>
<td>73</td>
</tr>
<tr>
<td>3.3.4.</td>
<td>Simulation of Asynchronous Networks</td>
<td>77</td>
</tr>
<tr>
<td>3.4.</td>
<td>Implementation</td>
<td>85</td>
</tr>
<tr>
<td>3.4.1.</td>
<td>Logic and Circuit Design</td>
<td>86</td>
</tr>
<tr>
<td>3.4.2.</td>
<td>VLSI Layout and Circuit Simulation</td>
<td>88</td>
</tr>
<tr>
<td>3.5.</td>
<td>Summary and Conclusions</td>
<td>91</td>
</tr>
<tr>
<td>Chapter Four - Decomposed Arbiters for Large Crossbars</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>4.1.</td>
<td>Performance Impact of Large Crossbars</td>
<td>97</td>
</tr>
<tr>
<td>4.2.</td>
<td>Decomposed Arbitration for Large Crossbars</td>
<td>101</td>
</tr>
<tr>
<td>4.3.</td>
<td>Performance Evaluation</td>
<td>110</td>
</tr>
<tr>
<td>4.4.</td>
<td>Design of Decomposed Arbiters</td>
<td>118</td>
</tr>
<tr>
<td>4.5.</td>
<td>Reducing the Number of Queues</td>
<td>125</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.6. Summary and Conclusions</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter Five - Starvation Prevention</strong></td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>5.1. Starvation Prevention for Crossbar Arbitration</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>5.1.1. Starvation Prevention in Synchronous Networks</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>5.1.2. Starvation Prevention in Asynchronous Networks</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>5.2. Performance Evaluation</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>5.2.1. Uniform Traffic</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>5.2.2. Nonuniform Traffic</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>5.2.2.1. Simulations for a 4×4 Switch</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>5.2.2.2. Simulations for a 16×16 Network</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>5.3. Implementation</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>5.4. Summary and Conclusions</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter Six - Arbiter Support for Adaptive Routing and</strong></td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>for High-Priority Traffic</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>6.1. Arbiter Design Supporting Adaptive Routing</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>6.2. Arbiter Design Supporting High-Priority Traffic</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter Seven - Summary</strong></td>
<td>181</td>
<td></td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>185</td>
<td></td>
</tr>
</tbody>
</table>
List of Figures

1.1 An Indirect Network for Multiprocessors ............................................. 2
1.2 A Direct Network for Multicomputers ................................................ 2
1.3 Organization of a Typical Communication Switch ................................. 3
1.4 A Switch with FIFO Input Buffers ....................................................... 5
1.5 A Switch with Multi-Queue Input Buffers ............................................ 7
1.6 A Switch with DAMQ Input Buffers ...................................................... 7
2.1 A Shared-Bus Multiprocessor ............................................................... 12
2.2 A User-Resource Model of Shared Bus Arbitration ......................... 13
2.3 A Multiple-Bus Multiprocessor ............................................................. 15
2.4 A User-Resource Model of Multiple Bus Arbitration .......................... 16
2.5 A Multiprocessor Based on the Segmented Ring Bus ....................... 17
2.6 A User-Resource Model of Segmented Ring Bus Arbitration ............. 18
2.7 Two Different Buffer Designs .............................................................. 20
2.8 Four Alternative Buffer Designs for Communication Switches .......... 28
2.9 A Virtual Circuit .................................................................................. 34
2.10 A Multicomputer Node ............................................................... 37
3.1 Switches with FIFO Buffers and DAMQ Buffers ............................. 40
3.2 A Crossbar Arbiter in Context ............................................................. 41
3.3 A User-Resource Model of Symmetric Crossbar Arbitration ........... 42
3.4 Example Arbitrations for FIFO and DAMQ Switches ......................... 43
3.5 A Two-Step Symmetric Crossbar Arbiter ........................................ 46
3.6 Normalized Throughput Based on Probabilistic Analysis ................. 48
3.7 A Skewed Two-Step Symmetric Crossbar Arbiter ......................... 50
3.8 Wave Front Symmetric Crossbar Arbitration .................................... 51
3.9 A Wrapped Wave Front Symmetric Crossbar Arbiter ..................... 53
3.10 Probabilistic Analysis of Throughput for 2×2 Switches ................. 56
3.11 Probabilistic Analysis of Throughput for 4×4 Switches ................ 63
3.12 Probabilistic Analysis of Throughput for 8×8 Switches ............... 63
3.13 Probabilistic Analysis of Throughput for 16×16 Switches ............. 64
3.14 Probabilistic Analysis of Throughput for 64×64 Switches .......... 64
3.15 Average Latency of a 4×4 Switch ............................................. 70
3.16 99th Percentile Latency of a 4×4 Switch ................................... 70
3.17 Average Latency of a 64×64 Network ..................................... 71
3.18 99th Percentile Latency of a 64×64 Network ........................... 72
3.19 The Impact of Input Buffer Size on Average Latency ................. 73
3.20 The Impact of Input Buffer Size on 99th Percentile Latency .......... 74
3.21 The Impact of Switch Size ...................................................... 76
3.22 Performance of a 4×4 Switch in Asynchronous Networks I .......... 79
3.23 Performance of a 4×4 Switch in Asynchronous Networks II .......... 79
3.24 Performance of an Asynchronous 64×64 Network I .................. 80
3.25 Performance of an Asynchronous 64×64 Network II ................ 80
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.26</td>
<td>Performance of a 4×4 Switch in Asynchronous Networks III</td>
<td>81</td>
</tr>
<tr>
<td>3.27</td>
<td>Performance of a 4×4 Switch in Asynchronous Networks IV</td>
<td>81</td>
</tr>
<tr>
<td>3.28</td>
<td>Performance of an Asynchronous 64×64 Network III</td>
<td>82</td>
</tr>
<tr>
<td>3.29</td>
<td>Performance of an Asynchronous 64×64 Network IV</td>
<td>82</td>
</tr>
<tr>
<td>3.30</td>
<td>Performance of an Asynchronous 64×64 Network V</td>
<td>84</td>
</tr>
<tr>
<td>3.31</td>
<td>Performance of an Asynchronous 64×64 Network VI</td>
<td>84</td>
</tr>
<tr>
<td>3.32</td>
<td>Organization of a 2×2 Wave Front Arbiter</td>
<td>85</td>
</tr>
<tr>
<td>3.33</td>
<td>Organization of a 2×2 Wrapped Wave Front Arbiter</td>
<td>86</td>
</tr>
<tr>
<td>3.34</td>
<td>An Arbitration Cell</td>
<td>87</td>
</tr>
<tr>
<td>3.35</td>
<td>Horizontal and Vertical Priority Token Rings</td>
<td>88</td>
</tr>
<tr>
<td>3.36</td>
<td>The Floorplan of a 4×4 Crossbar with an Arbiter</td>
<td>89</td>
</tr>
<tr>
<td>3.37</td>
<td>VLSI Layout of a Single WFA Arbitration Cell</td>
<td>90</td>
</tr>
<tr>
<td>4.1</td>
<td>Base Network Latency vs. Switch Size I</td>
<td>99</td>
</tr>
<tr>
<td>4.2</td>
<td>A 6×6 Decomposed Arbiter</td>
<td>101</td>
</tr>
<tr>
<td>4.3</td>
<td>Base Network Latency vs. Switch Size II</td>
<td>105</td>
</tr>
<tr>
<td>4.4</td>
<td>Analysis for the Generalized Decomposed Arbitration</td>
<td>107</td>
</tr>
<tr>
<td>4.5</td>
<td>Optimal Switch Size vs. Arbitration Subarray Size</td>
<td>108</td>
</tr>
<tr>
<td>4.6</td>
<td>Average Latency of an 8×8 Switch</td>
<td>112</td>
</tr>
<tr>
<td>4.7</td>
<td>99th Percentile Latency of an 8×8 Switch</td>
<td>112</td>
</tr>
<tr>
<td>4.8</td>
<td>Average Latency of a 16×16 Switch</td>
<td>113</td>
</tr>
<tr>
<td>4.9</td>
<td>99th Percentile Latency of a 16×16 Switch</td>
<td>113</td>
</tr>
<tr>
<td>4.10</td>
<td>Average Latency of a 32×32 Switch</td>
<td>114</td>
</tr>
</tbody>
</table>
4.11 99th Percentile Latency of a 32×32 Switch ........................................ 114
4.12 Average Latency of a 64×64 Network ............................................... 115
4.13 99th Percentile Latency of a 64×64 Network .................................... 115
4.14 Average Latency of a 256×256 Network ........................................... 117
4.15 99th Percentile Latency of a 256×256 Network .................................... 117
4.16 Organization of a 2×2 Wrapped Wave Front Arbiter ............................ 119
4.17 A 4×4 Decomposed Arbiter ............................................................ 120
4.18 An Arbitration Cell ........................................................................... 121
4.19 Logic Design of the Arbitration Cell .................................................. 121
4.20 The Impact of Varying the Number of Queues I ................................. 127
4.21 The Impact of Varying the Number of Queues II ................................. 127
4.22 The Impact of Varying the Number of Queues III ............................... 128
4.23 The Impact of Varying the Number of Queues IV ............................... 128
5.1 A 2×2 Starvation-Free Wrapped Wave Front Arbiter .................................. 134
5.2 An Example of Starvation .................................................................... 136
5.3 Average Latency of a 4×4 Switch I ...................................................... 140
5.4 99th Percentile Latency of a 4×4 Switch I ............................................ 140
5.5 Comparison of SGR Arbiters and CGR Arbiters .................................. 141
5.6 Average Latency of a 2×2 Switch ....................................................... 142
5.7 Average Latency of an 8×8 Switch ..................................................... 142
5.8 Average Latency of a 16×16 Network I ............................................... 143
5.9 99th Percentile Latency of a 16×16 Network I ..................................... 143
List of Tables

4.1 Analysis of the Base Network Latency I .............................................. 100
4.2 Analysis of the Base Network Latency II ............................................. 106
ACKNOWLEDGMENTS

During the years I stayed as a student at UCLA, many people have taught me or helped me along the way. I wish to express my gratitude to all of them.

First and foremost, I would like to thank my advisor, Yuval Tamir, for introducing me to the computer architecture area. The discussions with him in the course of this work have enlightened me and broaden my view in the pursuit of knowledge. His constant encouragement of finding the best way to present ideas clearly has benefited me a great deal. This work would not have been possible without his guidance and inspiration.

I am also grateful to Prof. David Rennels, Prof. Milos Ercegovac, and Prof. Thomas Ferguson for being on my committee and providing many helpful comments. Prof. Rennels and Prof. Ercegovac have always been supportive during this research. Their encouragement has helped me achieve what I have accomplished.

A million thanks are owed to Verra Morgan. She has smoothed numerous difficulties away for me time and again. She will be forever remembered.

Over the years, I have enjoyed the discussions with my fellow officemates in the Computer Science VLSI Systems Laboratory, including Greg Frazier, Tiffany Frazier, Yoshio Turner, John Janakiraman, and Gonzalo Archondo-Callao. I thank them all.

My life-long good friend, Kuo-Chun Lee, and his wife, Hsing-Huei Wen, have been instrumental to what I am now. When I am in need of help, they are always there. Their unreserved support has comforted a soul countless times when things looked dark. I am truly in their debt.

Thanks are also due to many friends and classmates of mine, especially
Lee-Chung Lu, Chee-Fai Yung, Chao-Liang Chen, Wang-Chang Su, Jen-Shan Lin, Shu-Li Yang, Heng-Hui Lue, Shen-Tzay Huang, Ying-Dar Lin, and Kong Li. I have always appreciated and enjoyed their friendships and support.

Finally, I would like to dedicate this dissertation to my parents. During this period I have been away from home, they have endured tough times. Their son is now coming home.
VITA

February 27, 1960 Born in Taichung, Taiwan

1982 B.S. Electrical Engineering
National Taiwan University
Taipei, Taiwan

1984 M.S. Electrical Engineering
National Taiwan University
Taipei, Taiwan

1984-1986 Instructor
Chinese Army Signal School
Chungli, Taiwan

1988-1993 Teaching Assistant, Teaching Associate, Teaching Fellow
University of California, Los Angeles
Los Angeles, California

1993-1994 Staff Research Associate
University of California, Los Angeles
Los Angeles, California

PUBLICATIONS AND PRESENTATIONS


xvi


ABSTRACT OF THE DISSERTATION

Crossbar Arbitration in Interconnection Networks for Multiprocessors and Multicomputers

by

Hsin-Chou Chi

University of California, Los Angeles, 1994

Professor Yuval Tamir, Chair

Small crossbars are key components of communication switches used in interconnection networks for multiprocessors and multicomputers. The traffic through the switches is often delayed due to conflicting demands for resources, such as buffer space or output ports. Hence, switches must include arbiters that resolve conflicting resource demands. Efficient design and implementation of these arbiters is critical for maximizing network performance.

In order to maximize performance, recent communication switch designs allow packets at an input port, destined to different output ports, to be transmitted through the switch in any order. In this dissertation several symmetric crossbar arbiters which can be used in this type of switches are proposed. These arbiter designs are compared based on simulations of a multistage interconnection network, VLSI implementation, and circuit simulation. Two efficient arbiter implementations are described.

For a fixed number of nodes, larger crossbars result in reduced probability of conflicts and allows packets to traverse the network in fewer hops. However, the increased arbitration delay can lead to overall poor network performance. A technique is proposed for the arbiter design in asynchronous networks, which can
mitigate the impact of the increased arbitration delay by decomposing the arbitration process into multiple steps.

While our symmetric crossbar arbiters lead to excellent performance in terms of throughput and average latency, they do not guarantee fairness. Hence, it is possible for an “unlucky” packet to be left in a switch buffer for a long time, potentially forever, while other packets are forwarded quickly through the switch. We introduce and evaluate a technique for preventing such starvation situations.

Finally, we describe how our proposed arbiters can be used to support adaptive routing and high-priority traffic in interconnection networks. Efficient arbiter designs for these two applications are described.
Chapter One
Introduction

The ability of multiprocessors and multicomputers to achieve high performance is dependent on interconnection networks that provide high-bandwidth low-latency interprocessor communication. The key components of interconnection networks for large multiprocessors and multicomputers are $n \times n$ switches [Crow85, Dall86, Dall88, Gott83, Pfis85a, Seit85, Whit85]. An efficient internal micro architecture for these switches is essential for achieving high-performance communication with cost-effective implementations.

Figure 1.1 and Figure 1.2 demonstrate the usage of communication switches in different interconnection networks [Feng81]. Figure 1.1 shows an indirect network for multiprocessors. In an indirect network, processors are connected to memory modules through multiple stages of communication switches. Packet transmission between processors and memory modules is provided by these communication switches. Figure 1.2 shows a direct network for multicomputers. In a direct network, a point-to-point net is constructed from many communication switches. Each node, which includes a processor and a memory module, can transmit packets to any other node over the network. Basically the architecture of the communication switches for both indirect and direct networks are the same.

The function of a communication switch is to receive packets arriving at its input ports and route them to the appropriate output ports. The bandwidth of the input ports is typically equal to the bandwidth of the output ports. If two or more packets destined to the same output port arrive at the input ports of the switch...
simultaneously, only one can be forwarded and the other packets are blocked. In an unbuffered network, these blocked packets must be discarded or misrouted. In a buffered network, where buffer storage is associated with each switch, the blocked
packets are stored in the switch until their destination output ports are free [Dias81b]. By adding buffer storage to each switch, the maximum throughput of the network is increased [Dias81b].

In general, a communication switch consists of input ports, output ports, an $n \times n$ crossbar, and some buffer memory [Dias81b, Kuma84, Reed87]. The organization of a typical communication switch is shown in Figure 1.3. Depending on the traffic patterns, there may be conflicting demands for these resources. Arbiter that resolve the conflicts and provide efficient and fair scheduling of these resources are critical for achieving the maximum possible performance from a given network.

![Buffers Crossbar](image)

**Figure 1.3:** Organization of a typical communication switch

In communication switches where packets are processed at each input in first-in-first-out (FIFO) order, arbitration of the internal crossbar is relatively simple. Following arbitration, each input port may be either active (transmitting) or idle (blocked). For each output port, there is a choice of which of several contending input ports will connect to it. Hence, for each output port, there is a
simple independent arbiter that selects one of the inputs requesting that output and blocks the others [Bhuy87, Dall86].

Recent communication switch designs maximize performance by allowing packets at an input port to be processed in non-FIFO order [Dall90, Kuma84, McMi80, Tami88a]. Specifically, packets at an input port, destined to different output ports, may be transmitted through the switch in any order. Each input port contends for multiple output ports but needs only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. Since the arbitration result for each port is dependent on the arbitration for other ports, the arbitration task is more complex than for switches with FIFO input ports and the task cannot be performed by separate independent arbiters at the input ports or output ports.

In order to realize the potential for high performance of non-FIFO processing of packets, the crossbar arbitration should result in switch configurations that allow for the maximum number of packets to be transmitted simultaneously. The arbitration process itself must be fast relative to the rate at which packets are received and relative to the latency of packet transmission through the switch. This dissertation deals with the design and VLSI implementation of crossbar arbiters that meet these requirements. This work is part of the UCLA ComCoBB (Communication Coprocessor Building-Block) project, whose focus is the design and implementation of a high-performance communication coprocessor for VLSI multicomputers. The investigation of the crossbar arbiter designs in this dissertation was motivated by the fact that our design of a communication coprocessor involves the use of non-FIFO input buffers [Tami88a].
1.1. Crossbar Arbitration

While many different internal organizations of buffered communication switches are possible, critical implementation considerations lead to the use of buffers at the input ports rather than central buffer pools or buffers at the output ports [Tami88a]. Figure 1.4 shows a switch organization with traditional FIFO (first-in-first-out) input port buffers. A key component in these switches is a crossbar, which allows arbitrary permutations in connecting the input buffers to output ports. The crossbar consists of $n$ horizontal buses (rows) and $n$ vertical buses (columns). Each horizontal bus is connected to an input buffer, while each vertical bus is connected to an output port. A horizontal bus intersects a vertical bus at a crosspoint. At each crosspoint there is a switch, which may be closed to form a connection between the corresponding input buffer and output port.

![Crossbar Arbitration Diagram](image)

**Figure 1.4:** A switch with FIFO input buffers.

In communication switches with FIFO buffers, there is at most one packet at each buffer that is ready for transmission through the crossbar. Based on the
destination of the packet at the head of its FIFO queue, each non-empty input buffer contends for the internal bus connected to one of the output ports. Access to each internal output port bus can thus be arbitrated independently of arbitration of access to other internal buses. Hence, a crossbar arbiter for an $n \times n$ FIFO switch (i.e., a switch with FIFO input port buffers) can be constructed out of $n$ independent conventional bus arbiters [Bhuy87, Dall86].

An $n \times n$ communication switch with FIFO buffers does not utilize the resources of the switch efficiently. Specifically, the problem with such a switch is that packets may be blocked unnecessarily — if the packet at the head of the queue is blocked, all other packets in the buffer, even those destined to idle output ports, are also blocked [Tami88a]. In order to improve output port utilization, and thus increase the throughput of the switch, packets can be segregated according to the output port to which they have been routed. This can be done using separate FIFO buffers for each of the output ports at each of the input ports [McMi80, Tami88a]. Figure 1.5 shows a switch organization with this kind of multi-queue input buffers. In order to better utilize the available buffer space, the input buffers can be dynamically partitioned between the different queues. Such a dynamically-allocated, multi-queue (DAMQ) buffer has been designed, and has been shown to significantly increase network throughput [Fraz89, Tami88a]. A switch organization with DAMQ input buffers is shown in Figure 1.6.

DAMQ buffers and other multi-queue buffers allow more than one request to the crossbar arbiter, since each nonempty queue can assert a request. Hence there is the opportunity to connect more crosspoints of the crossbar than with FIFO buffers, thus leading to higher throughput and lower latency. In this dissertation
we address the various issues involved in the design and VLSI implementation of the crossbar arbiters in communication switches with multi-queue input buffers. The performance evaluation based on simulation in the following chapters is for communication switches with DAMQ buffers only. However, the arbiter design
can also be applied to communication switches with other multi-queue buffer organizations.

1.2. Organization of the Dissertation

The next chapter surveys the previous work related to this research. In general, different applications lead to different arbitration requirements. A brief overview of various arbitration problems in computer systems is covered. The design of the different arbiters investigated by researchers and system developers are summarized. We then describe several experimental communication switch designs proposed by other researchers. The other design issues studied in the ComCoBB project are also discussed in this chapter.

In Chapter 3 we discuss the arbitration problem for communication switches with multi-queue input buffers. Several practical arbiter designs are proposed. Their functions are described and their estimated arbitration speeds are compared. Simple probabilistic analysis is used to provide an initial indication of the expected throughput for single switches. More complete performance study for different arbiter designs is based on event-driven simulations of individual switches and of multi-stage interconnection networks. The impact of buffer size and switch size is also studied. The VLSI implementation of efficient arbiters is then described. Circuit simulation is used to evaluate the circuit performance of the VLSI implementation.

In general, larger crossbar switches help lower the probability of conflicts and allow packets to traverse the network in fewer hops. However, increasing the size of the crossbar also increases the delay of the arbiter used to resolve conflicting
requests. Chapter 4 proposes and evaluates a *decomposition* technique used to improve the performance of large crossbar switches in asynchronous networks. The latency for a packet traversing an unloaded network is analyzed for different arbitration schemes and different switch sizes. The proposed technique is evaluated based on event-driven simulations for networks with different switch sizes under various traffic loads. The impact of reducing the number of queues per input buffer on performance and its implementation efficiency is also discussed.

While our proposed arbiter designs efficiently resolve conflicting requests in switches with multi-queue input buffers, they do not guarantee *fairness*. It is possible for an “unlucky” packet to be left in a switch buffer for an inordinate time. Chapter 5 describes such a *starvation* problem with focus on communication switches in asynchronous networks. Techniques for starvation prevention are proposed. Their performance impact is evaluated based on event-driven simulations for uniform traffic and nonuniform traffic. A VLSI implementation of a starvation-free arbiter is compared with that of an arbiter without the starvation prevention mechanism to measure the cost of adding such a mechanism.

In Chapter 6 we describe how the arbiter designs proposed in the preceding chapters can be modified and applied to support other types of network traffic. In a communication switch with adaptive routing capability, a packet can have more than one destined output. Packets arriving at the inputs can adapt to the congested or faulty area in the network and are routed to the appropriate outputs to avoid the problem network area. We describe how our symmetric crossbar arbiters can be applied to the arbiter design for adaptive routing switches with conventional FIFO input buffers. Another issue discussed is the design of crossbar arbiters for
supporting high-priority traffic. Arbiter designs supporting two levels of packet priorities are described.
Chapter Two
Previous Work

In this chapter, research related to this work is surveyed. Section 2.1 describes several arbitration problems discovered in various applications. In many computer systems, resources (e.g. buses) are shared among users (e.g. processors). Conflicts occur when users compete for these limited resources at the same time. We describe previous research on the arbitration schemes which are used to resolve the conflicts that appear in different applications. In Section 2.2, we give an overview of the related work on buffered communication switch design. The effect of adding buffers in the communication switch on the network performance is discussed. Several experimental switch designs proposed recently by researchers are summarized. Section 2.2 describes other design issues in our ComCoBB project.

2.1. Arbitration Problems in Computer Systems

Several arbitration mechanisms used in different computer applications are presented in this section. Section 2.1.1 describes the most common arbitration problem — conventional bus arbitration. In conventional bus arbitration, more than one processor competes for a single bus, and only one processor can get the bus grant at a time. Section 2.1.2 describes multiple bus arbitration where two or more buses are used to connect processors and memory modules. In this application, more than one processor competes for a pool of resources and these resources are interchangeable. Another application of arbitration is found in the
segmented ring bus architecture, which is presented in Section 2.1.3. Similar to multiple bus arbitration, segmented ring bus arbitration has also more than one user and more than one resource. However, unlike the multiple bus arbitration, the resources are not interchangeable in segmented ring bus arbitration.

2.1.1. Conventional Bus Arbitration

In many multiprocessor systems, processors and memory modules are connected by a shared bus. Figure 2.1 shows the configuration of a typical shared-bus multiprocessor. Since there is only one bus, a bus arbiter is used to arbitrate the memory access requests issued by processors. This conventional bus arbiter receives up to $n$ requests for use of the bus and grants one of these requests. Figure 2.2 shows a user-resource model of shared bus arbitration. There are $n$ users (processors) and one resource (shared bus) in the model. The design of such $n$-user 1-server (or 1-of-$n$) arbiters has been investigated by many researchers and system developers [Gust84, Taub84].

![Figure 2.1: A shared-bus multiprocessor.](image)

One of the most important issues regarding arbiter design is the selection policy for the arbitration. In most systems, there is a FIFO queue at each processor
to buffer the requests to the shared bus. The selection policy of the arbiter decides the priority of the requests at the heads of the FIFO queues. Fixed priority arbiters are relatively simple and fast, but they have the disadvantages that they are not fair; lower priority processors can be forced to wait indefinitely if higher priority processors keep the memory busy. This fairness problem can be solved by giving priority to the processors on a rotating basis, with the lowest priority given to the processor that most recently used the bus [Pear75].

Priority rotation can be implemented in a ring-structured arbiter. However, the arbitration time grows linearly with the number of processors. If we arrange the arbiter as a binary tree of depth $\log_2 N$ constructed from 1-of-2 arbiter modules, the arbitration time grows as $O(\log_2 N)$ instead of $O(N)$. Fairness can be assured by placing a flip-flop in each 1-of-2 arbiter module, toggled automatically to alternate priorities when the arbiter module receives simultaneous

*Figure 2.2:* A user-resource model of shared bus arbitration.
The arbiter for shared-bus multiprocessors can be applied to the design of crossbar switches. In an $n \times n$ crossbar switch, $n$ inputs are connected to $n$ outputs through an $n \times n$ crossbar. With FIFO input buffers, there is at most one packet at each buffer that is ready for transmission through the crossbar. Based on the destination of the packet at the head of its FIFO queue, each non-empty input buffer contends for the internal bus connected to one of the output ports. Access to each internal output port bus can thus be arbitrated independently of arbitration of access to other internal buses. Hence, a crossbar arbiter for an $n \times n$ FIFO switch (i.e., a switch with FIFO input port buffers) can be constructed out of $n$ independent conventional bus arbiters. Such crossbar arbitration with $n$ 1-of-$n$ arbiters is used in the Torus Routing Chip [Dall86], where there is a fixed priority order of the inputs to each of the arbiters. Bhuyan [Bhuy87] describes crossbar arbitration where there is an attempt to maintain fairness by rotating the priority order in each 1-of-$n$ arbiter such that the top priority is given to the user following the one who was last serviced.

2.1.2. Multiple Bus Arbitration

In order to increase the bandwidth between processors and memory modules, systems with multiple buses have been proposed [Lang82a, Mudg87]. In these systems $n$ processors are connected to $m$ memory modules through $b$ buses. Figure 2.3 shows the configuration of a multiple-bus multiprocessor. In order to complete, a request from a processor must win the arbitration for two types of resources: a memory module and a bus. Since each processor requests one
particular memory module and a memory module can handle only one request at a time, this arbitration can be performed by $m$ independent conventional 1-of-$n$ arbiters [Bhuy87, Lang82b]. All the memory modules for which there are requests must then contend for the available $b$ buses. Figure 2.4 shows a user-resource model of multiple bus arbitration. Since a memory module can use any of the $b$ buses, a different type of arbiter must be used.

![Multiple buses](image)

**Figure 2.3:** A multiple-bus multiprocessor.

Lang and Valero [Lang82b] describe the design and implementation of such an $m$-user $b$-server ($b$-of-$m$) arbiter. The basic arbiter consists of an iterative ring of $m$ arbiter modules that compute the bus assignments, and a state register to store the arbiter state after each arbitration. The storage of the state is used to make the arbiter fair by taking into account previous bus assignments. After each arbitration, the highest priority is given to the module immediately following the last one serviced.
Section 2.1.3. Segmented Ring Bus Arbitration

In the Concert multiprocessor system [Hals86], a segmented ring bus is used to interconnect processor clusters. Figure 2.5 shows a multiprocessor based on the segmented ring bus. Each one of the \( n \) processor clusters (users) can request up to \( n/2 \) bus segments (servers) in order to complete an access. Since each bus segment can be used by only one requests at a time, requests from different processor clusters can conflict and arbitration is needed. Each cluster request is translated into a request for a particular set of bus segments. If any one of the requested bus segments is busy, the cluster request cannot be serviced. If a cluster request is denied because one of its required bus segments is busy, any other bus segments already “given” to the cluster become free for use by another cluster. Hence, for maximum utilization of the ring bus, the arbitration for each bus segment cannot be performed independently of the arbitration of other bus segments. Thus,
conventional 1-of-n arbiters cannot be used. The required arbitration is also different from the $b$-of-$m$ arbitration used for multiple-bus systems since the resources (bus segments) are not interchangeable and a request may require more than one resource. Figure 2.6 shows a user-resource model of segmented ring bus arbitration.

Figure 2.5: A multiprocessor based on the segmented ring bus.

In the Concert multiprocessor, priority rotation is used in the arbiter for the priority selection policy [Hals86]. For each arbitration cycle, the Concert arbiter loops through the requests in order of decreasing priority and make decisions as to whether or not to grant each request. The arbiter grants a request if it does not conflict with either a request already granted or an ungranted higher priority request. Since a request cannot be granted if it conflicts with an ungranted higher

17
priority request, it is guaranteed that no request can be locked out forever with the priority rotating scheme. Hence, fairness is ensured.

2.2. Buffered Communication Switches

Communication switches are used to construct interconnection networks for supporting interprocessor communication in multiprocessors and multicomputers. Many researchers have studied various design issues of communication switch architectures. It has been shown that adding buffers in communication switches generally can increase the throughput of the interconnection network [Dias81a, Dias81b]. Hence, buffers are often included in many communication switch designs. This section surveys previous research on the design of buffered communication switches. Section 2.2.1 describes the related work on adding buffers in an interconnection network. The performance impact of different buffer
organizations is discussed. In Section 2.2.2, we describe several designs of communication switches proposed by other researchers. The important features of each design are introduced.

2.2.1. Performance Impact of Switch Buffers

In an interconnection network, two or more packets arriving in a communication switch may be destined to the same output. When the output conflict occurs, only one packet can be forwarded to the next switch through this output. In an unbuffered switch, packets which cannot be forwarded immediately have to be dropped or misrouted. Patel [Pate79] analyzed the performance of unbuffered interconnection networks. Dias and Jump [Dias81a, Dias81b] extended Patel’s results and investigated how addition of buffers to the links between stages affects the performance of an interconnection network by analysis and simulations. The performance of buffered interconnection networks is characterized by their throughput (number of packets transferred through the network per unit time) and the latency (average time spent by the packet in the network). The insertion of buffers between the stages of an interconnection network considerably improves its throughput. However, if the buffer size (the capacity of the buffer) is large, the average latency for a packet traversing the network is also significantly increased when the traffic load is high.

Kumar and Jump [Kuma84] investigated the effect of two different buffer designs on the performance of communication switches. One design is that there is one buffer for each input (Figure 2.7-a). Hence, there are \( n \) buffers in an \( n \times n \) switch with this approach. The other design is that there is one buffer for each
crosspoint. (Figure 2.7-b). That is, for each input, the arriving packets destined to the same output are placed in the same buffer. Hence, there are \( n^2 \) buffers in an \( n \times n \) switch with this approach. Kumar and Jump compared the network performance of the communication switches for these two designs with the same total buffer capacity in the network. It was observed that unless the buffer size is very small, the design with one buffer for each crosspoint achieves superior performance, providing significantly higher throughput. When the buffer size is very small (e.g. only one packet can be accommodated in each buffer), buffer blocking limits the throughput for the design with one buffer for each crosspoint. However, when the buffer size is sufficiently large (e.g. two or more packets can be accommodated in each buffer), this buffer design utilizes the output links more efficiently.

![Diagram](image)

a) One buffer for each input.  
b) One buffer for each crosspoint.

**Figure 2.7:** Two different buffer designs.
2.2.2. Communication Switch Designs

There have been a few experimental switch designs proposed by researchers in the past. Gottlieb et al. described a switch design for the multistage interconnection network used in their NYU Ultracomputer [Gott83]. The switch is a $2 \times 2$ bidirectional routing element, which can transmit a message from the input port where the message arrives to the appropriate output port. An arriving message in the switch is routed based on a single bit in the destination address. A queue is associated with each output port, where messages waiting for transmission to the adjacent stage are stored. The distinguishing feature of the switch is the capability of combining messages. Memory requests are messages that are sent from the processors to memory modules by traversing several stages of switches. When concurrent memory requests are directed at the same memory location and meet at a switch, they can be combined. Combining requests reduces communication traffic and thus decreases the lengths of the queues, leading to lower network latency.

Stevens, Robison, and Davis proposed a switch, called the Post Office, designed for their multicomputer system with hexagonal meshes topology [Stev86]. The switch is coupled with a processing element to form a six-ported node. The Post Office uses virtual cut-through for the switching mechanism, in which the packet header can be examined as soon as it arrives to determine the output. If the destined output is free, the packet can be transmitted to the next switch immediately without waiting for the whole packet arrives. All ports in a Post Office share the same buffer pool from which packets are retrieved for transmission and stored on arrival. This shared buffer organization minimizes the
number of internal buses, wires and controllers, and reduces the total buffer size needed in the switch owing to resource sharing. However, it also results in performance degradation because of the bottleneck of buffer contention. The Post Office supports adaptive routing. The packet buffer controller in the switch determines when packets are stagnating in the buffers. When this occurs, a routing mechanism which dynamically re-routes packets via alternative links is invoked. The buffer controller also contains the deadlock avoidance mechanism.

Dally and Seitz designed a VLSI switch chip, called the torus routing chip (TRC), which is used in the $k$-ary $n$-cube networks for multicomputers. The TRC provides four bidirectional ports connected to its neighboring routing chips, and one bidirectional port connected to its local processing node. The TRC is entirely self-timed, thus permitting each processing node to operate at its own rate with no need for global synchronization. Similar to the Post Office, the TRC uses virtual cut-through to reduce the latency for packets traversing the network. Dimension-order routing is used in the TRC, in which the network routes packets first in the X direction, then in the Y direction. Packets are routed in the direction of decreasing address, decrementing the relative address at each switch. When the relative X or Y address is decremented to zero, the packet has reached the correct X or Y coordinate. The TRC uses virtual channels to perform deadlock-free routing in torus networks. By splitting each physical channel into two virtual channels and making routing dependent on the virtual channel on which a packet arrives, the TRC breaks the cycle of channel dependencies. Both virtual channels share a physical communication channel. However, a queue is associated with each virtual channel.
Fujimoto described the architecture of a VLSI communication component proposed for multicomputer networks [Reed87, Fuji83]. In the design, *virtual circuits* is used for the transport mechanism. Each physical link is divided into some fixed number of unidirectional *virtual channels*. The bandwidth is allocated to channels when they have data to send, such that the bandwidth can be efficiently utilized. A translation table is associated with each input port to generate the outgoing link and the channel number. This design employs lookup tables to implement the routing scheme, thus the switch can be used in irregular networks as well as regular networks. To reduce the size of the routing table, hierarchical names are used and there are multiple routing tables per node. There is a single buffer pool shared by all channels of the switch. To avoid performance bottleneck of accessing the buffer by several ports, the buffer consists of 16 memory modules and is interleaved by 16 ways. Strategies that limit buffer usage by individual circuits are used to avoid buffer hogging.

Joerg presented a communication switch design, called the *Packet Switched Routing Chip* (PaRC), which is used as the building block for the indirect $n$-cube network in the Monsoon multiprocessor dataflow computer [Joer87]. The PaRC is a 4×4 VLSI routing element implemented in a VLSI gate array chip. Instead of a FIFO buffer at each input port, there are four separate buffers there. The capacity of a buffer is for accommodating one packet, and a packet destined to any output port can be stored in any buffer. Through a 16×4 crossbar, each output port can read from any of the 16 buffers at the input ports. The reason for this buffer arrangement is to maximize the throughput of the switch. If only a large FIFO buffer is used at each input port, the blocked front packet in the queue can prohibit
the packets behind it from transmitting to free destined output ports. Because of this buffer organization, the PaRC has a scheduling circuit at each output port to guarantee that the order of the packets is preserved if these packets have the same input and the same output. As a result, packets sent from the same source node to the same destination node in the network arrive in order. A facility for sending circuit switch packets has also been included. This allows a processor to ensure that its message has been received before it continues.

Hsu and Banerjee described their routing controller, called virtual channel router (VCR), used in the hypercube or other multicomputer networks [Hsu90]. The VCR is a switch designed for circuit switched networks with blocking when contention occurs. Hsu and Banerjee observed that the communications in multicomputer networks exhibit high locality in both message destination and length. The switch can keep the circuit (from source to destination) connected even after the message transmission is completed in order to exploit the locality of communication. This reserved circuit is called the cached circuit. When a new message generated at a node has the same destination as the last message, this new message can be sent immediately along the cached circuit, thus saving the circuit set-up overhead. In case of contention, cached circuits can be torn down for other new messages. To improve the connectivity, each physical link in the VCR is divided into two virtual channels.

Konstantinidou and Snyder proposed the chaos router, a self-timed adaptive routing switch, for the k-ary n-cube multicomputer networks [Kons90a]. The chaos router is an adaptive router which can misroute messages. That is, messages can be sent further from their destination in the presence of congestion. There is a
shared queue in the chaos router, which is used to store messages waiting for transmission. The hardware algorithm implemented in the router can randomly select a message in the queue to misroute. A hardware circuit in the router provides the source of randomness for selecting the message in the queue. The chaos router operates fully asynchronously, and the whole router is a self-timed circuit. The hardware algorithm implemented in the chaos router guarantees no message is stalled forever (deadlock freedom). It is also shown that every message injected into the network is eventually delivered (livelock freedom) and every node is able to inject a message eventually (starvation freedom).

2.3. The ComCoBB Project

Small \( n \times n \) switches are the key components of the interconnection network used in large-scale multiprocessors and multicomputers. The architecture of these small switches is critical for achieving high-bandwidth low-latency interprocessor communication for multiprocessors and multicomputers. The UCLA ComCoBB project is aimed at investigating the design and implementation issues of VLSI communication coprocessors for multicomputer networks. In this section, a survey of design issues which have been studied by the ComCoBB project is presented. In Section 2.3.1, various buffer organizations are compared based on their performance. Among them a particular buffer design is shown to achieve higher performance than the other practical designs. Section 2.3.2 discusses switch designs supporting high-priority traffic. It is shown that the buffer organization suggested in Section 2.3.1 can be modified to provide superior support for high-priority traffic at a relatively low cost compared to other options. In Section 2.3.3,
a message transport mechanism is introduced. This message transport mechanism can be used to efficiently support adaptive routing.

2.3.1. High-Performance Multi-Queue Buffers

It has been shown that adding buffers in communication switches can significantly improve the performance of interconnection networks [Dias81b, Dias81a]. In [Tami92a], alternative buffer designs are studied for small \( n \times n \) switches. The goals are to maximize the network throughput and minimize the average latency for the packets traversing the network. Two key design decisions are considered: the location of the buffers and the organization of the buffers.

There are three approaches regarding the location of the buffers in the switch: a central buffer pool, buffers at the input ports, and buffers at the output ports. A central buffer pool provides complete sharing of the available buffer space by all communication ports in a switch, thus can achieve excellent storage utilization. However, there are fundamental difficulties in implementing a high-performance switch with a central buffer pool. Specifically, the buffer will be slow and consume a great deal of chip area, if it needs to support simultaneous reads and writes from several input and output ports. Similar to the problems with the central buffer pool, the option of placing buffers at the output ports is also undesirable, because it also needs to support simultaneous writes from several input ports. These considerations lead to choosing the option of placing the buffers at the input ports, which can achieve high performance with reasonable implementation cost in switch design.

Given that buffers are placed at the input ports, there are two design choices
for the buffer organization: FIFO buffers and non-FIFO buffers. In most switch designs, the input port buffers are organized as FIFO queues. However, as discussed in Section 1.1, the performance of FIFO input buffers is limited because of unnecessary blocking. Specifically, when the packet at the head of the queue is blocked, all other packets in the buffer, even those destined to idle output ports, are also blocked. This poor performance of FIFO input buffers can be avoided by allowing handling of packets in non-FIFO order. Three different non-FIFO buffers are studied: statically allocated multi-queue (SAMQ) buffers, statically allocated fully connected (SAFC) buffers, and dynamically allocated multi-queue (DAMQ) buffers. Figure 2.8 shows the organizations of the FIFO buffers and the three non-FIFO buffers.

In an $n \times n$ switch with SAMQ buffers or SAFC buffers, the storage allocated for each input port is statically partitioned into $n$ queues, and each queue stores the packets destined to the same output port. SAMQ buffers and SAFC buffers provide non-FIFO order of handling packets, because now the packets at the head of any one of the $n$ queues can be accessed. Hence, when a packet at a particular input port is blocked, it is still possible to transmit another packet from the same input port to a free different output port. The difference between SAMQ buffers and SAFC buffers is that SAFC buffers allow more than one packet to be transmitted to the outputs for each input port, while SAMQ buffers allow only one for each input port.

A potential performance problem with SAMQ and SAFC buffers is that the storage space in the whole switch is statically partitioned into $n^2$ queues. When the available storage space is limited in the switch, each queue can be too small to be
effective. In [Tami88a, Tami92a, Fraz89], the DAMQ buffer is proposed. In this buffer, the storage at each input port is partitioned into \( n \) queues \textit{dynamically}, one for each output port. The DAMQ buffer at an input port can achieve better utilization of the buffer space owing to the effect of resource sharing, while providing non-FIFO order of handling packets. The DAMQ buffer can be managed by using hardware linked lists. The control of the linked lists is provided by a dedicated finite state machine. This additional hardware requires about the
same chip area as 32 bytes of storage in the buffer, which is equal to the buffer space to accommodate a packet in the implementation of the ComCoBB switch.

Simulations for a 64×64 Omega network constructed from 4×4 switches have been conducted for the performance comparison of the four different buffer designs. It is shown that the DAMQ buffer achieve superior performance to the other alternatives. For uniform traffic, the DAMQ buffer results in significantly lower latency and higher maximum throughput than the others with the same total buffer storage capacity. For instance, with four packet slots per input port, the networks with FIFO, SAMQ, and SAFC buffers saturate at approximately 0.50. However, the network with DAMQ buffers can achieve a throughput of 0.71. Even the additional chip area is accounted for the more complex control, DAMQ buffers still considerably outperforms the others.

2.3.2. Support for High-Priority Traffic

Large-scale multiprocessors and multicomputers can be used for real-time applications. For these systems, it is critical that high-priority traffic generated by such as I/O devices or system-wide exception handling be transmitted through the network with low latency. To meet this requirement, communication switches used in the interconnection network may have to provide special support for such high-priority traffic. In [Tami88b, Tami92b], communication switches supporting high-priority traffic are discussed and evaluated. The focus is on the design of the input port buffers in the communication switch. Simulations for a 64×64 Omega network are conducted to evaluate the various designs. As a measure of “worst-case” performance, the 99th percentile latency is used. The 99th percentile latency
is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies) from the network.

From the network simulations, the 99th percentile latency may be more than twice the average latency, and the situation deteriorates when the network load increases. This phenomenon does not change with the different buffer organizations described in Section 2.3.1. Hence, systems for real-time applications have to identify the packets in need of fast service and provide special support for these high-priority packets. Four approaches to supporting high-priority packets in a communication switch are discussed in [Tami88b, Tami92b]:

A. **Preferential Arbitration for High-Priority Traffic**

The crossbar arbiter is modified in such a way that the requests from the queues which have a high-priority packet at the head are given higher priority. At each cycle, the crossbar arbiter attempts to connect queues with high-priority packets at their heads to the destined output ports first, and then connect queues with normal packets to any remaining available output ports. From the simulation results, it shows that this approach does not provide sufficient support for the high-priority traffic. The 99th percentile latency of the high-priority packets is not reduced to the level of average normal packet latency, even with moderate network load. This conclusion applies to both FIFO and non-FIFO buffers described in Section 2.3.1.

B. **Dedicated Queues for High-Priority Traffic**

The buffer organization is modified in this approach. At each input port, an additional queue is dedicated to high-priority packets. Since there is a dedicated queue at each input port, it will not happen that the high-priority packets are
blocked behind the normal packets as in the preferential arbitration case. Hence, the performance for high-priority packets is greatly improved. The simulation results indicate that as long as the proportion of high-priority traffic is low, dedicated queues for high-priority traffic reduce the 99th percentile latency of the high-priority packets to the level of the average latency for normal traffic under a wide range of network throughputs. The DAMQ buffer is particularly amenable to this approach in terms of implementation since no additional buffer is required. Furthermore, the modified DAMQ buffers achieve better performance than the other modified non-FIFO buffers. Hence, only designs based on DAMQ buffers are considered in the following.

C. Multiple Dedicated Queues for High-Priority Traffic

In this approach, at each input port there is a dedicated queue for the high-priority packets destined to each output port. With a DAMQ buffer at each input port, there are \(2n\) queues: \(n\) queues for normal packets and \(n\) queues for high-priority packets. The simulations results show that when a large proportion of network traffic is high-priority and the network is heavily loaded, this approach performs better than the approach of using dedicated queues for high-priority packets described in the above. However, this situation is less likely, and the performance advantage is small while the associated implementation cost is high.

D. Dedicated Buffers for High-Priority Traffic

In this approach, there are two independent buffers at each input port: one for normal packets and one for high-priority packets. The buffer for normal packets is organized as a DAMQ buffer. The buffer for high-priority packets can be either a
FIFO queue or a DAMQ queue. The simulation results show that dedicated buffers for the high-priority traffic can provide the same performance advantage as dedicated queues in shared DAMQ buffers. However, the implementation cost for dedicated buffers is much higher than for dedicated queues in shared DAMQ buffers. Thus, the approach of using dedicated buffers for high-priority traffic is not cost effective in comparison.

From the above discussion, the approach of using dedicated queue for high-priority traffic is clearly the choice. It provides superior support for high-priority traffic while the implementation cost is relatively low.

2.3.3. Dynamic Virtual Circuits

Adaptive routing can be used to improve network performance and provide fault tolerance in multicomputer systems [Gall77, Bell86, Ngai89, Kim88, Chie86, Chow88, Chen90a, Chen90b, Kons90b, Kons90a, Kons91, Dall93]. With adaptive routing, multicomputers achieves better link utilization and fault tolerance by routing packets around congested or faulty areas in the network. In [Tami91], a message transport mechanism which can adapt to congested or faulty network areas is proposed. This message transport mechanism is called dynamic virtual circuits.

Traditional virtual circuits scheme is a message transport mechanism which combines the desirable features from both circuit switching and packet switching [Bert87, Reed87]. With conventional virtual circuits, a logical path is set up from the source node to the destination node before the communication starts. Packets are then sent through pre-established logical paths, thus minimizing packet
routing time and overhead. On each virtual circuit, packets arrive in the destination node in first-in-first-out (FIFO) order. To improve the utilization of the links in the network, each physical link along the path is logically divided into multiple virtual channels and time-shared among them. Virtual circuits are paths which consist of a sequence of virtual channels.

To establish a virtual circuit, the source node sends a circuit establishment packet (CEP) through the network to the destination node. In the intermediate nodes, the CEP arrives on an unused input channel of an input port and then is routed to a chosen output port according to its destination address, using one of the unused channels of the output port. The mapping table is also set to route future packets arriving on the same input channel to the chosen output channel. After the circuit is established, the source node can then transmit packets along the way to the destination node. Once the circuit is no longer needed, a packet called circuit destruction packet (CDP) is sent by the source node to delete the circuit from the network. Figure 2.9 shows an established virtual circuit. A mapping table is associated with each input port. The packets arriving in the input port uses the input channel number as the index of the table and extracts the destined output port and the output channel number.

The problem with conventional virtual circuits is that they cannot be changed in response to congestion or failure since the paths are static. Dynamic virtual circuits proposed in [Tami91] can overcome this problem by allowing existing circuits to be torn down on demand. With dynamic virtual circuits, the establishment of a new virtual circuit is guaranteed to succeed even when there are no free virtual channels on a desired link. When a CEP arrives in a node where all
Figure 2.9: A virtual circuit from process A to process B. The circuit uses channel 3 of the first link and channel 2 of the second. PIF is the processor interface.

the channels of the needed link have been allocated, a victim channel is selected and the circuit is disestablished from the intermediate node. The decision is made
locally at this intermediate node. The part of the circuit from the source node to the destination node remains intact so that the circuit can be reestablished later.

Tearing down a circuit temporarily is achieved by sending a CDP along the path from the intermediate node. After the victim channel is chosen and deallocated, the node generates a CDP and sends it along the victim channel. Then the CEP can be sent through the output port, continuing to establish the new circuit. The generated CDP is marked *terminal* so that the destination node knows that the circuit is torn down temporarily from an intermediate node, and it will be reestablished later.

To reestablish the cut circuit, the information regarding the ultimate destination node has to be provided. This information is kept at each intermediate node when the circuit is established originally. The reestablishment of the cut circuit is triggered by its next arriving packet at the node where the circuit is previously cut. The node chooses an output port, creates a CEP, updates the mapping tables, and sends the CEP. The arriving packet and the following ones on the same circuit can then be sent along the new path.

Since a circuit can be torn down temporarily and reestablished later, physical FIFO packet arrival at the destination node is not guaranteed. It is possible that the reestablishment CEP follows a different path and arrives at the destination node before the CDP generated at the intermediate node arrives. As a result, the packets on the reestablished branch can arrive before the packets on the torn-down branch. To ensure proper packet order, a mechanism based on logical timestamps can be employed. The mechanism achieves proper packet order by providing a kind of logical timestamps in CEPs and CDPs so that the destination node can identify and
order the arriving branches from the same circuit. In each node, a count of the number of nonterminal circuit destructions the node has initiated is maintained. This count should be large enough and can thus serve as a logical timestamp on the node. A circuit destruction event can be uniquely identified by combining the identifier of the node and the timestamp. When the node sends out the reestablishment CEP, the node identifier and the stored destruction timestamp are included in this packet. At the destination node, the reestablishment CEP is matched with the corresponding CDP by checking the values of the node identifier and the timestamp.

A multicomputer node supporting dynamic virtual circuits consists of an application processor, local memory associated with the application processor, a crossbar switch, a special routing processor with its memory. Figure 2.10 shows the organization of such a multicomputer node. The routing processor is a general-purpose processor which is used as a dedicated controller to perform some of the infrequent but complex operations that are needed to support dynamic virtual circuits. These infrequent operations include initiating circuit destruction, reestablishing a circuit, updating of global routing tables, and resolution of deadlocks. Those frequent operations such as routing and forwarding of a packet on an established circuit are handled by the crossbar switch.
Figure 2.10: A multicomputer node.
Chapter Three
Symmetric Crossbar Arbiters

Communication switches are basic components of interconnection networks for multiprocessors and multicomputers. The traffic through the switches is often delayed due to conflicting demands for resources, such as buffer space or output ports. Hence, switches must include arbiters that resolve conflicting resource demands. Efficient design and implementation of these arbiters is critical for maximizing network performance.

In order to maximize performance, recent communication switch designs allow packets at an input port, destined to different output ports, to be transmitted through the switch in any order. Each input port contends for multiple output ports but needs only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. The arbitration task is thus symmetrical with respect to inputs and outputs. This chapter discusses the design and implementation of symmetric crossbar arbiters for VLSI communication switches.

In the next section, symmetric crossbar arbitration is defined and its use in modern communication switches is explained. Section 3.2 presents several possible symmetric crossbar arbitration schemes. Basic implementation considerations are discussed, and the operation of three practical symmetric crossbar arbiter designs is described. The performance of several symmetric crossbar arbitration schemes is evaluated in Section 3.3. The performance advantage of using an efficient arbitration policy is indicated using probabilistic
analysis of single switches. More accurate and complete evaluation is based on event-driven simulations of individual switches and of multi-stage interconnection networks. The VLSI implementation of the best practical arbitration schemes is described in Section 3.4. Logic and circuit design is described, and circuit simulation is used to determine the performance of the proposed arbiters.

3.1. Symmetric Crossbar Arbitration

Crossbar switches often include FIFO buffers at their input ports for storing incoming packets that cannot be forwarded immediately due to output port contention or blocking [Dias81b]. The strict FIFO order of handling packets at each input port unnecessarily reduces the throughput of the switch [Tami88a, Tami92]. When the packet at the head of the queue is blocked, all other packets in the same buffer are also blocked, even if they are destined to idle output ports.

Multi-queue buffers avoid the shortcomings of FIFO buffers by partitioning each input buffer into several queues. The packets at the head of the queues in a buffer may be accessed in any order. Even if one of the queues of the buffer is blocked, it may be possible to transmit a packet from the head of another queue, which is destined to a different output port [McMi80, Kuma84, Tami88a, Tami92, Dall90]. A possible organization of a multi-queue buffer is to maintain one FIFO queue for each output port. In order to utilize buffer storage efficiently, it is desirable for all the queues to share common storage. A multi-queue buffer with this organization is called a dynamically-allocated multi-queue (DAMQ) buffer [Tami88a, Tami92]. Figure 3.1 shows the switch organizations with FIFO input buffers and DAMQ input buffers.
An $n \times n$ crossbar and its associated arbiter is shown in Figure 3.2. The inputs to the arbiter consist of $n^2$ request lines, one per crosspoint, and $n$ output port blocked (OPB) lines, one per output port. A request line is asserted when the use (‘’service’’) of the particular crosspoint is needed. In order to provide flow control, it is sometimes necessary to block a switch from forwarding packets to one of its outputs. An OPB line is asserted in order to prevent use of a particular output port by inhibiting granting of any crosspoint in the corresponding column. In this and the following chapters we consider a crosspoint to be requested only if the request line is asserted while the block line is negated. The outputs from the arbiter consist of $n^2$ grant lines and $n^2$ control lines. The grant lines indicate which crosspoint requests have been granted, while the control lines connect or disconnect the individual crosspoints in the crossbar.

At any point in time, a multi-queue input buffer can transmit through the crossbar the packet at the head of any of its queues. If any one of these packets is
Figure 3.2: A crossbar arbiter in context. The crossbar connects $n^2$-bit buses from the input buffers to the $n$ output ports. Crosspoints (row/column connections) are requested using the $n^2$ request lines, and granted using the grant lines. The OPB (output port blocked) lines indicate which output ports are blocked and should not participate in the arbitration.

transmitted, the input buffer output bandwidth is fully utilized. We consider each queue to be a “user” and each internal crossbar bus to be a “server” (resource). Hence, the switch consists of $n^2$ users and $2n$ resources. A request from a queue can be granted only when the required input bus and the required output bus are available. Hence, as with the Concert segmented ring bus described in Chapter 2, each user requests more than one resource. Furthermore, as with the segmented bus, arbitration for the different resources cannot be performed independently. In order to maximize resource utilization and performance, the result of the arbitration of one resource (e.g. an input bus) may have to be modified if the winner loses the arbitration for another resource (an output bus). Unlike the segmented bus
arbitration problem, if a user has a request, it is always for the same two resources. Hence, only $n$ out of the $n^2$ users may contend for any resource. Furthermore, at most $n$ out of the $n^2$ users may win each arbitration.

![Diagram of user-resource model of symmetric arbitration of a 2x2 crossbar]

**Figure 3.3:** A user-resource model of symmetric arbitration of a 2x2 crossbar.

A user-resource model of a 2x2 crossbar with multi-queue input buffers is shown in Figure 3.3. For any crossbar, the arbitration problem can be described as a matrix of requests, each one for a crosspoint of the crossbar. The goal is to arbitrate among the requests so that at most one grant is given to a row of the matrix, and at most one grant is given to a column of the matrix. With conventional FIFO buffers, there are never conflicting requests for crosspoints on the same row. With multi-queue input buffers, conflicting requests on the same row can occur. Due to this symmetry between rows (input buses and input ports) and columns (output buses and output ports), the function of the required arbiter is called symmetric crossbar arbitration. Note that conventional crossbar arbitration is a special case of symmetric crossbar arbitration.
Since DAMQ buffers and other multi-queue buffers allow more than one request to the crossbar arbiter, there is the opportunity to connect more crosspoints of the crossbar than with FIFO buffers, thus leading to higher throughput and lower latency. Figure 3.4 shows an example of buffer contents and how requests can be arbitrated for FIFO and DAMQ buffers. The numbers in the buffers represent the destination output ports of the packets. The crosspoints with single circles indicate denied requests, while those with double circles indicate granted requests.

### 3.2. Symmetric Crossbar Arbiters

With multi-queue input buffers there is the potential for achieving significantly higher network throughput than with conventional FIFO buffers [Tami88a]. In order to realize this potential, it is necessary to use symmetric crossbar arbiters which, on the average, connect more crosspoints following every arbitration cycle than conventional crossbar arbiters. Since the
The arbitration task is more complex than with FIFO buffers, there is a question of whether ‘‘good’’ arbiters will be too slow and/or use excessive chip area.

In this section the importance of the arbitration scheme for achieving high performance is demonstrated. Several possible schemes are presented and discussed. The focus is on three schemes that appear particularly promising in terms of the potential for high performance and practical implementation. More extensive discussion of the performance and implementation of the proposed schemes are found in Sections 3.3 and 3.4, respectively.

3.2.1. The Importance of the Arbitration Scheme

There may be a question of whether there is a significant performance difference between ‘‘good’’ and ‘‘poor’’ symmetric crossbar arbitration schemes. In this section we demonstrate that there is such a difference by using static probabilistic analysis to compare two arbitration schemes for a 2×2 switch: a practical scheme that follows naturally from previous work on arbiter design, and a theoretical scheme which is not practical but can achieve nearly optimal performance. In order to calibrate the performance of the proposed schemes, they are compared to conventional crossbar arbitration for a switch with FIFO buffers.

As described earlier, with FIFO arbitration (FIFOA), there is at most one request for each row. Multiple requests for each column are arbitrated independently, using round-robin arbiters [Bhuy87, Vern88]. For a switch with multi-queue buffers, efficient symmetric crossbar arbitration maximizes the number of crosspoints utilized. We consider the performance of a statically optimal arbiter (SOA), which examines all the requests and searches for the
arbitration result that maximizes the throughput for the next cycle. If there are several configurations with equal throughput, one is selected randomly. The SOA uses exhaustive search of all legal configurations and is clearly not a practical arbiter design. We use it for comparison with other, more practical arbiter designs.

A straightforward extension of FIFO arbitration to symmetric crossbar arbitration is to decompose the arbitration process into two steps: arbitration among conflicting column requests followed by arbitration among conflicting row requests that have won the first step. Such a two-step arbiter (TSA) is amenable to VLSI implementation as an $n \times n$ array of arbitration cells, one cell per crosspoint (Figure 3.5). For each crosspoint $(i,j)$, in addition to the request ($R_{i,j}$) input and grant ($G_{i,j}$) output, each cell has two inputs, north ($N_{i,j}$) and west ($W_{i,j}$), and two outputs, south ($S_{i,j}$) and east ($E_{i,j}$). Note that $N_{i,j} = S_{i-1,j}$ and $W_{i,j} = E_{i,j-1}$. The $N_{i,j}$ signal indicates that the rows above did not request column $j$. The $W_{i,j}$ signal indicates that there are no granted requests for the crosspoints to the left. The $G$ output is asserted if, and only if, the crosspoint is requested and both the $N$ and the $W$ inputs are asserted. Thus, $G_{i,j} = R_{i,j} \land N_{i,j} \land W_{i,j}$, $S_{i,j} = N_{i,j} \land \overline{R_{i,j}}$, and $E_{i,j} = W_{i,j} \land \overline{G_{i,j}}$. For the highest priority row and column, all the $N$ and $W$ inputs, respectively, are set to 1. If the arbitration cells are combinational circuits with propagation delay $T$, then, for an $n \times n$ crossbar, the arbiter reaches a valid arbitration configuration in $(2n-1)T$ time units.

In order to ensure fairness with the two-step arbiter, the highest-priority row and column must not be fixed. Hence, while in the discussion above we assumed that crosspoint $(1,1)$ has the highest priority, fairness requires that each crosspoint has an equal opportunity to have the highest priority. With top priority at a
crosspoint other than \((1,1)\), the scheme requires that the \(S\) output from the bottom row is connected to the \(N\) input of the top row and the \(E\) output from the rightmost column is connected to the \(W\) input of the left-most column. Two \(n\)-stage token rings (circular shift registers) are used to keep track of the highest priority row and the highest priority column. The column shift register is advanced following each arbitration cycle while the row shift register is advanced following every \(n\) arbitration cycles. Figure 3.5 shows an example of the operation of the 4×4 two-step arbiter, in which the top left crosspoint has the highest priority. In the figure, double squares indicate that the crosspoint has been requested while shaded squares indicate that the crosspoint has been granted.
Simple probabilistic analysis can be used to evaluate the throughput of switches with the arbitration schemes discussed above. We assume that requests for various crosspoints are independent, and that the probability that there is at least one packet in buffer $i$ destined for output $j$ is $p$, for all $1 \leq i, j \leq n$. For a switch with multi-queue buffers, this implies that the request probability for any crosspoint is $p$. For a FIFO switch, there can be only one request on each row, so the request probability for any crosspoint is $\frac{1}{n} [1 - (1 - p)^n]$. The probability of a grant for crosspoint $(i, j)$ will be denoted by $g_{i,j}$. The normalized throughput of the switch (the throughput per column) can be calculated using: $\left( \sum_{i=1}^{n} \sum_{j=1}^{n} g_{i,j} \right) / n$.

As an example, we will compare the three arbitration schemes for a 2x2 switch: FIFOA, SOA, and TSA.

Probabilistic analysis of crossbars in FIFO switches is discussed in [Mudg82]. For FIFOA, if the crosspoint request probability is $q$, the normalized throughput is $1 - (1 - q)^2$. Using $q = \frac{1}{2} \left[ 1 - (1 - p)^2 \right]$, the normalized throughput is $2p - 2p^2 + p^3 - \frac{1}{4}p^4$.

For SOA, six cases must be considered. (a) no requests: The probability is $(1 - p)^4$, and the throughput is 0. (b) 1 request: The probability is $\left[ \begin{array}{c} 4 \\ 1 \end{array} \right] p (1 - p)^3$ and the total throughput is 1. (c) 2 requests in the same row or column: The probability is $4p^2(1 - p)^2$ and the total throughput is 1. (d) 2 requests in different rows and in different columns: The probability is $2p^2(1 - p)^2$ and the total throughput is 2. (e) 3 requests: The probability is $\left[ \begin{array}{c} 4 \\ 3 \end{array} \right] p^3(1 - p)$ and the total throughput is 2. (f) 4 requests: The probability is $p^4$ and the total throughput is 2. The normalized throughput is 50% of the expected value of the total throughput,
calculated from the analysis above, i.e., $2p - 2p^2 + 2p^3 - p^4$.

For TSA, without loss of generality, we assume that crosspoint (1,1) has the highest priority. Let $v_{i,j}$ denote the probability that crosspoint $(i,j)$ wins the (vertical) arbitration for column $j$ in the first arbitration step. Clearly, $v_{1,1} = v_{1,2} = p$, and $v_{2,1} = v_{2,2} = (1 - p)p$. Based on this, we can calculate the grant probability for the second arbitration step: $g_{1,1} = v_{1,1} = p$, $g_{1,2} = (1 - v_{1,1})v_{1,2} = (1 - p)p$, $g_{2,1} = v_{2,1} = (1 - p)p$, and $g_{2,2} = (1 - v_{2,1})v_{2,2} = [1 - (1 - p)p](1 - p)p$. The normalized throughput is 50% of the sum of the grant probabilities, yielding $2p - 2p^2 + p^3 - \frac{1}{2}p^4$.

\[ \text{Normalized throughput} = 0.5 \left( 2p - 2p^2 + p^3 - \frac{1}{2}p^4 \right) \]

\[ g_{1,1} = v_{1,1} = p, \quad g_{1,2} = (1 - v_{1,1})v_{1,2} = (1 - p)p, \quad g_{2,1} = v_{2,1} = (1 - p)p, \quad g_{2,2} = (1 - v_{2,1})v_{2,2} = [1 - (1 - p)p](1 - p)p. \]

\[ \text{Normalized throughput} = 0.5 \left( 2p - 2p^2 + p^3 - \frac{1}{2}p^4 \right) \]

\[ g_{1,1} = v_{1,1} = p, \quad g_{1,2} = (1 - v_{1,1})v_{1,2} = (1 - p)p, \quad g_{2,1} = v_{2,1} = (1 - p)p, \quad g_{2,2} = (1 - v_{2,1})v_{2,2} = [1 - (1 - p)p](1 - p)p. \]

The results of the above analysis are shown in Figure 3.6. As expected, SOA outperforms FIFOA. TSA, however, is even worse than FIFOA, especially when

\[ \text{Normalized throughput} = 0.5 \left( 2p - 2p^2 + p^3 - \frac{1}{2}p^4 \right) \]
the request probability is high. In the extreme case where \( p = 1 \), the normalized throughput with TSA is \( 1/2 \), because the requests granted after the first arbitration step are all in the same row. This result indicates that a poor symmetric crossbar arbiter can negate the potential performance advantage of multi-queue buffers.

### 3.2.2. The Skewed Two-Step Arbiter

A key problem with the two step arbiter (TSA) is that, in the first step, the top priority is given to the same row for all the column arbitrations. This increases the probability that multiple crosspoints in one row will win the first step, even though only one of them can be used at a time. A possible solution to this problem is to give the top priority to different rows for the different column arbitration. The ‘‘skew’’ in the column arbitration starting points is provided in the skewed two-step arbiter (STSA).

The operation of the STSA is identical to the TSA, except for the mechanism used to indicate the top priority cells. Specifically, instead of the two \( n \) bit circular shift registers which point to the top priority row and column, the STSA uses a single \( n \) bit circular shift register, which points to a ‘‘wrapped diagonal’’ of top priority crosspoints. For example, Figure 3.7 shows a STSA where the ‘‘diagonal’’ of high priority cells consists of cells (1,1), (2,4), (3,3), and (4,2). The result is that four connections are made by the STSA, while with the TSA, there is no way for four connections to be made for the specific pattern of requests shown.

The arbitration cells of the STSA are identical to those used in the TSA. The periphery of the array of arbitration cells, which changes the top priority cells to maintain fairness, is simpler since there is only one shift register instead of two.
Figure 3.7: A skewed two-step symmetric crossbar arbiter. For the example arbitration shown, the top priority cells are (1,1), (2,4), (3,3), and (4,2).

With the STSA, the horizontal (row) arbitration is performed in parallel with the vertical (column) arbitration. Hence, the STSA is faster (has smaller worst case delay) than the TSA. Specifically, if the delay per cell is $T$ time units, the worst case arbitration time for the STSA is only $nT$ time units.

3.2.3. The Wave Front Arbiter

The low performance of the TSA is a direct result of the partitioning of the arbitration process into the two separate steps of column and row arbitration. As discussed earlier, this results in low throughput since multiple crosspoints on high priority rows are likely to win the first step and prevent other crosspoints on their columns from being used. With the wave front arbiter (WFA) the entire arbitration
process is performed in one step, so higher throughput is expected.

Rather than starting with a top priority row, the arbitration process with WFA begins with one top priority arbitration cell. The arbitration cells reach their final configuration in a “wave front” that moves diagonally from the top left to the bottom right corner of the arbiter. Figure 3.8 shows the operation of the WFA with the top priority in arbitration cell (1,1). In order to maintain fairness, two $n$ bit circular shift registers, vertical and horizontal, as with TSA, are used to select the current top priority cell. The horizontal shift register is shifted every cycle, while the vertical shift register is shifted every $n$ cycles.

The critical difference between WFA and TSA as well as STSA is the function of the arbitration cell. Specifically, the definition of the $N_{i,j}$ signal is
different from the one discussed in Section 3.2.1. With TSA and STSA, $N_{i,j}$ indicates that none of the rows above have requested column $j$. With WFA, $N_{i,j}$ indicates that there are no granted requests for the crosspoints above $(i,j)$. Hence, using the notation of Section 3.2.1, $G_{i,j} = R_{i,j} \land N_{i,j} \land W_{i,j}$, $S_{i,j} = N_{i,j} \land \overline{G_{i,j}}$, and $E_{i,j} = W_{i,j} \land \overline{G_{i,j}}$. As with TSA and STSA, the cells are simple combinational circuits. If cell $(1,1)$ has the top priority and a cell performs its operation in $T$ time units, the outputs of cell $(i,j)$ are stable in their final values after $(i+j-1)T$ time units. Hence, the arbitration completes after $(2n-1)T$ time units.

### 3.2.4. The Wrapped Wave Front Arbiter

With WFA, in the first stage of arbitration (in the first $T$ time units), only one crosspoint is “processed” so at most one final grant signal is generated. This appears wasteful since even in the first stage it is possible to process $n$ crosspoints which are guaranteed not to conflict. The $n$ crosspoints of any “wrapped diagonal” of the arbitration array (see Figure 3.9) are guaranteed not to conflict since they are all on different rows and different columns. If the arbitration “wave front” begins with all $n$ crosspoints of such a diagonal, the arbitration of $n$ crosspoints instead of 1 crosspoint is completed after the first $T$ time units. This basic idea is the basis of the wrapped wavefront arbiter (WWFA).

The arbitration cells used in the WWFA are identical to those used with the WFA. The difference between the schemes is only in the mechanism used to indicate the top priority cells when the arbitration process begins. Specifically, with WFA two circular $n$ bit shift registers are used to indicate the top priority cell, as with TSA. With WWFA, a single $n$ bit circular shift register is used to indicate
Figure 3.9: A wrapped wave front symmetric crossbar arbiter. The “wrapped diagonal,” (1,1), (2,4), (3,3), and (4,2) has the top priority. The numbered diagonals indicate the progression of the arbitration wave front.

the wrapped diagonal of top priority crosspoints, as with STSA.

Figure 3.9 shows the arbitration process where the diagonal of high priority cells consists of cells (1,1), (2,4), (3,3), and (4,2). If a cell performs its operation in $T$ time units, the outputs of cell $(i,j)$ are stable in their final values after $[((i+j-2) \mod n) + 1]T$ time units. Hence, the WWFA is faster than the WFA, completing an arbitration in $nT$ time units.
3.3. Performance Evaluation

In order to understand whether there is a significant performance difference between ‘‘good’’ and ‘‘poor’’ symmetric crossbar arbitration schemes, static probabilistic analysis and extensive simulations have been conducted. The simple static analysis provides an initial indication of the expected throughput for single switches. Realistic performance evaluation for both single switches and 64×64 interconnection networks, which takes into account the queueing effects, is shown using event-driven simulations.

3.3.1. Simple Probabilistic Analysis for a 2×2 Switch

The analysis of the STSA is similar to the analysis of TSA. Let \( v_{i,j} \) denote the probability that crosspoint \((i,j)\) wins the (vertical) arbitration for column \(j\) in the first arbitration step. Without loss of generality, we assume that the diagonal of top priority cells consists of cells \((1,2)\) and \((2,1)\). Clearly, \( v_{2,1} = v_{1,2} = p \), and \( v_{1,1} = v_{2,2} = (1 - p)p \). Based on this, we can calculate the grant probability for the second arbitration step:

\[
\begin{align*}
g_{1,2} &= v_{1,2} = p, \quad g_{2,1} = v_{2,1} = p, \\
g_{1,1} &= (1 - v_{1,2})v_{1,1} = p(1 - p)^2, \quad g_{2,2} = (1 - v_{2,1})v_{2,2} = p(1 - p)^2.
\end{align*}
\]

The normalized throughput is 50\% of the sum of the grant probabilities, yielding

\[2p - 2p^2 + p^3.\]

For WFA, we assume, without loss of generality, that cell \((1,1)\) has the top priority. This implies that if cell \((1,1)\) is requested, it is granted, i.e., \( g_{1,1} = p \). Cells \((1,2)\) and \((2,1)\) are granted if requested and cell \((1,1)\) has not been granted, i.e., \( g_{1,2} = g_{2,1} = p(1 - p) \). Cell \((2,2)\) is granted if, and only if, either cell \((2,2)\) is
the only cell requested, or cell (1,1) is also requested, thus blocking possible grants for cells (1,2) and (2,1). Hence, \( g_{2,2} = p(1-p)^3 + p^2 \). The normalized throughput is 50% of the sum of the grant probabilities, yielding 
\[
2p - 2p^2 + \frac{3}{2}p^3 - \frac{1}{2}p^4.
\]

It is easy to show that, for a 2×2 switch, WWFA always results in the same arbitration result as STSA. It should be noted that these two arbitration schemes are not identical with larger switches (see Section 3.3.3). For WWFA, we assume, without loss of generality, that the diagonal of top priority cell consists of cells (1,1) and (2,2). For both of these cells the implication is that if they are requested, they are granted, i.e., \( g_{1,1} = g_{2,2} = p \). Cells (1,2) and (2,1) can be granted only if they are requested and neither cells (1,1) or (2,2) have been requested, i.e., \( g_{1,2} = g_{2,1} = p(1-p)^2 \). The normalized throughput is 50% of the sum of the grant probabilities, yielding 
\[
2p - 2p^2 + p^3.
\]

Figure 3.10 shows the results of the probabilistic analysis of a 2×2 switch with all the arbitration schemes discussed in this section. As expected, SOA outperforms all the practical schemes. It should be noted that three of the practical arbitration schemes proposed (STSA, WFA, and WWFA) achieve nearly the same performance as the SOA scheme. This important result, that the benefits of multi-queue buffers can be fully realized with practical arbitration schemes, is confirmed in Section 3.3.3.

On the other hand, TSA is even worse than FIFOA, especially when the request probability is high. In the extreme case where \( p = 1 \), the normalized throughput with TSA is 1/2, because the requests granted after the first arbitration step are all in the same row. This result indicates that a poor symmetric crossbar arbiter can negate the potential performance advantage of multi-queue buffers.
3.3.2. Probabilistic Analysis for an \( n \times n \) Switch

The probabilistic analysis in Section 3.3.1 is only practical for small switches since the number of cases is limited. In this section, a more general method is presented for performance analysis of the different arbitration schemes for any size \( n \times n \) switches. The SOA scheme is not included here due to the difficulty of analyzing its performance for any size switches.

For FIFOA, if the crosspoint request probability is \( q \), the normalized throughput is \( 1 - (1-q)^n \) for an \( n \times n \) switch. As shown in Section 3.2.1, \( q \) is equal to \( \frac{1}{n} [1 - (1-p)^n] \). Hence, the normalized throughput is

\[
1 - \left( 1 - \frac{1}{n} [1 - (1-p)^n] \right)^n.
\]
For TSA, as in Section 3.2.1, we can assume that crosspoint (1,1) has the top priority without loss of generality. Let $v_{i,j}$ denote the probability that crosspoint $(i,j)$ wins the arbitration for column $j$ in the first arbitration step. Then $v_{i,j} = (1 - p)^{i-1}p$. The throughput for row $i$ is $1 - \prod_{j=1}^{n} (1 - v_{i,j})$, which is equal to

$$1 - \prod_{j=1}^{n} [1 - (1 - p)^{j-1}p].$$

The normalized throughput is $1/n$ of the total throughput, yielding

$$\frac{1}{n} \sum_{i=1}^{n} \left\{ 1 - \prod_{j=1}^{n} [1 - (1 - p)^{j-1}p] \right\}.$$

For STSA, we can assume that the wrapped diagonal (1,1), (n,2), (n−1,3), · · ·, (2,n) has the top priority without loss of generality. Since the throughput for any row or any column is the same due to the symmetry of STSA, we will derive the normalized throughput by using the sum of the grant probabilities of row 1 in the following. Again, let $v_{i,j}$ denote the probability that crosspoint $(i,j)$ wins the arbitration for column $j$ in the first arbitration step. Then $v_{1,j} = (1 - p)^{j-1}p$. The grant probability of crosspoint (1,j) can be calculated based the equation:

$$g_{1,j} = \prod_{k=1}^{j} (1 - v_{1,k}) v_{1,j}.$$  

The throughput of row 1 is $\sum_{j=1}^{n} g_{1,j}$, which is equal to

$$\sum_{j=1}^{n} \prod_{k=1}^{j} (1 - v_{1,k}) v_{1,j}.$$  

Hence, the normalized throughput is:

$$\sum_{j=1}^{n} \left\{ \prod_{k=1}^{j} [1 - (1 - p)^{k-1}p] (1-p)^{j-1}p \right\}.$$

The analysis of WFA is more complex than the above schemes, because the arbitration progresses in two dimensions instead of one dimension. For a boundary crosspoint (1,j), the probability of getting the grant is $(1 - p)^{j-1}p$, because it can
get the grant only when the crosspoints $(1,k)$, $k < j$, do not request it. Similarly, the probability of getting the grant for a boundary crosspoint $(i,1)$ is $(1 - p)^{j-1}p$.

For an arbitrary “inside” crosspoint, the analysis is not obvious, especially for crosspoints $(i,j)$ where $i$ and $j$ are large numbers. In the following theorem, we establish an equation for the relation between $g_{i-1,j-1}$, $g_{i,j-1}$, and $g_{i,j}$ where $i,j \geq 2$. By this equation and the already available results in the above for the boundary crosspoints, we can derive the grant probabilities for all the crosspoints.

**Theorem 3.1.** In the wave front arbitration scheme, the grant probabilities of crosspoints $(i,j)$, $(i-1,j-1)$, and $(i,j-1)$ have the relation:

$$g_{i,j} = \sum_{k=0}^{i-2} (1-p)^k p g_{i-1,j-1} + (1-p)^i g_{i,j-1}$$

where each crosspoint has an independent request probability $p$, and $g_{i,j}$ is the grant probability of crosspoint $(i,j)$.

**Proof.** The key to deriving the grant probability for an arbitrary crosspoint is to condition the boundary crosspoints $(i,1)$, $1 \leq i \leq n$. Let $P[(i,j)]$ denote the probability that crosspoint $(i,j)$ gets the grant, and $P[(i,j) | (x,y)]$ the probability that crosspoint $(i,j)$ gets the grant, given the condition that $(x,y)$ gets its own required grant. Then,

$$P[(i,j)] = P[(1,1)] \cdot P[(i,j) | (1,1)]$$

$$+ P[(2,1)] \cdot P[(i,j) | (2,1)]$$

$$+ \cdots$$

$$+ P[(i,1)] \cdot P[(i,j) | (i,1)]$$

$$+ U \cdot W.$$
$U$ is the probability that crosspoints $(k,1), k \leq i$, do not get the grant. $W$ is the probability that crosspoint $(i,j)$ gets the grant, given the condition that crosspoints $(k,1), k \leq i$, do not get the grant. $P[(i,j) | (1,1)]$ is actually equal to $P[(i-1,j-1)]$. The reason for this is that, given the condition that $(1,1)$ gets the grant, we can virtually remove row 1 and column 1 for deriving the original $P[(i,j)]$. When $(1,1)$ gets the grant, no other crosspoints in the same row and the same column can be granted. The other crosspoints in row 1 will ''pass'' the value received from the vertical input to the vertical output. Hence, row 1 can be removed without affecting the grant probability of $(i,j)$. Similarly, column 1 can be removed without affecting the grant probability of $(i,j)$. As a result, $P[(1,1)] \cdot P[(i,j) | (1,1)]$, which is the first product term in the above equation, is equal to $p \cdot g_{i-1,j-1}$. For the second product term $P[(2,1)] \cdot P[(i,j) | (2,1)]$, we can virtually remove row 2 and column 1 by the same reasoning. Hence this term is equal to $(1-p)p \cdot g_{i-1,j-1}$. The following product terms in the equation can be derived likewise except the last two product terms. The product term $P[(i,1)] \cdot P[(i,j) | (i,1)]$ is equal to 0. This is due to the fact that $P[(i,j) | (i,1)]$ equals 0, because $(i,j)$ cannot get the grant if $(i,1)$ gets it. In the last product term, $Q$ is equal to $(1-p)^j$, and $R$ is equal to $g_{i,j-1}$ because column 1 has no effect on $P[(i,j)]$ and thus can be virtually removed for this. Hence the last product term is equal to $(1-p)^j g_{i,j-1}$. As a result,

$$g_{i,j} = p \cdot g_{i-1,j-1} + (1-p)p \cdot g_{i-1,j-1} + (1-p)^2 p \cdot g_{i-1,j-1} + \cdots + (1-p)^j g_{i,j-1}.$$  

This can be expressed as

$$g_{i,j} = \sum_{k=0}^{i-2} (1-p)^k p \cdot g_{i-1,j-1} + (1-p)^j g_{i,j-1}.$$  

59
By using the grant probabilities for the boundary crosspoints and the recurrence equation in Theorem 3.1, we can derive, in order, all the grant probabilities of crosspoints (2,2), (2,3), (2,4), · · · , (2,n), (3,2), (3,3), (3,4), · · · , (n,n). The total throughput is the sum of all the grant probabilities. Hence the normalized throughput can be obtained from \( \sum_{i=1}^{n} \sum_{j=1}^{n} g_{i,j} \).

Similar to WFA, the arbitration in WWFA progresses in two dimensions and the analysis can be based on a recurrence equation. Again, to calculate the normalized throughput, the grant probability for each crosspoint is derived first. In WWFA, each crosspoint in the same wrapped diagonal has the same grant probability due to the symmetry. Let \( r_k, 1 \leq k \leq n \), denote the grant probability of any crosspoint in the \( k \)th wrapped diagonal, and \( g_{i,j} = r_k \) where \( [(i+j-1) \mod n] + 1 = k \). Note that the first wrapped diagonal consists of crosspoints \( (n,1), (n-1,2), (n-2,3), \cdots, (1,n) \), and any of them has the grant probability \( r_1 \). In the following theorem, we show that \( r_k \) can be derived from \( r_m \) where \( 1 \leq m \leq k-1 \).

**Theorem 3.2.** In the wrapped wave front arbitration scheme,

\[
r_k = (1-p)^2 \sum_{l=0}^{k-2} \binom{k-2}{l} (1-p)^{k-2-l} p^l r_{k-l-1}
\]

where each crosspoint has an independent request probability \( p \), and \( r_k \) is the grant probability of any crosspoint \( (i,j) \) in the same wrapped diagonal where \( [(i+j-1) \mod n] + 1 = k \).

**Proof.** Assume the crosspoints in the first wrapped diagonal are polled first in WWFA, so \( r_1 = p \). For the other wrapped diagonals of crosspoints, we can
condition the first wrapped diagonal to get their grant probabilities. Since the
arbitration wave front moves toward the right bottom corner, in the first wrapped
diagonal only the requests of the crosspoints between crosspoints \((n+1-j,j)\) and
\((i,n+1-i)\) and these two crosspoints themselves affect the grant probability of
crosspoint \((i,j)\). Let \(Q[l]\) denote the probability that there are \(l\) crosspoints
between crosspoints \((n+1-j,j)\) and \((i,n+1)\) which get the grant. Furthermore,
let \(Q[(i,j)|l]\) denote the probability that crosspoint \((i,j)\) gets the grant, given the
condition that there are \(l\) crosspoints between crosspoints \((n+1-j,j)\) and
\((i,n+1-i)\) which get the grant. Since crosspoints \((n+1-j,j)\) and \((i,n+1-i)\) cannot
have requests if crosspoint \((i,j)\) gets the grant, we can establish the following
equation:

\[
r_k = g_{i,j} = (1-p)^2(Q[0]Q[(i,j)|0] + Q[1]Q[(i,j)|1]) + Q[2]Q[(i,j)|2] + \cdots + Q[k-2]Q[(i,j)|k-2]).
\]

Note that there are \(k-2\) crosspoints between crosspoints \((n+1-j,j)\) and \((i,n+1-i)\).
The term \((1-p)^2\) is the probability that \((n+1-j,j)\) and \((i,n+1-i)\) have no requests.
The term \(Q[0]Q[(i,j)|0]\) is equal to \(\binom{k-2}{0} (1-p)^{k-2} r_{l-1}\), since \(Q[0]\) equals
\(\binom{k-2}{0} (1-p)^{k-2}\) and \(Q[(i,j)|0]\) equals \(r_{k-1}\). The reason for this is that, if all the
crosspoints in the first wrapped diagonal have no requests, they are virtually
nonexistent from the viewpoint of deriving \(g_{i,j}\). The term \(Q[1]Q[(i,j)|1]\) is
equal to \(\binom{k-3}{1} (1-p)^{k-3} r_{l-2}\). The reason why \(r_{l-2}\) is used here is that this
requesting crosspoint between \((n + 1, j, j)\) and \((i, n + 1, i)\) blocks all the crosspoints in its associated row and column. In other words, this row and this column are virtually nonexistent from the viewpoint of deriving \(g_{i,j}\). From the arbitration behavior in this case, \(Q[(i, j) | 1]\) is equal to \(r_{l-2}\). The same reasoning can be applied to the following terms in the above equation. Hence, the equation can be expressed as:

\[
r_k = (1 - p)^2 \sum_{l=0}^{k-2} \binom{k-2}{l} (1 - p)^{k-2-l} p^l r_{k-1-l}.
\]

This theorem basically is a recurrence equation, which can be used to derive \(r_k\) based on \(r_m\) where \(1 \leq m \leq k - 1\). Since \(r_1\) is equal to \(p\), the grant probabilities for the crosspoints in the other wrapped diagonals can be derived in the sequence of \(r_2, r_3, \ldots, r_n\) using the equation. The normalized throughput is the average of the grant probabilities for all the wrapped diagonals, which is \(\frac{1}{n} \sum_{k=1}^{n} r_k\).

Figures 3.11, 3.12, 3.13, and 3.14 show the probabilistic analysis results for FIFOA, TSA, STSA, WFA, and WWFA for 4×4, 8×8, 16×16, and 64×64 switches. In general, performance comparison between different arbitration schemes in these switches is similar to the 2×2 switch. However, the performance difference between “good” schemes and “poor” schemes increases as the switch size increases. When the switch size is large, WFA and WWFA significantly outperform the other schemes, with WFA slightly better.
Figure 3.11: Probabilistic analysis of throughput for 4×4 switches. A 4×4 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

Figure 3.12: Probabilistic analysis of throughput for 8×8 switches. An 8×8 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.
Figure 3.13: Probabilistic analysis of throughput for 16×16 switches. A 16×16 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

Figure 3.14: Probabilistic analysis of throughput for 64×64 switches. A 64×64 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.
3.3.3. Simulation of Synchronous Networks

The probabilistic analysis used in Section 3.3.1 and Section 3.3.2 does not take into account the queueing effects and cannot be easily extended to evaluating the performance of a complete network rather than a single switch. In order to perform more realistic performance evaluation of the proposed arbitration schemes, we used an event-driven simulator [Swop86], that allows detailed simulation of arbitrary objects interacting via messages.

3.3.3.1. Performance Measurement

Our simulations focus on a 64×64 Omega network [Lawr75] which uses blocking switches. This multistage buffered network operates synchronously. The minimum delay per stage is called a stage cycle [Yoon90]. A stage cycle is the latency of a packet per stage in an empty network, where there is no contention for buffer space or output ports. A packet may be forwarded to a switch in the next stage only if there is a free buffer slot in the input buffer of the switch.

Ideally, before implementing a system, the performance of the different arbitration schemes should be studied using traces from actual applications. Such traces may include nonuniform traffic patterns and variable length packets. The simulations for synchronous networks in this chapter are based on a traffic model with the following properties: 1) packets are of a fixed size, 2) during each stage cycle there is an equal probability of generating a packet at each of the source nodes, 3) packet destinations are uniformly distributed over all the network outputs. Two of the performance measures used are the average latency and the normalized network throughput. The latency is the number of stage cycles from
the creation of a packet to its delivery at the destination. Hence, in a three stage network, the minimum latency is three stage cycles. The normalized throughput is the average number of packets received by each output of the network per stage cycle.

Another important measure of the “quality” of arbitration is fairness. Ideally, all packets receive fair (equal) treatment so that the latency for all packets should be the same. In reality, some packets are forwarded through the network faster than others. A possible measure of fairness is the difference between the maximum and minimum latency through the network. For all arbitration schemes, the minimum latency is bounded by the speed of the hardware. Hence, the maximum (worst case) latency can be used as a measure of fairness. In simulations, the maximum latency is a poor measure since it is susceptible to statistical anomalies of the simulation. In particular, the simulated maximum latency is sensitive to the initial random number seeds and increases as the number of packets transmitted during the simulation is increased. As a measure of fairness that is less susceptible to statistical anomalies of the simulation, we use the “99th percentile latency.” Considering all the packets transmitted through the network during a simulation run, the 99th percentile latency is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies) from the network. Hence, 99% of the packets were transmitted through the network faster than the 99th percentile latency. Ideally, the 99th percentile latency will be the same as the average latency. In reality, even if all conflicts local to the switches are resolved in a fair way, the 99th percentile latency may be several times larger than the average latency (see Section 3.3.3.3). Any lack of fairness in
resolving conflicts local to the switches will result in increasing the 99th latency and increasing the difference between the 99th percentile and average latencies.

We report the results from simulating a single switch as well as from simulating the entire 64x64 network. In all cases we are interested in the steady-state performance. Hence, the data collected during the start-up phase of the simulation was discarded [Mitr82]. In order to determine the duration of the start-up phase, the simulator produced the performance measures (throughput, average latency, and 99th percentile latency) for “sub-runs” of the simulation. Specifically, for the network simulations, for every 1,000 packets that left the network, the performance measures for those 1000 packets were produced. For the single switch simulations intervals of 100 packets were used. The end of the start-up phase is detected when the interval performance measures stop moving in one direction and start oscillating [Mitr82]. In all the simulations, the performance measures started to oscillate after at most five intervals. Hence, for the network simulations the start-up phase lasts less than 5,000 packets and for the single switch simulation it lasts less than 500 packets. In our network simulations the first 32,000 packets were discarded while for the single switch simulations the first 4,000 packets were discarded. Hence, our results reflect steady-state performance.

The single switch simulations were run with each sender generating approximately 3,000 packets. The network simulations were run with each sender generating approximately 1,500 packets. Simulation was terminated once there was one sender that had completed sending all its allotted packets. In order to verify the validity of the results, for each rate of generating messages at the sources, at least eight runs were performed with different random number seeds.
The throughput and latency numbers used are the averages of results from these multiple runs [Welc83]. Each curve is based on twelve points.

For each point used in the curves presented, we calculated the 95% confidence interval [Welc83]. In all cases the confidence interval for the throughput is within 1% of the number reported. For average latencies, the confidence interval is within 3% of the results reported. For the 99th percentile latencies, the confidence interval is within 7% of the numbers reported. Out of a total of 432 points used to generate the graphs for 99th percentile latencies, there were only 20 points for which the 95% confidence interval was not within 5% of the number reported.

In addition to the six arbitration schemes discussed in Section 3.3.1 (FIFOA, TSA, STSA, WFA, WWFA, and SOA), we consider two other possible schemes:
1) fixed priority wave front arbitration (FPWFA), which is identical to WFA except that the top priority always remains in the same arbitration cell, and
2) longest queue first arbitration (LQFA), which assigns priorities to inputs in direct proportion to the number of packets in their buffers and arbitrates between different queues in an input buffer by assigning queue priorities in direct proportion to the number of packets in each queue [Kuma84]. Since there is no need to rotate priorities with FPWFA, its implementation is expected to be simpler. LQFA would be difficult to implement efficiently, but it may be expected to perform better than other practical arbitration schemes. DAMQ buffers [Tami88a] are assumed in the evaluation of all seven symmetric crossbar arbitration schemes.

In the following two sections we consider the effect of the crossbar arbitration scheme on the performance of 4×4 switches with input buffers that can
accommodate up to four packets. We consider a single switch first and then the 64×64 network, consisting of three stages of 4×4 switches. The effects of switch size (number of inputs and outputs) and buffer size (number of packet slots per input buffer) are considered in Section 3.3.3.4.

3.3.3.2. Performance Evaluation of a 4×4 Switch

Figure 3.15 shows the average latency versus throughput of a single 4×4 switch with four packets slots per input buffer. In general, the results corroborate the probabilistic analysis of Section 3.3.1. The maximum throughput achieved, as well as the average latencies for high throughputs, are dependent on the arbitration policy. In particular, poor policies, such as FIFOA and TSA, achieve significantly lower maximum throughput than efficient policies, such as WFA or LQFA. WFA and WWFA provide approximately the same performance as the much more expensive LQFA, and only slightly worse performance than the theoretical SOA. While the probabilistic analysis of a 2×2 switch showed identical results for STSA and WWFA, similar analysis of 4×4 and larger switches shows that WWFA results in significantly better performance. This mediocre performance of STSA relative to WFA and WWFA is also shown in Figure 3.15.

The simulation results for 99th percentile latency are shown in Figure 3.16. TSA results in the worst performance with respect to 99th percentile latency as well as with respect to average latency. SOA results in relatively poor (high) 99th percentile latency. Thus, it appears that the optimization for maximum throughput is at the expense of fairness. As might be expected, fixed priority results in unfair arbitration — FPWFA results in the second highest 99th percentile latencies even
Figure 3.15: Average latency vs. normalized throughput of a single 4×4 switch. Four packet slots per input buffer.

Figure 3.16: 99th percentile latency vs. normalized throughput of a single 4×4 switch. Four packet slots per input buffer.
though it achieves low average latencies (Figure 3.15). Relative to the other schemes, FIFOA performs well with respect to 99th percentile latency despite its poor average latency performance. This is due to the inherent fairness in a first-in-first-out mechanism.

![Average Latency vs. Normalized Throughput](image)

**Figure 3.17:** Average latency vs. normalized throughput of a 64×64 Omega network using 4×4 switches. Four packet slots per input buffer.

### 3.3.3.3. Performance Evaluation of a 64×64 Omega Network

Figure 3.17 shows the average latency versus normalized throughput of a 64×64 Omega network, consisting of three stages of 4×4 switches with four packet slots per input buffer. Qualitatively, the results are similar to the results of the single switch simulations. One difference is that FIFOA is now shown to provide approximately the same performance as TSA, while the single switch simulations indicated that the FIFOA performance is significantly better. The reason for this apparent discrepancy is that the performance with FIFOA is more severely
degraded by output ports which are blocked due to a full buffer in the next stage. This effect is very important, but is obviously not relevant to a single switch. Another interesting result is that LQFA performs best, even better than SOA. This is due to the fact that LQFA is optimized for reducing the probability of buffers overflowing, which is often the cause of blocking in a highly utilized network.

![Figure 3.18](image)

**Figure 3.18:** 99th percentile latency vs. throughput of a 64×64 Omega network using 4×4 switches. Four packet slots per input buffer.

The simulation results for 99th percentile latency are shown in Figure 3.18. The effects of blocked output ports, discussed above, manifest themselves strongly in these measurements. For single switch simulations (Figure 3.16), the inherent fairness of FIFO resulted in similar 99th percentile latencies with FIFOA and, for example, SOA. On the other hand, with the Omega network simulations the switches with FIFOA reach saturation at a much lower throughput, resulting in different performance characteristics. As could be expected, LQFA provides the best performance with respect to the 99th percentile latency as well as with respect
to average latency. It should be noted that the practical arbitration schemes, WFA and WWFA, achieve the best performance relative to all the schemes other than LQFA.

3.3.3.4. Impact of Buffer Size and Switch Size

All the simulations discussed above were done for 4×4 switches with four packet slots per input buffer. We consider here whether the results of these simulations are significantly different if two important parameters are changed: buffer size (number of packet slots per buffer) and switch size (number of inputs and outputs in each switch).

Figure 3.19: The impact of input buffer size on average latency. Average latency vs. normalized throughput of 64×64 Omega networks using 4×4 switches with various buffer sizes.

Figure 3.19 shows the average latency versus normalized throughput of 64×64 networks.
Figure 3.20: The impact of input buffer size on 99th percentile latency. 99th percentile latency vs. normalized throughput of 64×64 Omega networks using 4×4 switches with various buffer sizes.

Omega networks of 4×4 switches with two, four, and six packet slots per input buffer. As shown elsewhere [Dias81b, Tami88a], the maximum throughput increases as the buffer size increases. For small buffers (two packet slots), there is almost no difference between the various symmetric arbitration schemes and there is only a small performance improvement of those over FIFOA. The performance advantage of multi-queue buffers, and thus of symmetric crossbar arbitration, increases as the buffer size increases. The reason for this is that for a larger buffer there is a higher probability that the buffer contains packets destined to several outputs and thus, with a multi-queue buffer, there is a higher probability of sending a packet from the input port. The increase in the relative performance of multi-queue buffers over FIFO buffers is large as the buffer size increases from two slots to four slots, but is relatively small for buffer increase of from four slots to six
slots. This is due to the fact that, with a 4×4 switch, the number of different destinations of packets in a particular buffer cannot be above 4, regardless of the size of the buffer. As the buffer size increases from two to four slots, there is a significant increase in the difference between the maximum achievable throughput with an efficient symmetric crossbar arbitration scheme (SOA) and an inefficient scheme (STSA). However, the difference in performance between efficient and inefficient schemes remains approximately the same, as the buffer size increases from four to six slots. The reason for this is, once again, that with uniform traffic and 4×4 switches, there is only a small increase in the expected number of destinations as the buffer size increases from four to six slots. Hence, the patterns of requests to the arbiter are approximately the same for four and six slot buffers, leading to approximately the same number of connections (packets transferred) per cycle.

Figure 3.20 shows the 99th percentile latency versus normalized throughput of 64×64 Omega networks of 4×4 switches with two, four, and six packet slots per input buffer. As with the average latencies, the performance advantage of good arbitration schemes increase with increasing buffer size. It is interesting to note that, with respect to 99th percentile latency, for small buffers, STSA performs better than SOA, but SOA performs slightly better for large buffers. The reason for the relatively good performance of STSA for small buffers is that STSA is inherently a fair arbitration scheme — each row and each column have equal time at the top priority position. On the other hand, as mentioned earlier, SOA achieves higher maximum throughput at the expense of fairness. For large buffers, the difference in the maximum throughputs achieved by the two schemes becomes
more pronounced and leads to higher 99th percentile latencies with STSA.

Figure 3.21: The impact of switch size. Average latency per stage vs. normalized throughput of 64×64 Omega networks with four packet slots per input buffer and various switch sizes.

Figure 3.21 shows the average latency per stage versus normalized throughput of 64×64 Omega networks with four packet slot input buffers in their switches, using 6, 3, and 2 stages of 2×2, 4×4, and 8×8 switches, respectively. It should be noted that the average latency through the network is the product of the average latency per stage, shown in Figure 3.21, and the number of stages. For FIFOA, as the switch size increases from 2×2 to 4×4, the maximum throughput decreases [Yoon90]. The maximum FIFOA throughput remains approximately the same when the switch size increases from 4×4 to 8×8. A relatively poor symmetric arbitration scheme (STSA) results in approximately the same behavior. As the switch size increases, there is an increase in the performance advantage of the good symmetric arbitration schemes (WFA, WWFA, LQFA), in terms of higher
maximum throughput, over FIFOA and STSA. With WFA, WWFA, and LQFA there is no decrease in maximum throughput as the switch size is increased from 2×2 to 4×4. Furthermore, the maximum throughput increases when the switch size is increased to 8×8. The differences in maximum throughput among WFA, WWFA, and LQFA remain approximately the same for all three switch sizes. Hence, with increasing switch size, the arbitration scheme becomes more important for achieving the maximum possible performance. WFA and WWFA consistently achieve almost the same performance as LQFA, which is too complex for practical implementation.

### 3.3.4. Simulation of Asynchronous Networks

In the simulations presented in Section 3.3.3, we assume that the network operates synchronously. In such synchronous networks, packets are transmitted and received by the communication switches in lock-step. In *asynchronous networks*, it takes multiple clock cycles to transmit an entire packet from one switch to its neighbor. Packets may arrive at a switch during any clock cycle and may be of variable length. In this section, we study the performance impact of the various arbitration schemes in asynchronous networks.

Links and buffers for each switch are assumed to be byte wide. A byte can be read from the buffer, written into the buffer, or transferred through a link in one cycle. Virtual cut-through switching [Kerm79] is used and the minimum delay for a packet going through the communication switch is five cycles [Tami92]. *Senders* generate packets and send them to the network. The interval between packet creation follows a geometric distribution. However, a packet is sent into the
network only if the corresponding input buffer has room for the entire packet. Otherwise, the sender blocks.

Figure 3.22 and Figure 3.23 show the performance of a single 4×4 switch with FIFOA, TSA, STSA, WFA, and WWFA in asynchronous networks. The packet size is evenly distributed between 8 and 32 bytes. The buffer size for each input port is 128 bytes. These two figures show that while the symmetric crossbar arbiters significantly outperform the FIFOA, there is no performance difference between the various symmetric crossbar arbiters. Figure 3.24 and Figure 3.25 show the performance of a 64×64 Omega network. Similar to the simulation results for single switches, the various symmetric crossbar arbiters exhibit very little performance difference. Given a request pattern in a crossbar, a poor arbitration scheme connects fewer crosspoints than a good arbitration scheme, leading to idle resources. However, the arbiter attempts to connect crosspoints for the crossbar continuously in every clock cycle. The idle crosspoints which could otherwise be connected in a good arbitration are likely to be connected in the arbitration in the next clock cycle. Hence, the negative performance impact of a poor arbitration scheme in an asynchronous network is likely to be very small.

Our simulation studies show that even with a small packet size such as 2 bytes, the performance of a poor arbitration scheme is comparable to that of a good arbitration scheme. Figure 3.26 and Figure 3.27 show the performance of a single 4×4 switch with the various arbiters in asynchronous networks. The packet size is 2 bytes and the buffer size for each input port is 10 bytes. Figure 3.28 and Figure 3.29 show the performance of a 64×64 network composed of two stages of 8×8 switches. The packet size is 2 bytes and the buffer size for each input port is
**Figure 3.22:** Performance of a single 4×4 switch in asynchronous networks. Average latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.

**Figure 3.23:** Performance of a single 4×4 switch in asynchronous networks. 99th percentile latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.
**Figure 3.24:** Performance of an asynchronous 64×64 network. Average latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.

**Figure 3.25:** Performance of an asynchronous 64×64 network. 99<sup>th</sup> percentile latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.
Figure 3.26: Performance of a single 4×4 switch in asynchronous networks. Average latency vs. normalized throughput. The packet size is 2 bytes. 10 bytes per input buffer.

Figure 3.27: Performance of a single 4×4 switch in asynchronous networks. 99th percentile latency vs. normalized throughput. The packet size is 2 bytes. 10 bytes per input buffer.
Figure 3.28: Performance of an asynchronous 64×64 network. Average latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 16 bytes per input buffer.

Figure 3.29: Performance of an asynchronous 64×64 network. 99th percentile latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 16 bytes per input buffer.
16 bytes. These figures show that even if the packet size is very small, the good symmetric crossbar arbiters achieve almost the same performance as the poor symmetric crossbar arbiters. The reason for this is that, again, with a poor arbitration scheme, the idle crosspoints which could otherwise be connected in a good arbitration are likely to be connected in the arbitration in the next clock cycle. Hence, the performance of a poor arbiter is very close to that of a good arbiter.

We also simulated for a 64×64 network composed of two stages of 8×8 switches, where each input of the switches has a large buffer. Figure 3.30 and Figure 3.31 show the simulation results. The packet size is still 2 bytes, but the buffer size is 80 bytes. The performance difference between the various symmetric crossbar arbiters in Figure 3.30 and Figure 3.31 is larger than that in Figure 3.28 and Figure 3.29. The reason for this is that a poor symmetric crossbar arbiter such as TSA connects much fewer crosspoints than a good symmetric crossbar arbiter when almost every crosspoint has a request. As we mentioned in Section 3.2, with TSA the probability that multiple crosspoints in a row win the first arbitration step is increased. Thus, compared to a good arbitration scheme, much fewer crosspoints can be connected due to substantial conflict in the second arbitration step. As a result, it is no longer true that the idle crosspoints which could otherwise be connected in a good arbitration are likely to be connected in the arbitration in the next clock cycle.
**Figure 3.30:** Performance of an asynchronous 64×64 network. Average latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 80 bytes per input buffer.

**Figure 3.31:** Performance of an asynchronous 64×64 network. 99th percentile latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 80 bytes per input buffer.
3.4. Implementation

In Section 3.2 and Section 3.3 we have shown that the crossbar arbitration policy has a significant impact on performance. The wave front and wrapped wave front arbitration schemes were shown to outperform many other schemes. WFA and WWFA, whose design is fundamentally simple to implement, were shown to achieve nearly the same performance as theoretical schemes which are optimized for high performance without regard to implementation complexity. For a final confirmation that WFA and WWFA are indeed practical arbitration schemes, this section discusses the circuit design and VLSI implementation of WFA and WWFA. Since the implementation of the two schemes is similar, the focus will be on WFA. However, most of the discussion applies to WWFA as well.

Figure 3.32: Organization of a 2×2 wave front arbiter.
3.4.1. Logic and Circuit Design

The structure of a 2×2 WFA and of a 2×2 WWFA arbiter is shown in Figures 3.32 and 3.33, respectively. The same arbitration cell, which is similar to the arbitration cell of Figure 3.5, can be used for both WFA and WWFA. The cell used in Figures 3.32 and 3.33 has the additional XP and YP inputs, which indicate top priority within a row (X direction), and within a column (Y direction), respectively. The XI, XO, YI, and YO signals correspond to the W, E, N, and S signals, respectively, of Figure 3.5. They are relabeled here due to the addition of the explicit priority signal, XP and YP. The R (request) and G (grant) signals are as in Figure 3.5, but are not shown in Figures 3.32 and 3.33.

A logic diagram of the cells used in Figures 3.32 and 3.33 is shown in Figure 3.34. The OPB signal indicates that the output port is blocked, so there is
Figure 3.34: An arbitration cell. \( XP \) and \( YP \) indicate top priority in the horizontal and vertical directions, respectively. The \( OPB \) line is 1 when the output port is not blocked.

no reason to include in the arbitration any crosspoint connecting to that port (column). The function implemented is a modification of the logic equations presented in Section 3.2.3. The difference is that the priority signals, \( XP \) and \( YP \), override the \( XI \) and \( YI \) signals. Since the inputs to the cells may go through more than one transition between 0 and 1 before the arbiter settles to its final value, static combinational logic is used to implement the cells.

\[
G = (R \land \overline{OPB}) \land (YI \lor YP) \land (XI \lor XP)
\]
\[
YO = (YI \lor YP) \land \overline{G}
\]
\[
XO = (XI \lor XP) \land \overline{G}
\]

As discussed in Section 3.2.3, for WFA, two token rings (shift registers) are used to identify the cell with the highest priority. The horizontal token is advanced every clock cycle, while the vertical token is advanced every \( n \) cycles, once the horizontal reaches its right-most cell. The token rings are implemented as simple
Figure 3.35: Horizontal and vertical priority token rings for a wave front arbiter. The reset signal (RES) initializes the two token rings to a valid state.

dynamic shift registers which, on system reset, are initialized so that the first cell is 1 while all the others are 0 (Figure 3.35). For WWFA, only one token ring of this type is needed. With the WFA arbiter, for each clock cycle only a single cell has both the vertical and horizontal priority tokens. For WWFA, the priority line from the token ring delivers a priority token to a whole diagonal of $n$ arbitration cells.

3.4.2. VLSI Layout and Circuit Simulation

Due to their simple regular structure, the WFA and WWFA arbiters are amenable to efficient VLSI implementation. In order to determine the actual implementation complexity and performance, we have laid out a 4×4 crossbar of eight-bit wide internal buses with a wave front arbiter. The layout was done for
**Figure 3.36:** The floorplan of a 4×4 crossbar of eight-bit wide buses with a wave front arbiter. The modules are drawn to scale.

CMOS technology, using the MOSIS scalable design rules. One possible floorplan of such a switch is based on integrating the crossbar with the arbiter so that each arbitration cell is together with the corresponding crosspoint. We have investigated this possibility but discovered that separating the arbiter from the crossbar, as shown in Figure 3.36, results in more compact layout.

Figure 3.36 shows the floorplan of the crossbar with the arbiter. The boxes in the figure are drawn to scale with respect to the sizes of the modules in the layout. The crossbar itself is 615 λ wide by 376 λ tall. The arbiter, with the priority shift registers is 649 λ by 527 λ. The layout of a single arbitration cell is shown in Figure 3.37. It should be noted that the crossbar and the arbiter are small relative to the input buffers. In our layout of a single 96 byte, four queue, DAMQ buffer,
Figure 3.37: VLSI layout of a single WFA arbitration cell. This cell is 152 $\lambda$ wide by 127 $\lambda$ tall.

the size of the storage is, approximately, $2500\lambda \times 1000\lambda$, while the size of the control is $3000\lambda \times 900\lambda$ [Fraz89]. Assuming a $2\mu$ technology, circuit simulation using SPICE indicates that the worst case delay for reaching an arbitration result is 15.5 ns. In our current design of the ComCoBB switch [Fraz89, Tami88a], the arbitration task must be completed in one 20 ns clock cycle. Hence, for the current design, our straightforward implementation of the wave front arbiter is fast enough. If higher speed is required, the WWFA arbiter can be used. With the WWFA arbiter, each arbitration can be completed in 10 ns from our circuit simulation.
3.5. Summary and Conclusions

Significant improvements in the performance of communication switches can be achieved by using multi-queue input buffers instead of FIFO buffers. A crossbar is often used to connect multi-queue input buffers to the output ports of the switch and an arbiter is needed to resolve conflicting requests for crossbar resources from the various queues. If all the queues of each input buffer are connected to a single crossbar input, the arbitration task is symmetric with respect to inputs and outputs. Since the result of the arbitration for each crosspoint depends on or influences the arbitration of all the crosspoints in the corresponding row and column, the arbiter is relatively complex and cannot be partitioned into independent row or column arbiters.

We have evaluated different symmetric crossbar arbitration schemes using static probabilistic analysis, single switch event-driven simulations, and simulations of a buffered Omega network. Our simulations of networks consisting of switches using different types of arbiters provided an evaluation of the quality of arbitration of these arbiters. The quality of arbitration of an arbiter is a measure of the “performance” of the arbiter in a network operating in steady state for a long period of time. Specifically, for a network operating in steady state, it is a measure of the average number of requests per unit time that are granted by the arbiter relative to the average number of requests per unit time that would be granted by an “optimal arbiter.” Although these simulations can be used to measure the quality of arbitration of the various arbiters, they did not take into account the fact that there are differences in the speeds of the circuits implementing the different arbiters (see Section 3.2). Depending on the implementation, these differences
may impact the latency of packets through the network.

In synchronous networks, a “good” symmetric crossbar arbitration scheme is essential to realizing the potential for performance improvement from multi-queue input buffers. In particular, a “poor” scheme may result in lower performance than with FIFO buffers while a “good” scheme can increase the maximum throughput of the network by more than 40%. As the buffer size and switch size increase, the benefits of a “good” arbitration scheme increases.

We have investigated several arbitration schemes, including complex theoretical schemes (statically optimal and longest-queue-first arbitrations), which cannot be implemented but could be expected to perform better than practical schemes. We have shown that, in order to achieve fairness, it is important to use a scheme which does not assign static priorities to queues. The wave front arbiter and the wrapped wave front arbiter, which are based on the propagation of an arbitration “wave” across an array of arbitration cells, were shown to achieve nearly the same performance as the complex theoretical schemes.

The proposed efficient wave front and wrapped wave front arbiters are amenable to simple regular implementation in VLSI. For a 4×4 switch, the layout of a wave front arbiter is larger than the data portion of the corresponding crossbar of eight-bit wide buses. This indicates that the arbiter is a significant module in terms of size, as well as performance, in communication switch implementation. Circuit simulation of our CMOS implementation of the wave front arbiter, using 2 μ technology, indicates that, for a 4×4 crossbar, this arbiter can produce a valid and efficient crossbar configuration in 15.5 ns.

For asynchronous networks, the choice among the various symmetric crossbar
arbiter has a less impact on performance than for synchronous networks. Just as in synchronous networks, for a given set of requests and port states, different arbiters will allow different connections to be made. For requests not granted, packets are blocked until the next arbitration is performed. For synchronous networks, this delay is a stage cycle. For asynchronous networks, this delay is likely a clock cycle, which corresponds to only a fraction of a stage cycle. The increase of the network latency due to this delay is thus relatively small in asynchronous networks. Hence, the impact of the arbitration scheme on the performance of an asynchronous network is less than that of a synchronous network.

The impact of the arbitration scheme on the performance of an asynchronous network depends on the packet size, switch size, and input buffer size. If the packet size is large and the switch size and input buffer size are small, the number of requests to be arbitrated in a clock cycle is small. For some practical values of these parameters, our simulations show that the quality of arbitration of a “poor” arbiter is approximately the same as that of a “good” arbiter. As the packet size decreases and the switch size and input buffer size increase, the difference in the quality of arbitration between a “poor” arbiter and a “good” arbiter increases since there are more requests to be arbitrated in each clock cycle. However, even if the packet size is small and the switch size and buffer size are large, the difference in the quality of arbitration between a “good” arbiter and a “poor” arbiter is less significant than in synchronous networks.

Three factors should be considered in designing a crossbar arbiter for VLSI communication switches: quality of arbitration, circuit speed, and circuit size. The
circuit sizes of the different arbiters considered are approximately the same. The circuit speeds of STSA and WWFA are approximately the same and faster than the other designs. For synchronous networks, either the wave front arbiter or the wrapped wave front arbiter should be considered for their superior quality of arbitration relative to other arbiter designs. The quality of arbitration of the wave front arbiter is slightly better than that of the wrapped wave front arbiter. If using the wave front arbiter rather than the wrapped wave front arbiter does not require a slower clock, the wave front arbiter is the best choice. Otherwise, the wrapped wave front arbiter should be used.

For asynchronous networks, there is a much smaller difference in the quality of arbitration of different arbiters for a wide range of packet sizes, switch sizes, and input buffer sizes. If the packet size is large and the switch size and input buffer size are small, there is no difference which arbiter to use since their quality of arbitration and circuit size are approximately the same. If the packet size is small and the switch size and input buffer are large, either the wave front arbiter or the wrapped wave front arbiter should be used in order to exploit their better quality of arbitration. If the circuit speed of the arbiter has an impact on the network performance, the wrapped wave front should be considered.
Chapter Four
Decomposed Arbiters for Large Crossbars

Crossbars are key components of communication switches used to construct multiprocessor interconnection networks. Small networks can be implemented as a single crossbar while large networks are composed of many small crossbars [Dias81a, Dias81b, Kuma84, Dall86, Bhuy87, Lang82a]. Theoretically, throughput is maximized and latency is minimized if the entire network is one large crossbar. In general, given a choice between a small number of large crossbars or a large number of small crossbars, larger crossbar switches help lower the probability of conflicts and allow packets to traverse the network in fewer hops. With advances in VLSI fabrication and packaging technology, larger crossbars are becoming practical (e.g., the Inmos C104 packet-routing switch has 32 inputs and 32 outputs). A key factor in determining the performance of crossbars for communication is the mechanism used to arbitrate conflicting requests.

Several packets arriving at different input ports of the switch simultaneously may be destined to the same output port. However, only one packet at a time may be forwarded through each output port. Due to the resulting contention for output ports, packets may have to be buffered at the inputs of the crossbar while awaiting service. Recent communication switch designs utilize multi-queue buffers, which maximize performance by allowing packets at an input port to be processed in non-FIFO order [McMi80, Kuma84, Tami88a, Dall90]. Packets at each input port, which are destined to different output ports, may be forwarded through the switch in any order. Hence, each input port contends for multiple output ports but needs
only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. The arbitration task is thus symmetrical with respect to inputs and outputs. Since the arbitration result for each port is dependent on the arbitration for other ports, the crossbar arbitration for switches with multi-queue buffers is more complicated than for switches with FIFO buffers. We have described in Chapter 3 high-speed symmetric crossbar arbiters, which efficiently solve the arbitration problem for multi-queue buffers.

One of the major challenges with large crossbar design is the long arbitration delay. Since the arbitration delay grows with the crossbar size, the arbitration speed of a large crossbar can become a performance bottleneck. The design of symmetric arbiters for large crossbars is the topic of this chapter. It will be shown that high performance can be achieved by decomposing the arbitration process so that the arbitration of a large crossbar is performed using an array of smaller arbiters. Furthermore, in a large crossbar it is not necessary for the number of queues at each input port to be equal to the number of output ports. A small number of queues reduces the cost of buffer management and is sufficient for achieving high performance.

The next section discusses the possible impact of increasing the crossbar size on the network performance. Section 4.2 describes how to decompose the arbitration for large crossbars in order to improve the network performance. The analysis results for the network latency in an unloaded network are presented. In Section 4.3, simulations are used to evaluate different arbitration schemes for large crossbars in loaded networks. Section 4.4 describes the logic and circuit design of the proposed decomposed arbiter. The performance impact of varying the number
of queues per input buffer is discussed in Section 4.5.

4.1. Performance Impact of Large Crossbars

For interconnecting a fixed number of nodes, larger crossbars lower the probability of conflicts and reduce the number of hops for a packet to traverse the network. Hence, theoretically, larger crossbars improve the performance of the interconnection network. However, the crossbar arbitration delay increases as the crossbar size increases. Thus, if large crossbars are used, arbitration delays can become a critical performance bottleneck.

To study the speed of crossbar arbitration, the CMOS implementation of 2×2, 4×4, and 8×8 crossbars and their wave front arbiters and wrapped wave front arbiters have been laid out using the MOSIS scalable design rules. With 2μ technology, circuit simulations indicate that the worst case delays for reaching an arbitration result for 2×2, 4×4, and 8×8 wave front arbiters are 7.2 ns, 15.5 ns, and 32 ns respectively. For the wrapped wave front arbiters, the worse case delays are 4.5 ns, 10 ns, and 20 ns respectively. In general, for the WFA and WWFA arbiters, the arbitration delay grows linearly with the crossbar size.

If the arbitration must complete in one clock cycle, long arbitration delays for large crossbars may force the use of a slower clock. In order to overcome this problem, the arbiter may be allowed several clock cycles to complete the arbitration. In this scheme, new requests are blocked during the arbitration. In the following, the network latency is analyzed for this multi-cycle arbitration when the network is unloaded. We discuss how the arbitration delay can affect the network performance. This multi-cycle arbitration scheme is called the nondecomposed
arbitration (as opposed to the decomposed arbitration to be presented in Section 4.2). For comparison, an unrealistic ideal arbitration scheme which can arbitrate a crossbar of arbitrary size in one cycle is also analyzed.

We analyze the performance of an arbitration scheme in terms of base network latency, which is the latency for a packet traversing an unloaded (or empty) network [Agar91]. The latency for a packet going through an unloaded crossbar switch is called base switch latency. In the analysis, the base switch latency consists of two components: the arbitration-related delay and the non-arbitration-related delay. The arbitration-related delay includes the wait a packet has to stay in the buffer before entering the arbitration, and the arbitration time. The non-arbitration-related delay includes the time for receiving the packet, making routing decision, and transmitting the packet. Typically, the non-arbitration-related delay is approximately independent of the crossbar size, and thus a constant will be assumed in our analysis.

Consider a \(2^w \times 2^w\) multi-stage interconnection network (e.g. an Omega network) composed of \(2^s \times 2^s\) crossbar switches. Suppose that one cycle is needed to arbitrate a \(2^u \times 2^u\) array of crosspoints for the nondecomposed scheme. This array of crosspoints is called the \(1\)-cycle subarray. Since the switch arbitration delay is proportional to the switch size, the switch arbitration delay is

\[
d = 2^s / 2^u = 2^{s-u}.
\]

When a packet arrives in an unloaded crossbar switch, in the best case the wait for the packet to enter the arbitration is 0 cycle (the \(d\)-cycle arbitration is going to start). In the worst case, the packet has to wait \(d-1\) cycles in order to enter the arbitration (the \(d\)-cycle arbitration started a cycle ago). On the average, a packet waits \(\frac{1}{2}(d-1)\) cycles. Assume the non-arbitration-related delay
of the switch latency is $k$ cycles. The base switch latency is thus $k + \frac{1}{2}(d-1) + d$.

Since it takes $w/s$ hops for a packet to traverse the network, the base network latency is $[k + \frac{1}{2}(d-1) + d] \cdot (w/s)$, namely, $[k + \frac{1}{2}(3 \cdot 2^{s-n} - 1)] \cdot (w/s)$.

For the ideal scheme, the arbitration takes one cycle regardless of the crossbar size. Hence, the base switch latency is $k + 1$, and the base network latency is $(k + 1) \cdot (w/s)$.

![Figure 4.1: Base network latency vs. switch size of a 4096×4096 network for the ideal arbitration and the nondecomposed arbitration. 1-cycle subarray size 2×2. The non-arbitration-related component of the switch latency is 4 cycles.](image)

The results of the above analysis of the base network latency for the ideal arbitration scheme and the nondecomposed arbitration scheme are shown in Figure 4.1 and Table 4.1. The network size is 4096×4096. It is assumed that the 1-cycle subarray size is 2×2. The non-arbitration-related delay of the switch
Table 4.1: Analysis of the base network latency of a 4096×4096 network for the ideal arbitration and the nondecomposed arbitration. 1-cycle subarray size 2×2. The non-arbitration-related component of the switch latency is 4 cycles.

<table>
<thead>
<tr>
<th>Switch Size</th>
<th>Base Network Latency (4096×4096)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ideal</td>
<td>nondec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>60.0</td>
<td>60.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30.0</td>
<td>39.0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>20.0</td>
<td>38.0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15.0</td>
<td>46.5</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>10.0</td>
<td>103.0</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>5.0</td>
<td>3075.5</td>
<td></td>
</tr>
</tbody>
</table>

latency is 4 cycles [Tami88a, Fraz89]. Since in practice the switch size is usually a power of 2 for the efficiency of addressing and routing in the network, only these switch sizes are listed in Table 4.1. The figure and the table show that unlike the unrealistic ideal scheme, the practical nondecomposed scheme does not always reduce the base network latency as we increase the crossbar size. In Table 4.1, when the crossbar size is larger than 8×8 for the nondecomposed scheme, the base network latency starts to increase. It validates the assertion that the arbitration delay for a large crossbar can become a performance bottleneck. It can be derived that the optimal switch size for the nondecomposed scheme is 6×6 (the base network latency is minimum in the graph when the switch size equals 5.95). Note that different network sizes lead to the same conclusions since only $w$ in the equation is affected.
4.2. Decomposed Arbitration for Large Crossbars

In the multi-cycle nondecomposed scheme, new requests are blocked during the arbitration. Hence, it reduces throughput and increases latency. In order to improve the performance, the crossbar can be logically partitioned into subarrays, each of which is arbitrated in one cycle. Requests can be granted as soon as the arbitration of the relevant subarray is completed. This leads to a significant reduction in the average delay to granting a request. We call this scheme the *decomposed arbitration*.

Figure 4.2: A 6×6 decomposed arbiter with 2×2 subarrays.

Figure 4.2 shows the operation of the decomposed arbitration for a 6×6 crossbar. In the figure, the whole arbitration array is partitioned into 9 subarrays where each subarray is a 2×2 WWFA. With the simple WFA and WWFA,
conflicting requests are guaranteed to be on different “wave fronts” [Tami93]. The same principle is the key to the operation of the decomposed arbiter. Subarrays which are arbitrated in parallel, in the same clock cycle, have no inputs or outputs in common. Hence, it is guaranteed that requests which are arbitrated in different subarrays during each cycle cannot conflict. In the arbiter shown in Figure 4.2, three subarrays can be arbitrated during each cycle. For instance, subarrays <1,1>, <2,3>, and <3,2> may be arbitrated in the first cycle. Then subarrays <1,2>, <2,1>, and <3,3> are arbitrated in the second cycle, and <1,3>, <2,2>, and <3,1> are arbitrated in the third cycle. This allows requests to <1,1>, <2,3>, and <3,2> to be granted in the first cycle so that transmission can begin without waiting for the second cycle and the third cycle. Similarly, granted requests for <1,2>, <2,1>, and <3,3> can start transmitting after the second cycle is complete. A local circular shift register, which shifts every three cycles, is used for fairness within each subarray. Subarray arbitration is controlled by a global circular shift register which shifts every cycle. This global circular shift register indicates which “wrapped wave front” of the subarrays can be arbitrated in the current clock cycle. As we mentioned in Section 3.2, WWFA has a lower arbitration delay than WFA. Hence, WWFA is used for each arbitration subarray throughout this chapter.

To understand more precisely why decomposed arbitration outperforms nondecomposed arbitration, we consider a crossbar where nondecomposed arbitration takes \( d \) cycles. Two situations are discussed in the following: very light network load and heavy network load. When the network load is very light, with the nondecomposed scheme an arriving packet first has to wait, on the average,
\( \frac{1}{2}(d-1) \) cycles in order to enter the arbitration and then has to wait for another \( d \) cycles for the arbitration to complete. On the other hand, with the decomposed scheme under very light network load, the average wait is \( \frac{1}{2}(d-1) \) cycles to enter the arbitration and then one cycle for the arbitration to complete. When the network load is heavy, with the nondecomposed scheme, once a packet leaves an input port buffer, the port has to wait \( \frac{1}{2}(d-1) \) cycles on the average before entering the arbitration and then wait for another \( d \) cycles for the arbitration to complete. On the other hand, with the decomposed scheme under heavy load, once a packet leaves an input port buffer, some of the queues at the buffer may enter the arbitration immediately in the following cycle and may begin transmission after the one cycle arbitration of their subarray is complete. Hence, using the decomposed arbitration lowers the latency for a packet going through the crossbar switch under the above two scenarios, compared to the nondecomposed arbitration. Note that the performance difference between the decomposed arbitration and the nondecomposed arbitration is more significant with larger crossbars, since \( d \) increases as the crossbar size increases.

4.2.1. Analysis of Base Network Latency

The base network latency of the decomposed arbitration can be derived by using analysis similar to that in Section 4.1. Assume again that the network size is \( 2^w \times 2^w \), and the switch size is \( 2^s \times 2^s \). The number of hops for a packet to traverse the network is \( \frac{w}{s} \). Suppose \( 2^u \times 2^u \) is the 1-cycle subarray size. On the average, a packet waits \( \frac{1}{2}(d-1) \) cycles in order to enter the arbitration, where \( d \) equals \( 2^s-u \). After the wait for entering the arbitration, it takes one cycle for the arbitration of
the subarray to complete. Assuming $k$ is the non-arbitration-related delay of the switch latency, the base network latency is thus $[k + \frac{1}{2}(d-1) + 1] \cdot (w/s)$, which equals $[k + \frac{1}{2}(2^s - u + 1)] \cdot (w/s)$.

Long arbitration delay when the number of “agents” is high can be a problem even with conventional bus arbiters. A possible solution in this case is the use of tree-structured arbiters, whose delay grows logarithmically with the number of agents [Mudg87, Lang82a]. The existence of these logarithmic bus arbiters motivated us to compare our crossbar arbiter designs to a hypothetical crossbar arbiter whose delay grows logarithmically with the number of rows and columns. Although we use this logarithmic arbiter for comparison, we are not aware of any fast practical circuit that implements it. To analyze the base network latency of the hypothetical logarithmic arbitration, assume again that the switch size is $2^s \times 2^s$ and the 1-cycle subarray size is $2^u \times 2^u$. The arbitration delay for the hypothetical logarithmic scheme is then $\frac{s}{u}$. The average wait for a packet to enter the arbitration is $\frac{1}{2} \left( \frac{s}{u} - 1 \right)$. The base network latency for the hypothetical logarithmic scheme is $[k + \frac{1}{2} \left( \frac{s}{u} - 1 \right) + \frac{s}{u}] \cdot \frac{w}{s}$, which equals $[k + \frac{1}{2} \left( \frac{3s}{u} - 1 \right)] \cdot \frac{w}{s}$. This arbitration scheme is very expensive in terms of hardware cost if it is at all possible to implement.

The results of the analysis for the ideal, the nondecomposed, the decomposed, and the hypothetical logarithmic schemes are shown in Figure 4.3 and Table 4.2. It shows that like the nondecomposed scheme, the decomposed scheme does not always reduce the base network latency as we increase the crossbar size either. However, unlike the nondecomposed scheme, when the crossbar size increases from $8 \times 8$ to $16 \times 16$, the base network latency for the decomposed scheme still
improves. Furthermore, the decomposed scheme consistently outperforms the nondecomposed scheme. Their difference is especially significant when the switch size is large. Figure 4.3 also shows that unless the switch size is greater than 22, the decomposed scheme is superior to the hypothetical logarithmic scheme.

![Figure 4.3: Base network latency vs. switch size of a 4096×4096 network for the four arbitration schemes. 1-cycle subarray size 2×2. The non-arbitration-related component of the switch latency is 4 cycles.](image)

**Figure 4.3**: Base network latency vs. switch size of a 4096×4096 network for the four arbitration schemes. 1-cycle subarray size 2×2. The non-arbitration-related component of the switch latency is 4 cycles.

### 4.2.2. Switch Design Space for Optimal Networks

The decomposed arbitration scheme can be generalized such that each subarray can consist of more than one 1-cycle subarrays. This subarray is called the *arbitration subarray*. Assuming the 1-cycle subarray size is $2^u \times 2^u$ and the arbitration subarray size is $2^r \times 2^r$, it takes $2^{r-u}$ cycles to complete the arbitration of
Table 4.2: Analysis of the base network latency of a 4096×4096 network for the four arbitration schemes. 1-cycle subarray size 2×2. The non-arbitration-related component of the switch latency is 4 cycles.

<table>
<thead>
<tr>
<th>Switch Size</th>
<th>Base Network Latency (4096×4096)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ideal</td>
</tr>
<tr>
<td>2</td>
<td>60.0</td>
</tr>
<tr>
<td>4</td>
<td>30.0</td>
</tr>
<tr>
<td>8</td>
<td>20.0</td>
</tr>
<tr>
<td>16</td>
<td>15.0</td>
</tr>
<tr>
<td>64</td>
<td>10.0</td>
</tr>
<tr>
<td>4096</td>
<td>5.0</td>
</tr>
</tbody>
</table>

an arbitration subarray. The arbitration delay for the whole switch is \( d = 2^s - u \). On the average a packet waits \( \frac{1}{2}(d - 1) \) cycles in order to enter the arbitration, and the arbitration takes \( 2^{r-u} \) cycles to complete. Hence, the base network latency is \([k + \frac{1}{2}(2^s - u - 1) + 2^{r-u}] \cdot (w/s)\). Note that with \( r = s \), the scheme becomes the nondecomposed arbitration, while with \( r = u \), the scheme becomes the decomposed arbitration scheme we analyzed in Section 4.2.1, where the arbitration subarray is the 1-cycle subarray.

Figure 4.4 shows the analysis results of the generalized decomposed arbitration based on the above equation, assuming that the 1-cycle subarray is 2×2, and the network size 4096×4096. The switch size and the arbitration subarray size vary between 2×2 and 64×64, but the arbitration subarray size is always smaller than or equal to the switch size. The base network latency in the figure is an approximation. We assume the switch size and the arbitration subarray size can be real numbers. In reality, they have to be integers, and a typical practical network is usually composed of switches whose size is a power of 2 for the ease of
Figure 4.4: Analysis of base network latency for the generalized decomposed arbitration. Network size 4096×4096. Switch size $2^s \times 2^s$, $1 \leq s \leq 6$. Arbitration subarray size $2^r \times 2^r$, $1 \leq r \leq s$. The non-arbitration-related component of the switch latency is 4 cycles.

construction of the network. Furthermore, we also assume that the number of hops is $\frac{w}{s}$ for any switch size, including sizes for which $s$ is not an integer.

Basically, Figure 4.4 illustrates the design space of a given network, with the switch size and the arbitration subarray size as the design parameters. The curve above the diagonal represents the base network latency for the nondecomposed arbiter, as shown in Figure 4.1. The base network latency is the product of the switch latency ($[k + \frac{1}{2}(2^{s-u}-1)+2^{r-u}]$) and the number of hops ($\frac{w}{s}$). The switch latency includes the average arbitration wait, which is the time a packet has to wait before entering arbitration ($^\frac{1}{2}(2^{s-u}-1)^*$), and the subarray arbitration delay ($2^{r-u}$). For a fixed switch size, the base network latency decreases monotonically

107
Figure 4.5: Optimal switch size vs. arbitration subarray size. 1-cycle subarray size 2×2. The non-arbitration-related component of the switch latency is 4 cycles.

with decreasing subarray size since the subarray arbitration delay is decreased and the other factors (arbitration wait time and number of hops) are not affected. For a fixed subarray size, increasing the switch size decreases the number of hops, which reduces the network latency, while increasing the single switch latency, which increases the network latency. Hence, for each subarray size, there is a switch size for which the network latency is minimized.

Figure 4.5 shows the optimal switch size for each subarray size. The optimal switch size increases with increasing arbitration subarray size. The reason for this behavior is that the subarray arbitration delay component of the switch delay increases with increasing subarray size and is independent of the switch size. The switch size only impacts the arbitration wait delay, which is independent of subarray size. Hence, for a “large” arbitration subarray, the relative increase in
switch delay for a “‘unit’” increase in switch size is lower than for a “‘small’” arbitration subarray. On the other hand, the relative decrease in the number of hops for a “‘unit’” increase in switch size is independent of the arbitration subarray size. This means that with increasing subarray size the negative impact of increasing the switch size is reduced while the positive impact is unchanged. The result is that a larger switch size is optimal. Note that when the arbitration subarray is greater than 31.4, the optimal switch size is equal to the arbitration subarray size since the arbitration subarray cannot be larger than the switch.

Based on the assumptions in the analysis above, reducing the arbitration subarray size always reduces the total switch arbitration delay. Hence, the smallest possible arbitration subarray is optimal. However, in reality, reducing the subarray size can have negative implications. In particular, for a fixed switch size, reducing the subarray size implies that there are more arbitration subarray in the switch. This can lead to an increase in the complexity and size (chip area) of the arbitration circuitry. As a result, the total switch arbitration delay may increase as the subarray size is decreased. Hence, the smallest arbitration subarray may not be optimal. For example, consider an 8×8 crossbar. With 4×4 arbitration subarrays, there will be four circular shift registers, with a total of 16 shift register cells, needed for the priority rotation within the arbitration subarrays. With 2×2 arbitration subarrays, there will be 16 circular shift registers, with a total of 32 shift register cells, needed for the priority rotation within the arbitration subarrays. These additional circular shift register cells require extra chip area and thus additional delay for switch arbitration due to the longer wires. An analysis of the negative impact of reducing the subarray size for our decomposed arbiter design is
4.3. Performance Evaluation

The analysis of network latency in Section 4.2 is only valid when the network has no traffic. In order to evaluate the various arbitration schemes for networks under different traffic loads, an even-driven simulator is also used. Simulations were performed for single switches and multistage networks. The interconnection network is an Omega network. The arbitration schemes simulated include the ideal arbitration, the nondecomposed arbitration, the decomposed arbitration, and the hypothetical logarithmic arbitration. The size of both the 1-cycle subarray and the arbitration subarray is 2×2. The links and buffers are byte-wide and every cycle one byte can be read from a buffer, written to a buffer, or transferred through a link.

The simulations are based on traffic with the following properties: 1) packet size is evenly distributed between 8 and 32 bytes. 2) during each cycle there is an equal probability of generating a packet at each of the inputs. 3) packet destinations are uniformly distributed over the outputs. We again assume that the non-arbitration-related delay of the switch latency is 4 cycles. A packet requests the use of the crossbar three cycles after arriving at the switch. After the arbitration, one cycle is spent before the packet starts to leave the switch.

The performance measures used are the average latency, the normalized throughput, and the 99th percentile latency. The latency of a packet is the number of cycles that elapse from when the first byte of a packet generated at an input of the switch or network to when it leaves the switch or network. The normalized
throughput is the average number of bytes received by each output per clock cycle. The 99th percentile latency is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies). It is reported here as a measure of the fairness of the arbitration. Lack of fairness in the arbitration results in increasing the 99th percentile latency and increasing the difference between the 99th percentile and average latencies [Tami93]. Each simulation run is terminated after 70,000 packets have arrived at their destination outputs. Statistics were gathered only after 7,000 packets in order to remove start-up effects.

Single switch simulations were performed for 8×8, 16×16, and 32×32 switches. The buffer size for each input port is 128 bytes for all switch sizes. Figure 4.6 and Figure 4.7 show the performance impact of various arbitration schemes for an 8×8 switch, in terms of the average latency vs. the normalized throughput and the 99th percentile latency vs. the normalized throughput, respectively. Figures 4.8, 4.9, 4.10, and 4.11 show the results for the 16×16, and 32×32 switches. For all these switches, the decomposed arbitration outperforms the nondecomposed arbitration by achieving higher maximum throughput, as well as lower average latency and 99th percentile latency. The performance difference between the decomposed arbitration and the nondecomposed arbitration increases as the crossbar size increases, as we discussed in the analysis in Section 4.2. Furthermore, the performance difference between the decomposed arbitration and the ideal arbitration is limited, compared to the difference between the decomposed arbitration and the nondecomposed arbitration. The decomposed arbitration also performs better than the hypothetical logarithmic arbitration for 8×8 and 16×16 switches. For the 32×32 switch, although the decomposed arbitration has a higher
Figure 4.6: Average latency vs. normalized throughput of an 8×8 switch for various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 128 bytes.

Figure 4.7: 99th percentile latency vs. normalized throughput of an 8×8 switch for various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 128 bytes.
**Figure 4.8:** Average latency vs. normalized throughput of a 16×16 switch for various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 128 bytes.

**Figure 4.9:** 99th percentile latency vs. normalized throughput of a 16×16 switch for various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 128 bytes.
Figure 4.10: Average latency vs. normalized throughput of a 32×32 switch for various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 128 bytes.

Figure 4.11: 99th percentile latency vs. normalized throughput of a 32×32 switch for various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 128 bytes.
various
of
buffer
d 
input
bytes
switch
size
for
network
of

size
for
respectively.
schemes.
throughput

subarray
size is 2×2. The buffer size for each input is 64, 128, and 192 bytes for 2×2, 4×4, and 8×8 respectively.

Figure 4.12: Average latency vs. normalized throughput of a 64×64 network for various switch sizes and various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 64, 128, and 192 bytes for 2×2, 4×4, and 8×8 respectively.

99th
percentile

Latency

0
400
800
1200
1600

Throughput

0
0.2
0.4
0.6
0.8
1

2x2
4x4-ideal
4x4-dec
4x4-nondec
8x8-ideal
8x8-dec
8x8-nondec
8x8-hlog

Latency

2x2
4x4-ideal
4x4-dec
4x4-nondec
8x8-ideal
8x8-dec
8x8-nondec
8x8-hlog

Figure 4.13: 99\textsuperscript{th} percentile latency vs. normalized throughput of a 64×64 network for various switch sizes and various arbitration schemes. The subarray size is 2×2. The buffer size for each input is 64, 128, and 192 bytes for 2×2, 4×4, and 8×8 respectively.

115
base network latency than the hypothetical logarithmic arbitration (12.5 vs. 11.0),
the decomposed arbitration starts to perform better when the throughput is
moderate to high.

Multistage network simulations were performed for 64×64, and 256×256
networks. In the multistage network simulations, different input buffer sizes are
used for different switch sizes, such that we can compare the performance of the
networks with the same amount of buffer space in the whole network. The input
buffer sizes for 2×2, 4×4, 8×8, and 16×16 switches are 64, 128, 192, and 256 bytes,
respectively. The simulation results for a 64×64 network are shown in Figure 4.12
and Figure 4.13. Three different switch sizes (2×2, 4×4, and 8×8) and four
arbitration schemes (ideal, nondecomposed, decomposed, and hypothetical
logarithmic) are evaluated. The 64×64 large crossbar switch is not included here
due to the lack of enough computing power for its simulation. The figure shows
that, given the same switch size, the decomposed arbitration outperforms both the
nondecomposed arbitration and the hypothetical logarithmic arbitration, and is
close to the ideal arbitration. With the ideal arbitration, the performance steadily
improves as the switch size increases. However, with the nondecomposed
arbitration, the performance of the 8×8 switch is even worse than that of the 4×4
switch. On the other hand, when we increase the switch size from 4×4 to 8×8 for
the decomposed arbitration, the performance still improves.

Figure 4.14 and Figure 4.15 show the simulation results for a 256×256
network. The switch size varies from 2×2, 4×4, to 16×16. Again, the decomposed
arbitration outperforms the nondecomposed arbitration, and the performance
difference is particularly significant for using 16×16 switches. The performance of
Figure 4.14: Average latency vs. normalized throughput of a 256×256 network for various switch sizes and various arbitration schemes. The subarray size is 2x2. The buffer size for each input is 64, 128, and 192 bytes for 2×2, 4×4, and 8×8 respectively.

Figure 4.15: 99th percentile latency vs. normalized throughput of a 256×256 network for various switch sizes and various arbitration schemes. The subarray size is 2x2. The buffer size for each input is 64, 128, and 192 bytes for 2×2, 4×4, and 8×8 respectively.
the hypothetical arbitration is also not as good as the decomposed arbitration for 16×16 switches. With the ideal arbitration and the decomposed arbitration, the performance improves as we increase the switch size. In contrast, the performance of the nondecomposed arbitration for the 16×16 switch is worse than that for the 4×4 switch. The nondecomposed arbitration for the 16×16 switch even has higher average latency and 99\textsuperscript{th} latency than the 2×2 switch case unless the throughput is low.

It should be noted that for both the decomposed and nondecomposed schemes, performance under heavy load may be improved by using the length of the packet currently being forwarded from an input port to predict when the input port will be free and begin arbitration while the current transmission is still in progress. This technique will involve significant increase in hardware complexity and will not improve performance under light load.

4.4. Design of Decomposed Arbiters

The decomposed arbiter is composed of several subarrays. Each subarray is implemented as a wrapped wave front arbiter. Figure 4.16 shows the organization of a 2×2 WWFA [Tami93]. For each arbitration cell, the \(P\) (priority) input indicates whether the cell has the top priority. The \(XI, XO, YI,\) and \(YO\) signals correspond to the \(W, E, N,\) and \(S\) signals, respectively, of Figure 3.8. The circular shift register is used to rotate top priority among the wrapped diagonals of arbitration cells.

Figure 4.17 shows the organization of an example decomposed arbiter for a 4×4 crossbar. In this example the whole arbitration array is partitioned into four
Figure 4.16: Organization of a 2\times 2 wrapped wave front arbiter. This can be used as a subarray of a decomposed arbiter.

subarrays, and each subarray is a 2\times 2 WWFA. The detailed design of an arbitration cell is shown in Figure 4.18 and Figure 4.19. The \textit{OPB} (output port blocked) line is 0 when the output port is blocked. \textit{SAE} (subarray enable) is 1 when the associated subarray is enabled by a signal from the global circular shift register for subarray arbitration (Figure 4.17). \( R \) is the request line from a queue of the buffer and \( G \) is the grant line. \( P \) (priority) indicates if the cell gets the top priority. \( BT \) (buffer transmitting) signal is asserted by the buffer in order to reserve the crosspoint it is using while the packet is being transmitted. When \( BT \) is 1, only one \( R \) line from the buffer is asserted. \( CB \) (column busy) is a wired-OR line which connects all the cells in a column, indicating whether some buffer is using the column for an ongoing transmission and it should not be interrupted by
other buffer requests.

The circuitry in the right part of Figure 4.19 is identical to the circuitry of the arbitration cell described in Chapter 3 and implements the following Boolean equations:

\[
G = (RR \land \overline{OPB}) \land (YI \lor P) \land (XI \lor P) \\
YO = (YI \lor P) \land \overline{G} \\
XO = (XI \lor P) \land \overline{G}
\]
Figure 4.18: An arbitration cell.

Figure 4.19: Logic design of the arbitration cell for subarrays of decomposed arbiters.
The left part of the circuit is unique to the decomposed arbiter. The RR line is asserted when either the particular queue is currently forwarding a packet (R and BT are asserted and CB is pulled down), or when the column is idle, the subarray is enabled, and the queue has a packet ready for transmission to the column.

The design presented in this section is amenable for modular implementation where several arbiter subarrays are implemented on a single chip and multiple copies of this chip can be used to construct a large arbiter. A single chip with arbiter subarrays as well as their corresponding data part of the crossbar can be used as a flexible building block for crossbar switches.

To construct a nondecomposed crossbar switch whose size is multiple of the building block chip size, several chips can be connected in an array. The SAE inputs of each chip are set to 1. A decomposed arbiter can be constructed similarly but the SAE inputs of each chip are connected to their corresponding state latches of the global circular shift register. To provide a decomposed crossbar with the global circular shift register, latches can be included in each chip. Several latches in different chips are linked together to form the global circular shift register. The state of a latch is fed to the associated group of subarrays to indicate ‘‘the turn’’ of the group. Assume the crossbar size is $2^x \times 2^y$ and the subarray size is $2^u \times 2^u$. Then the switch arbitration delay is $d = 2^y / 2^u = 2^{y-u}$. Since the local circular shift register shifts every $d$ cycles the clock signal for the local circular shift register can be obtained from the latch output of one latch of the global circular shift register.
Switch Design Space

In Section 4.2.2 we indicate that with the smallest arbitration subarray size, the base network latency may not be optimal. In the following, we discuss how the arbiter circuit design discussed in this section affects the switch design space described in Section 4.2.2. The equation for calculating the base network latency for the generalized decomposed arbiter in Section 4.2.2 is modified to reflect the impact of the implementation of the arbiter.

In the design of our decomposed arbiter, each column has a wired-OR CB line connected to every arbitration cell in the column. The CB line indicates if any arbitration cell is using the column in the current cycle. After an arbitration subarray (size \(2^r \times 2^r\)) is arbitrated, the CB line has to be set to reflect the results of this arbitration, which will be used by the next arbitration. Hence, the time allotted for a complete \(2^r \times 2^r\) subarray arbitration must include the time to set the CB line. The delay for setting the CB line depends on the number of the arbitration cells \(2^s\) and the number of the circular shift register cells \(2^{s-r}\) in a column. For each arbitration cell, the gate inputs driven by the CB line and the length of the CB line through the arbitration cell add capacitances to the CB line. For each circular shift register cell, the length of the CB line through the register cell also adds capacitances to the CB line. These capacitances result in extra delay for the arbitration of a \(2^r \times 2^r\) subarray. If the switch size is fixed, this extra delay increases as the arbitration subarray size decreases, since the number of the circular shift register cells \(2^{s-r}\) increases.

As we decrease the arbitration subarray size, the CB line is set (pulled down) more times during the arbitration of the whole switch. Since the average
arbitration wait is half the total delay for arbitrating the whole switch, the switch latency increases due to more times for setting the $CB$ line. Furthermore, with the arbitration subarray size decreasing, the extra $CB$ line capacitance increases due to more shift register cells and thus causes longer delay for setting the $CB$ line. With both the effect of more times for setting the $CB$ line and the effect of longer delay for each $CB$ setting, the base network latency may no longer be the lowest with the smallest subarray size.

Assume the extra delay of setting the $CB$ line due to each arbitration cell is $\alpha$ and the extra delay due to each circular shift register cell is $\beta \cdot \alpha$, where $\alpha$ and $\beta$ are implementation-dependent parameters. The extra delay of setting the $CB$ line for each arbitration subarray is then $\alpha \cdot 2^s + \alpha \cdot \beta \cdot 2^{s-r}$. Hence, the delay for arbitrating a $2^r \times 2^r$ subarray is $h = 2^{r-u} + \alpha \cdot 2^s + \alpha \cdot \beta \cdot 2^{s-r} = 2^{r-u} + \alpha \cdot 2^s \cdot (1 + \beta \cdot 2^{-r})$. Since there are $2^{s-r}$ arbitration subarrays in each dimension of the arbiter, the average time a packet has to wait before entering the arbitration is $\frac{1}{2} \cdot 2^{s-r} \cdot h$. The resulting base network latency is thus $[k + \frac{1}{2} \cdot 2^{s-r} \cdot h + h] \cdot (w/s)$, which equals $[k + (2^{s-r-1} + 1) \cdot [2^{r-u} + \alpha \cdot 2^s \cdot (1 + \beta \cdot 2^{-r})]] \cdot (w/s)$. Note that the above derivation ignores the length of the clock cycle, and it is only an approximate delay calculation.

Based on the equation of the base network latency in the above, we can derive the optimal switch size $r$ from a given set of values for $k$, $w$, $s$, $u$, $\alpha$, and $\beta$. The reasonable range of the values for $\alpha$ and $\beta$ can be estimated as follows. The delay of each arbitration cell is approximately 3 to 4 gate delays according to the design in Figure 4.19. Assuming the 1-cycle subarray size is $2 \times 2$, the delay of each
1-cycle subarray is 6 to 8 gate delays. This delay corresponds to one time unit in the equation of the base network latency. Hence, the minimum size gate delay is on the order of 0.1 to 0.2 time units. As stated earlier, the number of arbitration cells and circular shift register cells per column determine the “fan-out” for the CB line pull-down transistor. Each arbitration cell is a load of approximately one gate plus the capacitance of the CB line segment across it. Hence, reasonable values for $\alpha$ are in the range of 0.01 to 0.2. The circular shift register cell is smaller than the arbitration cell and is not driven by the CB line. Hence, the extra delay for each shift register cell is smaller than for each arbitration cell and the value of $\beta$ is expected to be in the range of 0.1 to 0.5. As an example, we consider a network with the following parameters: $k = 4, w = 12, s = 4, u = 1$. For $0.02 \leq \alpha \leq 0.04$ and $0.1 \leq \beta \leq 0.5$, the optimal arbitration subarray size $r$ is 1. However, for $0.04 \leq \alpha \leq 0.1$ and $0.1 \leq \beta \leq 0.5$, the base network latency with $r = 2$ is lower than with $r = 1$. Hence, in the latter case, a network with arbitration subarray size of 4×4 will perform better than a network with subarray size of 2×2.

4.5. Reducing the Number of Queues

For a large crossbar, the cost of managing, at each input port, a separate queue for each output port may be prohibitive. The control registers can require significant chip area and the large number of these register will reduce circuit performance (larger decoders, longer buses, additional loads on buses, etc) [Fraz89]. This problem can be alleviated by reducing the number of queues in each buffer. The output ports are partitioned into several groups, and all the packets destined to the output ports in the same group share the same queue. For
example, consider an $n \times n$ switch, with $q$ queues in each input buffer. The output ports are partitioned into $q$ groups. A packet destined to output port $t$ is stored in queue $\lfloor qt/n \rfloor$. The crossbar arbiter design described in Section 4.4 can still be used here even though fewer queues are implemented in each buffer.

Reducing the number of queues in a buffer may allow a reduction in the number of wires for request and grant signals between the buffers and the arbiter. For example, if there are eight queues in a buffer for a $32 \times 32$ crossbar, then we need only three wires for the requests from each queue. One wire is used to indicate if there is a request from this queue, and the other two specify which crosspoint is requested. Only one grant wire is needed per queue. Hence, the total number of wires for request and grant signals between each buffer and the arbiter is 32. This is a significant reduction compared to the 64 wires that would be required if 32 queues were maintained in each buffer. With this scheme there is additional hardware required in the arbiter for decoding the lines carrying the number of the output port requested by each queue.

In order to determine the impact of varying the number of queues per input port, we have simulated a $32 \times 32$ switch and a $256 \times 256$ network composed of $16 \times 16$ switches. The arbitration is performed by a decomposed arbiter with $2 \times 2$ subarrays. The simulation parameters and performance measures are as described in Section 4.3. Figure 4.20 and Figure 4.21 show the simulation results for the $32 \times 32$ switch, and Figure 4.22 and Figure 4.23 show the simulation results for the $256 \times 256$ network. In the $32 \times 32$ single switch simulation, the number of queues per buffer can be reduced to 16 or even 8 while the performance is only degraded a little. In the simulation for the $256 \times 256$ network with $16 \times 16$ switches, when there
Figure 4.20: The impact of varying the number of queues per input for a 32×32 switch. Average latency vs. normalized throughput. The subarray size is 2×2. The buffer size for each input is 128 bytes.

Figure 4.21: The impact of varying the number of queues per input for a 32×32 switch. 99th percentile latency vs. normalized throughput. The subarray size is 2×2. The buffer size for each input is 128 bytes.
Figure 4.22: The impact of varying the number of queues per input for 16×16 switches in a 256×256 network. Average latency vs. normalized throughput. The subarray size is 2×2. The buffer size for each input is 128 bytes.

Figure 4.23: The impact of varying the number of queues per input for 16×16 switches in a 256×256 network. 99th percentile latency vs. normalized throughput. The subarray size is 2×2. The buffer size for each input is 128 bytes.
are 8 queues per buffer, the performance is still close to that with 16 queues. Note that with only one queue, the buffer degenerates into a FIFO buffer.

4.6. Summary and Conclusions

Theoretically, larger crossbars improve the network performance by lowering the probability of conflicts and reducing the number of hops for a packet to traverse the network. However, the time required to arbitrate conflicting requests to crossbar switches increases with the size of the crossbar. In reality, this long arbitration delay can become a critical performance bottleneck. We have proposed an arbiter design, which is called the decomposed arbiter, to solve the arbitration delay problem. By decomposing the arbitration process, some requests can be granted before the arbitration of the entire crossbar is complete. This leads to a significant reduction in the average delay to granting a request. Our analysis and simulation results show that decomposed arbiters effectively increase the bandwidth and reduce the latency for the interconnection network. The additional implementation cost compared to a nondecomposed scheme is minimal. This design lends itself to efficient modular implementation of large crossbars using small crossbar building blocks.

As originally designed, switches based on DAMQ buffers manage, at each input port, a separate queue for each output port. The circuit complexity for managing multiple queues in an input buffer increases as the number of queues is increased. Furthermore, with the original DAMQ design, the number of request and grant lines between each buffer and the arbiter increases linearly with the size of the switch. Based on these considerations, for a large crossbar, it is undesirable
to have at each input port a separate queue for each output port. We have shown that this problem can be alleviated by reducing the number of queues per input buffer. For large switches, even if the number of queues is reduced significantly, there is only a small reduction in performance compared to switches that use full DAMQ buffers.
Chapter Five

Starvation Prevention

Multi-queue input buffers lead to high performance in multiprocessor interconnection networks by allowing packets at an input port to be processed in non-FIFO order. Symmetric crossbar arbiters efficiently resolve conflicting requests in switches with multi-queue input buffers. While these arbiters lead to excellent performance in terms of throughput and average latency, they do not guarantee fairness. Hence, it is possible for an “unlucky” packet to be left in a switch buffer for a long time, potentially forever, while other packets are forwarded quickly through the switch.

This chapter describes and evaluates a technique for preventing such starvation situations. The viability of the technique is demonstrated by implementing it in VLSI. Simulations are used to evaluate the performance of starvation-free arbiters. Their performance is compared with that of the arbiters without starvation prevention under uniform traffic and nonuniform traffic.

In Section 5.1, the starvation problems with symmetric crossbar arbiters in synchronous networks and asynchronous networks are discussed. The design of starvation-free arbiters for both types of networks are presented. The performance impact of starvation prevention under uniform and nonuniform traffic is evaluated in Section 5.2. Section 5.3 describes the VLSI implementation of a starvation-free arbiter. This starvation-free arbiter is compared with an arbiter without the starvation prevention mechanism to measure the additional cost in terms of circuit delay and chip area.
5.1. Starvation Prevention for Crossbar Arbitration

Without a starvation prevention mechanism, communication switches with multi-queue input buffers cannot guarantee that a packet arriving at the switch will be forwarded to the appropriate output within a finite period of time. This is obvious if the top priority is fixed at a queue (arbitration cell) which is not the one holding the packet in question. Even if the top priority is rotated to a different cell every cycle, eventual transmission of the packet is not guaranteed. Specifically, although the queue containing the packet periodically gets the top priority, it is possible that whenever that happens the output port that corresponds to this queue is blocked (e.g., due to a full buffer in the next switch). Hence, low priority queues may be allowed to transmit their packets while the top priority queue remains blocked and priority is then shifted to another queue. Such starvation conditions may delay packets in an “unlucky” queue indefinitely. In this section, the starvation problems in synchronous networks and asynchronous networks are described and their solutions are presented respectively.

5.1.1. Starvation Prevention in Synchronous Networks

Starvation prevention in synchronous networks is relatively simple [Tami93]. In a synchronous network, packets are transmitted and received by the communication switches in lock-step. The steps are called stage cycles [Yoon90]. Assuming no contention, an entire packet is transmitted from one switch to its neighbor in a single stage cycle. No packet is “in transit” across a stage cycle boundary so all the resources assigned by the crossbar arbiter — input ports and output ports — are freed at the end of each stage cycle. In such networks, each
queue in an \( n \times n \) communication switch has the top priority every \( n^2 \) stage cycles if a WFA arbiter is used. If the output ports are never blocked, it is guaranteed that no packet will wait in a queue more than \( bn^2 \) cycles, assuming \( b \) is the number of packet slots in an input buffer. However, since output ports can be blocked due to full buffers in the following stage, it is possible for an “unlucky” queue to be prevented from transmission every time it has the top priority. Hence, packets in this unlucky queue remains in the switch indefinitely.

In synchronous networks, the WFA arbiter can guarantee prevention of starvation if a nonempty top priority queue maintains its top priority until it succeeds in sending one packet. We call a WFA arbiter that uses this scheme to rotate the priority, a round-robin (RR) arbiter. An RR arbiter can be implemented by not shifting the horizontal \( XP \) shift register if there is a top priority request but it is rejected (see Figure 3.32).

With the WWFA arbiter, the starvation prevention mechanism is slightly more complex, because the wrapped diagonal with the top priority corresponds to \( n \) queues. It does not solve the starvation problem that we shift the priority when \textit{any} request in the top priority wrapped diagonal is granted. With this scheme, it is possible that the required output port from an unlucky queue is always blocked whenever the top priority visits. On the other hand, it does not solve the starvation problem either that we shift the priority only when \textit{all} requests in the top priority wrapped diagonal are granted. With this scheme, the top priority can stay with the same wrapped diagonal indefinitely if all the requests cannot be granted at the same time.

Figure 5.1 shows a design for achieving starvation prevention for the WWFA
Figure 5.1: Organization of a $2 \times 2$ starvation-free wrapped wave front arbiter in synchronous networks.

arbiter in synchronous networks. In addition to the WP token ring, the SP (starvation priority) token ring is used to point to a cell in the top priority wrapped diagonal. The SP token ring is a circular shift register which shifts every time the WP token has traversed the whole WP token ring a round. Each cell in the same wrapped diagonal is pointed to by the SP token ring “in turn” when the top priority visits the wrapped diagonal. Shifting in the WP token ring is decided by the cell pointed by the SP token ring in the top priority wrapped diagonal. If this cell has a request but is rejected due to output port blocking, the WP token ring does not shift until the request is granted. Since any output port cannot be blocked forever, eventually this cell will be granted and the WP token ring will shift. This ensures starvation prevention for each queue.
5.1.2. Starvation Prevention in Asynchronous Networks

Starvation prevention in asynchronous networks is more difficult than in synchronous networks. In asynchronous networks it takes multiple clock cycles to transmit an entire packet from one switch to its neighbor. However, packets may arrive at a switch during any clock cycle and may be of variable length. Thus, different packets utilize input and output ports for transmission for different durations. Once an entire packet is transmitted to a neighbor switch, the crossbar row and column it has been using become available to another packet. Hence, the resources assigned by the crossbar arbiter are not all freed simultaneously so that they can be assigned at will to the top priority queue. It is possible for the top priority queue to be starved indefinitely since the row and column that it needs are never free at the same time.

Figure 5.2 demonstrates a possible scenario of starvation in an asynchronous network with multi-queue buffers. Double squares at a crossbar point indicate that the corresponding queue has a request. Queue (3,2) is starved despite the fact that it maintains the top priority since row 3 and column 2 are never available at the same cycle. When row 3 is released by some other queue in the same row, column 2 is still being used by another queue in the same column. Hence, the arbiter allocates row 3 to another requesting queue in the same row before column 2 is released. Similarly, when column 2 is released by some other queue in the same column, row 3 is still being used by another queue in the same row. Hence, column 2 is allocated to another requesting queue in the same column before row 3 is released.

For an asynchronous network, a starvation-free arbiter can be implemented by
Figure 5.2: An example of starvation for symmetric crossbar arbitration. Double squares at a crossbar point indicate that the corresponding queue has a request. The requesting queue (3,2) keeps being rejected because row 3 and column 2 are occupied by other queues and are not both available at the same cycle for queue (3,2).

modifying the RR arbiter discussed above so that it may reserve a resource required by the top priority queue even if the other resource required by this queue is not available. Specifically, in addition to stopping the priority rotation, the top priority queue prevents the assignment of the row and column it needs to other queues until its request is granted. If one resource (either the row or the column)
becomes available, the arbiter does not allocate this resource to any other requesting queues. Since the other resource is eventually freed, the request from the top priority queue can be granted then. This starvation-free arbitration scheme is called the symmetric greedy reservation (SGR) arbiter. Note that while a row is unavailable only if another queue is using it, a column is unavailable if either another queue is using it or the corresponding output port is currently blocked.

Since some resources are kept idle while reserved, the reservation mechanism for an SGR arbiter may cause performance degradation. To alleviate this problem, a starvation counter can be incorporated in the arbiter. The purpose of the starvation counter is to check if the requesting top priority queue is “near starvation.” If the top priority request has been rejected for a certain number of clock cycles (starvation count threshold), the arbiter starts to reserve the two required resources. Otherwise, the arbiter operates as an RR arbiter. An SGR arbiter with a starvation count threshold equal to $k$ is denoted as an SGR-$k$ arbiter.

Although both the row and the column are needed for the top priority queue to be granted, reserving one resource is sufficient for resolving the starvation problem. A row-greedy reservation (RGR) arbiter is similar to an SGR arbiter except that only the row can be reserved. A column is never reserved and whenever it is freed it is assigned to a queue that can use it immediately. However, a row can be reserved by a top priority queue. After the row has been reserved, the top priority queue obtains and keeps the row the moment it becomes available. Eventually the column required by the top priority queue becomes available, and then the requesting top priority queue can obtain both the row and the column. Likewise, we can reserve the column only. This arbiter is called a column-greedy
reservations (CGR) arbiter.

5.2. Performance Evaluation

We have evaluated the performance of the starvation-free arbiters under uniform traffic and nonuniform traffic for single switches and multistage interconnection networks using the event-driven simulator described in Section 3.3.4. The network is an asynchronous network. Links and buffers are assumed to be byte wide. Each buffer has independent read and write ports and can store 128 bytes. A byte can be read from the buffer, written into the buffer, or transferred through a link in one cycle. Virtual cut-through switching [Kerm79] is used and the minimum delay for a packet going through the communication switch is five cycles [Tami92]. The simulations use packet sizes which are uniformly distributed between 8 and 32 bytes. Senders generate packets and send them to the network. The interval between packet creation follows a geometric distribution. However, a packet is sent into the network only if the corresponding input buffer has room for the entire packet. Otherwise, the sender blocks.

The latency of a packet is the number of cycles that elapse from when the first byte of a packet is generated in the sender to when it leaves the network, including the wait for the blocked network input. The normalized throughput is the average number of bytes received by each network output per clock cycle. Each simulation run is terminated after at least 50,000 packets have arrived at their destination outputs. Statistics were gathered only after 10,000 packets in order to remove start-up effects [Mitr82].

The starvation-free arbiters are compared to two arbiters that do not provide
guaranteed packet delivery. One of these two is the round-robin (RR) arbiter, discussed earlier. The other is the oblivious round-robin (ORR) arbiter, which is a WFA arbiter that shifts the priority every cycle. Our simulation studies indicate that the arbitration schemes based on the WFA arbiter and those based on the WWFA arbiter achieve approximately the same performance. Hence, only the results for the arbitration schemes based on the WFA arbiter are shown in the evaluation.

5.2.1. Uniform Traffic

Under uniform traffic load, packet destinations are uniformly distributed over all the network outputs. Figure 5.3 shows how different arbiters impact the performance of a 4×4 switch under uniform traffic, in terms of average latency vs. normalized throughput. The arbiters compared include the ORR arbiter, the RR arbiter, and the SGR arbiters with starvation count threshold equal to 32, 8, and 0. The results indicate that the reservation mechanism used by the SGR arbiters results in a small performance degradation, showing a lower maximum throughput and a higher average latency. A high starvation count threshold minimizes this performance degradation. Note that an RR arbiter is equivalent to an SGR-∞ arbiter. Figure 5.4 shows the performance impact of these different arbiters on the 99th latency. The simulation indicates that the SGR arbiters also have a higher 99th latency than the ORR and RR arbiters. It appears that under uniform traffic, the reservation mechanism does not improve the 99th latency. Instead, the 99th latency is increased a little.

The comparison of the SGR arbiters and the CGR arbiters is shown in
**Figure 5.3:** Average latency vs. normalized throughput of a 4×4 switch for the different arbiters under uniform traffic.

**Figure 5.4:** 99th percentile latency vs. normalized throughput of a 4×4 switch for the different arbiters under uniform traffic.
Figure 5.5. The RGR-\(k\) arbiters are omitted here since our simulation studies show that the performance of an RGR-\(k\) arbiter is the same as that of a corresponding CGR-\(k\) arbiter. The simulation results in the figure indicate that an SGR-\(k\) arbiter performs slightly better than a corresponding CGR-\(k\) arbiter. The rest of the chapter focuses on the SGR arbiters.

![Figure 5.5: Comparison of SGR arbiters and CGR arbiters. Average latency vs. normalized throughput of a 4×4 switch under uniform traffic.](image)

We also consider whether the results of these simulations are different if switch sizes other than 4×4 are used. Figure 5.6 and Figure 5.7 show the simulation results for a 2×2 switch and an 8×8 switch, respectively, under uniform traffic. As shown in the figures, the performance degradation caused by the starvation prevention mechanism is larger for the 2×2 switch and smaller for the 8×8 switch, compared to the 4×4 switch. The reason for this is that when a row or a column is reserved for the top priority queue, the number of resources the arbiter can allocate...
Figure 5.6: Average latency vs. normalized throughput of a 2x2 switch for the different arbiters under uniform traffic.

Figure 5.7: Average latency vs. normalized throughput of an 8x8 switch for the different arbiters under uniform traffic.
Figure 5.8: Average latency vs. normalized throughput of a 16×16 network for the different arbiters under uniform traffic.

Figure 5.9: 99th percentile latency vs. overall throughput of a 16×16 network for the different arbiters under uniform traffic.
to the other queues is reduced. For example, when a row is reserved for a 4×4 switch, only three rows and four columns can be used. As the switch size increases, the fraction of the reserved resources over the total resources becomes smaller. Specifically, when a resource is reserved, this reserved resource represents a 25%, 12.5%, or 8.25% decrease of the total resources for a 2×2, 4×4, or 8×8 switch, respectively.

Simulation results for a 16×16 network with two stages of 4×4 switches are shown in Figure 5.8 and Figure 5.9. The same conclusions can be derived from the network simulations as from single switch simulations. The SGR arbiters achieve starvation freedom at the cost of a small performance degradation, and this performance degradation decreases as the starvation count threshold increases. Note that under uniform traffic, the reservation mechanism increases the 99th percentile latency a little for the network rather than decreases it, similar to the results for single switches.

5.2.2. Nonuniform Traffic

One of the goals of our simulation studies was to evaluate the performance of the starvation prevention mechanism under nonuniform traffic patterns that could cause some packets to be “stuck” in the switch for an inordinately long time. In the following, several nonuniform traffic patterns are used to study this behavior. We consider a single 4×4 switch in Section 5.2.2.1, and then a 16×16 network with two stages of 4×4 switches in Section 5.2.2.2.
Figure 5.10: A nonuniform traffic pattern for a 4×4 switch. The numbers associated with each crosspoint (or each queue) are the probability distribution of the destinations of the packets in the same buffer for the corresponding output ports.

5.2.2.1. Simulations for a 4×4 Switch

An extreme case of a nonuniform traffic pattern for a 4×4 switch is shown in Figure 5.10. In the figure, the numbers at each crosspoint are the probability distribution of the destinations for packets arriving at the input port. The destinations of the packets arriving at input port 1 are uniformly distributed over the output ports. The packets arriving at input ports 2, 3, and 4 are all destined to a “hot spot” — output port 2. With this traffic pattern, we divide the non-empty queues into three different groups: unfavored, column contenders, and row contenders. The unfavored group consists of only queue (1,2). Queue (1,2) is unfavored because it is the only queue that receives packets and has to compete for two resources in order to send these packets. The column contenders group consists of queues (2,1), (3,1), and (4,1). The queues in the column contenders group only compete for an output port (column). The row contenders group
consists of queues (1,1), (1,3), and (1,4). The queues in the row contenders group only compete for an input port (row).

To evaluate the performance for different queue groups under nonuniform traffic, we use overall throughput, queue throughput and queue latency as performance measures. Overall throughput is the average normalized throughput over all the outputs (different outputs can have different normalized throughputs in the nonuniform traffic case). Queue throughput is the average number of bytes transmitted from a defined group of queues. Queue latency is the average latency of the packets which are transmitted from a defined group of queues.

**Figure 5.11:** The impact of starvation prevention on the queue throughput of different queue groups for the traffic pattern in Figure 5.10. U includes queue (1,2) only; C queues (2,2), (3,2), and (4,2); R queues (1,1), (1,3), and (1,4).

Using the traffic pattern from Figure 5.10, Figure 5.11 shows the performance comparison between the ORR arbiter and the SGR-8 arbiter in terms of queue
Figure 5.12: The impact of starvation prevention on the queue latency of different queue groups for the traffic pattern in Figure 5.10. U includes queue (1,2) only; C queues (2,2), (3,2), and (4,2); R queues (1,1), (1,3), and (1,4).

Figure 5.13: The performance impact of different arbiters on queue group U under the traffic pattern in Figure 5.10.
throughput vs. overall throughput for three groups of queues. Figure 5.12 shows the performance comparison in terms of queue latency vs. overall throughput. The SGR-8 achieves a higher maximum overall throughput than the ORR arbiter (0.403 vs. 0.355). Furthermore, the results indicate that the reservation mechanism for a starvation-free arbiter has little impact on groups C and R. However, under high throughput, it significantly reduces the queue latency of queue (1,2).

Figures 5.13, 5.14, and 5.15 show queue latency vs. queue throughput — the latency and throughput shown are only for the specific queue group in question. However, in all three figures the applied load from all the senders is varied uniformly to obtain these results.

Figure 5.13 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.10. A significant reduction in latency is achieved by changing the priority shifting scheme from ORR to RR. However, the reservation mechanism of the SGR arbiters leads to additional reductions in latency. As the starvation count threshold is decreased, the queue latency decreases since the reservation mechanism is triggered earlier.

Figure 5.14 shows the performance impact of the different arbiters on queues (1,1), (1,3), and (1,4) under the traffic pattern in Figure 5.10. The ORR arbiter achieves the lowest maximum queue throughput. The reason for this is that with ORR queue (1,2) is more likely to be starved, causing the row 1 input buffer to fill up with packets destined for output 2. In this case, no sender 1 packets destined to the other outputs can enter the switch. The SGR arbiters can achieve a slightly higher queue throughput than the RR arbiter since they are more likely to prevent input buffer 1 from filling up with queue (1,2) packets. This effect increases with
Figure 5.14: The performance impact of different arbiters on queue group R under the traffic pattern in Figure 5.10.

Figure 5.15: The performance impact of different arbiters on queue group C under the traffic pattern in Figure 5.10.
decreasing starvation count threshold since the reservation mechanism is triggered earlier. On the other hand, for a given queue throughput, decreasing the starvation count threshold increases the latency since the reservation mechanism is more likely to be triggered, allowing transmission from queue (1,2) and delaying packets in the other queues of input 1.

Figure 5.15 shows the performance impact of the different arbiters on queues (2,2), (3,2), and (4,2) under the traffic pattern in Figure 5.10. It might be expected that queues (2,2), (3,2), and (4,2) would exhibit poorer performance for starvation-free arbiters due to the reservation of column 2 for queue (1,2). Figure 5.15 shows that the maximum achievable queue throughput is the same for all the arbiters. However, as shown in Figure 5.11, the maximum queue throughput for this group (group C) is not achieved when the overall switch throughput is maximized. Instead, the maximum queue throughput for group C is achieved at a point where the reservation mechanism has no impact on the queue throughput for queue (1,2). As the applied load is increased above this point, the group C throughput actually decreases. This decrease is most pronounced with the SGR-0 arbiter precisely because this arbiter reserves column 2 for queue (1,2) most often.

Figure 5.16 shows average latency vs. overall throughput under the traffic pattern in Figure 5.10 for all the packets, regardless of which inputs they arrive and which outputs they leave. Figure 5.17 shows the results for 99th percentile latency vs. overall throughput. Unlike the results under uniform traffic, the ORR arbiter has the highest average latency and 99th latency when the overall throughput is high. Changing the priority shifting scheme from ORR to RR reduces the average latency and the 99th latency. The reservation mechanism of the SGR arbiters leads
Figure 5.16: Average latency vs. overall throughput of a 4×4 switch for the different arbiters under the traffic pattern in Figure 5.10.

Figure 5.17: 99th percentile latency vs. overall throughput of a 4×4 switch for the different arbiters under the traffic pattern in Figure 5.10.
Figure 5.18: A nonuniform traffic pattern for a 4×4 switch. The numbers associated with each crosspoint are the probability distribution of the destinations of the packets in the same buffer for the corresponding output ports.

Figure 5.19: The performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.18. Queue latency vs. queue throughput.
Figure 5.20: A nonuniform traffic pattern for a 4x4 switch. The numbers associated with each crosspoint are the probability distribution of the destinations of the packets in the same buffer for the corresponding output ports.

Figure 5.21: The performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.20. Queue latency vs. queue throughput.
to further reduction, particularly in 99th latency.

Figure 5.18 shows a nonuniform traffic pattern similar to Figure 5.10 but the packets arriving at input ports 2, 3, and 4 are not all destined to output port 2. Only 55% of these packets are destined to output port 2, and the destinations of the other 45% of the packets arriving at inputs 2, 3, and 4 are evenly distributed over outputs 1, 3, and 4. Figure 5.19 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.18. The simulation results indicate that performance improvement provided by the reservation mechanism for queue (1,2) is still significant.

Figure 5.20 shows another nonuniform traffic pattern where 40% of the packets arriving at inputs 2, 3, and 4 are destined to output port 2, and the destinations of the other 60% are evenly distributed over outputs 1, 3, and 4. Figure 5.21 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.20. In this case, the performance difference between the different arbiters is very small for queue (1,2). This indicates that as the percentage of the packets destined to the hot spot from input ports 2, 3, and 4 decreases, the latency for queue (1,2) decreases since it is easier for queue (1,2) to obtain column 2.

5.2.2.2. Simulations for a 16x16 Network

Figure 5.22 shows a nonuniform traffic pattern for a 16x16 network. The network consists of two stages of 4x4 switches. In this traffic pattern, 80% of the packets generated at senders 4, 8, and 12 are destined to a hot spot — receiver 1. The destinations of the other 20% of the packets are uniformly distributed over all
the network outputs. The other 13 senders generate packets whose destinations are uniformly distributed over all the network outputs. With this traffic pattern we divide the first-stage switches into two classes: switches handling uniform traffic patterns and switches handling nonuniform traffic patterns. For the switches handling uniform traffic patterns, all the inputs are injected from identical senders.

**Figure 5.22:** A nonuniform traffic pattern for a 16×16 network using 4×4 switches. 80% of the packets generated at senders 4, 8, and 12 are destined to receiver 1. The destinations of the other 20% of the packets are uniformly distributed over all the network outputs. All the packets generated at the senders other than 4, 8, and 12 are uniformly distributed over all the network outputs.
with uniform traffic. Only switch (1,1) belongs to this category. The switches handling nonuniform traffic are injected with traffic from two different types of senders. For each one of these switches, three of the inputs are injected from identical senders with uniform traffic. The fourth input is injected with traffic from a sender generating the nonuniform traffic as previously described. Switches (2,1), (3,1), and (4,1) belong to this category. Based on the traffic generated and the connected first-stage switches, we can partition the senders into three groups. Senders which generate uniform traffic and are connected to the switches handling uniform traffic patterns are called UU senders. This group includes senders 0, 1, 2, and 3. Senders which generate nonuniform traffic and are connected to the switches handling nonuniform traffic patterns are called NN senders. This group includes senders 4, 8, and 12. Senders which generate uniform traffic and are connected to the switches handling nonuniform traffic patterns are called UN senders. To measure the performance of each group of senders in the following, we define sender throughput as the average normalized throughput over all the senders in a group. Sender latency is the average latency of the packets transmitted from a defined group of senders.

Using the traffic pattern from Figure 5.22, Figure 5.23 and Figure 5.24 show the performance difference between the three sender groups in terms of the sender throughput vs. overall throughput and the sender latency vs. overall throughput, respectively. The arbitration schemes simulated include the ORR, RR, and SGR-8 arbiters. When the overall throughput is under 0.3, different groups have the same sender throughput. When the overall throughput is above 0.3, the network starts to saturate and the sender throughputs of different groups are distinguished from each
Figure 5.23: Sender throughput vs. overall throughput of a 16\times16 network for the ORR, RR, and SGR-8 arbiters under the traffic pattern in Figure 5.22.

Figure 5.24: Sender throughput vs. overall throughput of a 16\times16 network for the ORR, RR, and SGR-8 arbiters under the traffic pattern in Figure 5.22.
other. The NN senders have the lowest sender throughput and the highest sender latency since 80% of their packets are destined to the hot spot. As the overall throughput increases above the 0.3 point, the sender throughput of the NN senders is reduced due to tree saturation [Pfis85b]. Packets generated by the NN senders and destined to receivers other than the hot spot are blocked at the senders, since in the first-stage switches, the input buffers are full of the packets destined to the hot spot. For the UU senders and UN senders, the sender throughputs continue to increase as the overall throughput increases above the 0.3 point. The UU senders have a higher sender throughput and a lower sender latency than the UN senders. The reason for this is that for the packets generated by the UU senders, only those destined to receivers 0 to 3 have to traverse a switch handling nonuniform traffic (switch (1,2)), while all the packets generated by the UN senders have to traverse one or two switches handling nonuniform traffic.

Figure 5.23 and Figure 5.24 also show the impact of the arbitration scheme on the performance of three different groups of senders. For the NN senders, the performance of the RR arbiter is slightly better than that of the ORR arbiter, achieving approximately the same sender throughput and slightly lower sender latency. However, the performance of the SGR-8 arbiter for the NN senders is slightly worse than that of the ORR arbiter, with the SGR-8 arbiter achieving a little lower sender throughput and slightly higher sender latency. For the UN and UU senders, the RR arbiter performs considerably better than the ORR arbiter, achieving a higher sender throughput and a significantly lower sender latency. The performance of the SGR-8 arbiter for the UN and UU senders is even better than that of the RR arbiter, with the SGR-8 achieving a higher sender throughput and a
**Figure 5.25:** Average latency vs. overall throughput of a 16×16 network for the different arbiters under the traffic pattern in Figure 5.22.

**Figure 5.26:** 99th percentile latency vs. overall throughput of a 16×16 network for the different arbiters under the traffic pattern in Figure 5.22.
lower sender latency. Overall, the fair priority rotation mechanism used by the RR arbiter improves the performance of the UN and UU senders without reducing the performance of the NN senders. The reservation mechanism used by the SGR-8 arbiter improves the performance of the UN and UU senders at a cost of a small performance degradation of the NN senders, in addition to guaranteeing starvation freedom. These results indicate that by using a fairness mechanism, the RR and SGR-8 arbiters reduce the negative performance impact of the NN senders on the “innocent” UN and UU senders.

Figure 5.25 and Figure 5.26 show the average latency vs. overall throughput and the 99th percentile latency vs. overall throughput, respectively. Unlike the results for a 16×16 network under uniform traffic (see Figure 5.8 and Figure 5.9), the ORR arbiter has the lowest overall throughput. At the same overall throughput, the average latency and the 99th percentile latency for the ORR arbiter are also the highest. Changing the priority shifting scheme from ORR to RR increases the overall throughput, and decreases the average latency and the 99th percentile latency. With the reservation mechanism, the SGR arbiters further improve the network performance, in addition to guaranteeing starvation freedom.

5.3. Implementation

We have shown that starvation prevention can be achieved by adding a reservation mechanism to the symmetric crossbar arbiters. Its performance impact has been evaluated under uniform and nonuniform traffic. Under uniform traffic, the starvation-free arbiters is outperformed by the arbiters without starvation prevention. However, the performance difference is very small with a large
starvation count threshold. Under nonuniform traffic, the starvation-free arbiters can significantly improve the performance for queues inclined to be starved. In order to demonstrate the viability of the design of the starvation-free arbiters, we have laid out a WFA-based 4×4 SGR-32 arbiter in custom VLSI. In the SGR arbiters, the capability of starvation prevention is efficiently implemented owing to the regular structure of the symmetric crossbar arbiters. To reserve the row and the column, the top priority arbitration cell simply “disables” the wave front output signals in horizontal and vertical directions (see Figure 5.27).

The basic structure of a WFA-based SGR-k arbiter is similar to the WFA in Figure 3.32, which is shown again in Figure 5.27 for convenience. The detailed inputs and outputs of the arbitration cell for an SGR-k arbiter are shown in Figure 5.28-(a). Figure 5.28-(b) shows the starvation counter used for the starvation count threshold for the whole arbitration array. Signals \(X_P, Y_P, X_I, X_O, Y_I, Y_O, R,\) and \(G\) have been described in Chapter 3. \(QT\) (queue transmitting) comes from the input buffer and there is a separate \(QT\) for each arbitration cell. When \(QT\) is 1, it indicates that the corresponding queue is still transmitting a packet through the crosspoint, so the associated row and column should not be grabbed by any other queues in the same row or column. There is an \(RA\) (row available) for each row. \(RA\) is generated by a NOR gate whose inputs are the \(QT\) signals in the corresponding row. Similar to \(RA\), there is a \(CA\) (column available) signal for each column. However, the inputs of the NOR gate for generating \(CA\) include \(OPB\) as well as the \(QTs\) in the corresponding column.

Each arbitration cell generates a \(TPR\) (top priority rejected) signal, which becomes high if the cell has the top priority and the request is rejected. There is an
Figure 5.27: Organization of a $2 \times 2$ SGR-$k$ arbiter.

Figure 5.28: An SGR-$k$ arbiter. (a) Inputs and outputs of the arbitration cell. (b) Starvation counter of the whole arbitration array.
OR tree for the whole arbitration array. All the TPRs are the inputs of this OR tree. The output of the OR tree is \( GTPR \) (global top priority rejected), which is sent to the starvation counter. This \( GTPR \) indicates if the top priority request is rejected regardless of where the top priority is. If \( GTPR \) is 1, the starvation counter is incremented by 1 until it reaches the starvation count threshold. When the starvation counter reaches the starvation count threshold, \( RSV \) (reservation) becomes 1. The \( RSV \) signal is broadcast to every arbitration cell to inform the top priority cell if it should start to reserve the required row and column. This reservation can be achieved by simply disabling the \( XO \) and \( YO \) outputs. The detailed circuit of the arbitration cell is shown in Figure 5.29.

Figure 5.30 shows the floorplan of the WFA-based SGR-32 arbiter. The size of the boxes in the figure is proportional to the size of the modules in the layout. The layout of the crossbar is 990\( \lambda \) wide by 380\( \lambda \) tall, while the layout of the arbiter is 1050\( \lambda \) wide by 930\( \lambda \) tall, including the circular shift registers and the starvation counter. SPICE circuit simulations using 2\( \mu \) CMOS technology indicate that the worst case delay for the arbiter is 23 ns. A WWFA-based SGR-32 arbiter has been laid out too, and the worst case delay for the arbiter is 15.5 ns.

In order to measure the cost of adding the starvation prevention mechanism, a WFA-based 4x4 RR arbiter has also been laid out. The layout area of this RR arbiter is 1010\( \lambda \) wide by 870\( \lambda \) tall. The worst case delay for the RR arbiter is 20 ns. Hence, if the arbitration delay is part of the critical path of the switch, adding the starvation prevention mechanism does have a small performance penalty which was not taken into account in the previous section.
Figure 5.29: Detailed logic design of the arbitration cell.

Figure 5.30: The floorplan of a 4x4 crossbar of eight-bit wide buses with the SGR-32 arbiter. The modules are drawn to scale.
5.4. Summary and Conclusions

While our proposed symmetric crossbar arbiters lead to excellent performance in terms of throughput and average latency, they do not guarantee fairness. Hence, it is possible for an “unlucky” packet to be left in a switch buffer for a long time, potentially forever, while other packets are forwarded quickly through the switch. In a synchronous network, starvation prevention is relatively simple. In an asynchronous network where the communication switches use multi-queue input buffers, eventual packet delivery cannot be guaranteed unless special starvation prevention mechanisms are employed.

We have introduced and evaluated a reservation mechanism which guarantees starvation freedom in an asynchronous network. Simulations show that the proposed starvation-free arbiters cause a minor performance degradation for uniform traffic. However, under certain nonuniform traffic patterns the starvation-free arbiters may outperform arbiters that lack a starvation prevention mechanism. To evaluate the implementation cost of the starvation prevention mechanism, we have laid out in custom VLSI wave front arbiters with and without the starvation prevention mechanism. The overhead of the mechanism was found to be 11% in terms of layout area and, in the worst case, 15% in terms of arbiter circuit performance.
Chapter Six
Arbiter Support for Adaptive Routing
and for High-Priority Traffic

This chapter describes the arbiter designs that support other types of traffic in the interconnection network. Arbiters for supporting adaptive routing and high-priority traffic are presented. Adaptive routing can be used to improve the performance of the interconnection network and provide fault tolerance. In a communication switch with adaptive routing capability, a packet can have more than one destined output. Packets arriving at the inputs can adapt to the congested or faulty area in the network and decide which output they should be destined to in order to avoid the area. In Section 6.1, we discuss arbiter designs which can be used in adaptive routing switches. We describe how our symmetric crossbar arbiters can be applied to the arbiter design for adaptive routing switches with conventional FIFO input buffers. In some multiprocessors and multicomputers, high-priority traffic is used to support real-time applications. Section 6.2 discusses how to design a crossbar arbiter which can support high-priority traffic. Arbiter designs with two levels of packet priorities are described.

6.1. Arbiter Design Supporting Adaptive Routing

Many interconnection networks use deterministic routing to transmit packets from the source node to the destination node. With deterministic routing, a packet can only follow the pre-determined path in the network to get to the destination. If some portion of the path is heavily loaded or faulty, the packet cannot circumvent
the congested or faulty area. This can cause poor network performance or failure to deliver messages. Figure 6.1 shows an example of the deterministic dimension-order routing from the source node (5,1) to the destination node (2,5) in a mesh network. The packets sent from the source node to the destination node follow a fixed path by reducing the relative address in X dimension to 0 and then reducing the relative address in Y dimension to 0.

![Figure 6.1: A path using deterministic dimension-order routing. The path is fixed and determined by the addresses of the source node and the destination node.](image)

Adaptive routing has been proposed to overcome the problems with deterministic routing and achieve performance improvement and fault tolerance of interconnection networks [Ngai89, Dall93, Chow87, Chen90, Kons90b]. In general, adaptive routing can be classified into minimal routing and non-minimal routing. With minimal adaptive routing, packets can only be routed to switches
which are one hop closer to the destination node. Figure 6.2 shows an example of the minimal adaptive routing from the source node (5,1) to the destination node (2,5) in a mesh network. Along the path, the link connecting nodes (5,3) and (5,4) and the link connecting nodes (4,4) and (4,5) are heavily loaded or faulty. With non-minimal adaptive routing, packets can be misrouted farther away from their destination in the presence of congestion or faulty components. Figure 6.3 shows an example of the non-minimal adaptive routing in a mesh network. Along the path, the link connecting nodes (5,4) and (4,4) and the link connecting nodes (5,4) and (5,5) are heavily loaded or faulty. In this example, packets are misrouted at node (5,4) to avoid the problem links.

Figure 6.2: A path using minimal adaptive routing. The link connecting nodes (5,3) and (5,4) and the link between nodes (4,4) and (4,5) are heavily loaded or faulty.

Symmetric crossbar arbiters can be applied to the switch design supporting
Figure 6.3: A path using non-minimal adaptive routing. The link connecting nodes (5,4) and (4,4) and the link connecting nodes (5,4) and (5,5) are heavily loaded or faulty.

adaptive routing. Since each packet can have more than one destined output, the multi-queue buffers which partition the buffer space based on the single destined output of each packet are inappropriate here. Hence, FIFO buffers are used at the input ports and only packets at the heads of the FIFO buffers can submit requests to the crossbar arbiter. Figure 6.4 shows an example of arbitration for an adaptive routing switch. The routing decision circuit at each input determines the destined outputs for each packet arriving in the input port. The arbitration decision is made after the routing decision. The arbiter gathers the information of the destined outputs generated by all the routing decision circuits at the input ports and globally resolves the contention of the requests.

Figure 6.5 shows the arbitration model of the example in Figure 6.4. In the
**Buffers**  

**Crossbar Arbitration**

**Figure 6.4:** An example of arbitration for an adaptive routing switch. The destined outputs of packets A, B, C, and D are \{3,4\}, \{2\}, \{1,3\}, and \{1,2\}, respectively. Single circles are denied requests after the arbitration. Double circles are granted requests.

In some systems, the adaptive routing strategy employed provides cost information about each route. In order to optimize performance for a particular packet, at each switch that packet should be routed to the most ‘‘profitable’’ output, so that the packet will end up traversing the least cost route to its destination. However, routing packets only through the most profitable outputs, based on static cost information, does not result in optimal system performance.
Specifically, if only the most profitable outputs are considered, there may be contention for output ports that will result in blocking of some packets which could have been routed towards their destinations via less profitable outputs. System performance can be maximized if all the profitable outputs for each requesting packet are considered in the arbitration process. Ideally, the arbiter should arbitrate requests in a sequence of phases, incrementally trying less desirable outputs if the packet does not get access to the most profitable (desirable) switch outputs. However, such an arbiter would be too slow and complex. A practical alternative is an arbiter that divides all the possible outputs for a packet into only two classes: preferred outputs and acceptable outputs. The arbiter arbitrates for the preferred outputs in the first phase and then arbitrates for the acceptable outputs in the second phase.

The symmetric crossbar arbiters described in Chapter 3 can be used for the two-phase scheme described above. Both phases are performed by the same
circuit. Figure 6.6 shows an example of the two-phase arbitration in an adaptive routing switch. For any design of a crossbar arbiter, there is an interface circuit between the buffers and the arbiter. The interface circuit keeps track of which rows and columns are busy (granted) and blocks additional requests for busy rows and columns. In the two-phase arbiter design, this interface circuit must maintain the busy-row and busy-column information between the two phases. After grants in the first phase have been given to the preferred requests in the first phase, the acceptable requests for the rows and columns granted in the first phase are blocked by the interface circuit.

**Figure 6.6:** An example of two-phase arbitration in an adaptive routing switch. Single circles are denied preferred requests after the arbitration. Double circles are granted preferred requests. Single diamonds are denied acceptable requests. Double diamonds are granted acceptable requests.

The two-phase arbitration can be completed in either one clock cycle or two clock cycles. With the two clock cycles design, each arbitration phase corresponds to a separate clock cycle. The interface circuit must ensure that in the first cycle only the preferred requests of every packet are considered. The disadvantage of
this design is that it takes two cycles to grant an acceptable request for which the resources are available in the first cycle. Alternatively, the one clock cycle design can be used, where each arbitration phase is completed in one clock phase of a clock cycle. With this one clock cycle design, the circuit of the two-phase arbiter has to meet the timing constraints of the switch. The interface circuit which keeps track of the busy-row and busy-column information must be fast enough to latch the grant signals generated in the first phase.

The two-phase arbiter design can be used to support non-minimal routing. For example, in a switch with a non-minimal routing scheme the preferred outputs can be those which are connected to the switches that are one hop closer to the destination. For packets at the heads of the full buffers, the acceptable outputs are those connected to switches that are farther away from the destination. In such a switch design, the packet at the head of a buffer is misrouted only if all the preferred outputs are busy and the buffer is full. With this scheme, no packets can be blocked and deadlock is prevented [Kons90a]. The same two-phase arbiter design can also be used to support minimal routing if there is a significant difference in the cost of multiple outputs connected to switches that are one hop closer to the destination.

6.2. Arbiter Design Supporting High-Priority Traffic

As discussed in Section 2.3.2, communication switches for interconnection networks may have to support high-priority traffic for real-time applications. Various buffer designs supporting high-priority traffic have been studied, and it has been shown that a separate queue for high-priority packets in each input buffer is
the most cost-effective scheme [Tami88b, Tami92b]. To support this buffering scheme, the arbiter must be able to arbitrate high-priority requests as well as regular requests. This section describes arbiter designs which support high-priority traffic.

The two-phase arbiter design we described in Section 6.1 can be used to support high-priority traffic. In the first phase, only the requests from high-priority packets are arbitrated, and in the second phase the requests from regular packets are arbitrated. The operation of the arbitration for the high priority requests in the first phase is the conventional FIFO arbitration. In each row, only one high-priority request is asserted since there is only one high-priority queue in each buffer. The second phase is the symmetric crossbar arbitration. The wrapped wave front arbiter can be used here for both phases since the conventional FIFO arbitration is a special case of the symmetric crossbar arbitration. The interface circuit we described in Section 6.1 keeps track of the row-busy and column-busy information. This is to ensure that the regular requests that are in the same rows and columns with those high-priority requests granted in the first phase cannot join in the second phase.

An alternative to the two-phase arbitration is to overlap the arbitration of the high-priority requests and the regular requests in a single phase. This can be done in an arbiter where each arbitration cell performs two independent functions: it is a cell in a FIFO arbitration operation and it is a cell in a wave front arbitration operation. Figure 6.7 shows such an arbiter. The arbitration wave for the high-priority requests starts from the top row moving vertically toward the bottom row. The arbitration wave for the regular requests starts from the cell at the left top
Figure 6.7: An arbiter supporting high-priority traffic. A cell with a circle and a square indicates that the high-priority request for the corresponding crosspoint is asserted. A cell with double squares indicates that the regular request for the corresponding crosspoint is asserted. There is no request asserted for the cell with one square. The grants are given to those shaded cells after the arbitration is complete.

corner moving diagonally toward the cell at the right bottom corner. For each cell, the arbitration wave for the high-priority requests must traverse the cell before the arbitration wave for the regular requests.

The arbiter that performs the combined high-priority and regular arbitration is a combinational circuit consisting of an \( n \times n \) array of arbitration cells. Each cell receives two request signals from the corresponding buffer. One is the regular request and the other is the high-priority request. Since there is only one high-priority queue from each buffer, no more than one high-priority request is asserted in each row. Interface circuitry between the buffers and the arbiter must block any regular request requiring a column for which there are are high-priority requests.
This can be implemented by a wired-OR line, one per column, in the interface circuit. For each column, the interface circuit blocks the regular requests if the wired-OR line is pulled down by any asserted high-priority request signal. This wired-OR line is a dynamic circuit, and thus the delay should be much smaller than the delay for arbitrating a cell. In the arbiter, a broadcast line for each row is used to prevent the regular requests from being granted if the high-priority request in the same row is granted. This broadcast line is the output of a NOR gate whose inputs are the high-priority grant signals from all the cells in the row. If this line is 0, it indicates the row has been granted to a high-priority request and is not available to any regular requests.

The influence of the high-priority arbitration on the regular arbitration is through the wired-OR lines of the interface circuit and the row-busy broadcast lines. Assuming the cell delay is $T$ time units, the arbitration result for the high-priority requests for the top row is stable after the first $T$ time units. Since the delay of the wired-OR line of the interface circuits is smaller than the cell delay, the regular requests to the cells are all stable after the first $T$ time units. If the delay for the row-busy broadcast line is ignored, all the inputs to the regular arbitration of the first diagonal are stable after the first $T$ time units. Hence, the arbitration of the first diagonal is complete after the first $2T$ time units. The high-priority arbitration result of row $i$ is stable after the first $iT$ time units, and thus the arbitration of diagonal $i$ is complete after $(i+1)T$ time units. Since there are $2n - 1$ diagonals in the array, the arbitration delay of the entire array is $2nT$ time units. If the delay of the row-busy broadcast line is taken into account and this delay is $\delta$ time units, then the arbitration of the first diagonal is complete after the first $2T + \delta$ time units. The
arbitration of diagonal $i$ is complete after the first $(i+1)T+\delta$ time units. Hence, the arbitration delay of the entire array becomes $2nT+\delta$ time units.

![Arbitration Cell Diagram](image)

**Figure 6.8:** The arbitration cell supporting high-priority traffic.

Figure 6.8 shows the configuration of an arbitration cell for this design. $R$ is the regular request signal from the corresponding regular queue of the buffer. $HR$ is the request signal from the high-priority queue of the buffer. If $HR$ is 1, then $R$ has to be 0. If $HR$ is 0, then $R$ can be either 1 or 0. There is only one grant signal $G$ needed since the buffer is able to determine which request the grant is for. When $HR$ is 1, $G$ indicates if the high priority request is granted. When $R$ is 1, $G$ indicates if the regular request is granted. $XP$ and $YP$ indicate if the cell has the top priority. $HYI$ and $HYO$ are the propagation input and output signals in the high-priority arbitration. $XI$ and $XO$ are the propagation input and output signals for the row in the regular arbitration. $YI$ and $YO$ are the propagation input and output signals for the column in the regular arbitration. $OPB$ is the signal that
indicates if the output port requested is blocked. $HG$ is the high-priority grant which is fed to a NOR gate for generating the broadcast signal. $RGH$ is the broadcast signal that indicates if the row has been granted to a high-priority request. If $RGH$ is 1, the high-priority request in the row is not granted and the row is available to the regular requests.

![Logic design of the arbitration cell supporting high-priority traffic.](image)

**Figure 6.9:** Logic design of the arbitration cell supporting high-priority traffic.

Figure 6.9 shows the logic design of the arbitration cell. The function of the arbitration cell can be expressed in the following logic equations:

$$HG = (HYI \lor YP) \land HR \land \overline{OPB}$$

$$HYO = HYI \land \overline{HG}$$

$$G = ((R \land RGH \land \overline{OPB}) \land (YI \lor YP) \land (XI \lor XP)) \lor HG$$

$$YO = (YI \lor YP) \land \overline{G}$$

$$XO = (XI \lor XP) \land \overline{G}$$

While the two-phase arbiter in Section 6.1 can be used here for supporting
high-priority traffic, the single-phase arbiter design described above cannot be used to support the adaptive routing with two classes of requests in Section 6.1. It is possible to design a single-phase arbiter where each arbitration cell can arbitrate both preferred requests and acceptable requests in the same phase. However, unlike the single-phase arbiter for supporting high-priority traffic, the acceptable requests should not be blocked just because there are preferred requests for the same column. In the single-phase arbiter for supporting high-priority traffic, the high-priority arbitration is a FIFO arbitration. If there is any high-priority request for a particular column, it is guaranteed that there is a high-priority grant for that column. Hence, if there are any high-priority requests for a column, any regular requests for that column should be blocked. For the arbitration for adaptive routing with two classes of requests, the arbitration of the preferred requests is a symmetric crossbar arbitration. Hence, there is the possibility that there are several requests for a particular column but none of them are granted. Thus, despite the fact that there are preferred requests for a column, it ends up being available for acceptable requests. Hence, the arbitration consists of two phases, and the phase for the preferred requests has to be complete before the phase for the acceptable requests starts.

The circuit speeds for the two-phase arbiter and the single-phase arbiter are approximately the same. We assume that the arbitration cell delay in both arbiters is the same ($T$ time units). The wired-OR delay in the interface circuit and the delay for the row-busy broadcast signal in the single-phase arbiter are small compared to the cell delay. Hence, the arbiter delay of both designs is $2nT$.

The circuit of the two-phase arbiter is simpler than that of the single-phase
arbiter. The two-phase arbiter can use the same arbitration cell of the symmetric crossbar arbiter, while the arbitration cell in the single-phase arbiter is different and has more circuitry. A key disadvantage of the two-phase arbiter is the need to latch the grant signals generated in the first phase for use in the second phase. If each phase is performed in a different clock cycle, this is not a problem. In this case, the two-phase arbiter simply uses the same interface circuit that the single-level single-phase arbiter (Chapter 3) uses for maintaining the row-busy and column-busy information between cycles. If both phases of the arbiter are performed in the same cycle, the requirement to latch the results of the first phase may reduce the arbitration speed since the setup and hold time requirements of the latch must be met. The first-order approximation presented earlier indicated that the arbitration delays of the single-phase and two-phase arbiters are the same. However, the single-phase arbiter avoids the latching of signals in the middle of its operation. Hence, it is expected that the single phase design will be faster.
Chapter Seven

Summary

Significant improvements in the performance of communication switches can be achieved by using multi-queue input buffers instead of FIFO buffers. A crossbar is often used to connect multi-queue input buffers to the output ports of the switch and an arbiter is needed to resolve conflicting requests for crossbar resources from the various queues. If all the queues of each input buffer are connected to a single crossbar input, the arbitration task is symmetric with respect to inputs and outputs. Since the result of the arbitration for each crosspoint depends on or influences the arbitration of all the crosspoints in the corresponding row and column, the arbiter is relatively complex and cannot be partitioned into independent row or column arbiters.

We have evaluated different symmetric crossbar arbitration schemes using static probabilistic analysis, single switch event-driven simulations, and simulations of a buffered Omega network. We have shown that in synchronous networks, a good symmetric crossbar arbitration scheme is essential to realizing the potential for performance improvement from multi-queue buffers. In particular, a poor scheme may result in lower performance than with FIFO buffers while a good scheme can increase the maximum throughput of the network by more than 40%. As the buffer size and switch size increase, the benefits of a good arbitration scheme increases.

Two arbitration schemes, which are based on the propagation of an arbitration wave across an array of arbitration cells, were shown to achieve nearly the same
performance as the complex theoretical schemes. The proposed wave front and wrapped wave front arbiters are amenable to simple regular implementation in VLSI. Circuit simulation of our CMOS implementation of the 4×4 wave front arbiter, using 2 µ technology, indicates that, for a 4×4 crossbar, this arbiter can produce a valid and efficient crossbar configuration in 15.5 ns.

For asynchronous networks, the choice among the various symmetric crossbar arbiters has less impact on performance than for synchronous networks. The impact of the arbitration scheme on the performance of an asynchronous network depends on the packet size, switch size, and input buffer size. If the packet size is large and the switch size and input buffer size are small, the quality of arbitration of a poor arbiter is approximately the same as that of a good arbiter. As the packet size decreases and the switch size and input buffer size increase, the difference in the quality of arbitration between a poor arbiter and a good arbiter increases. However, even if the packet size is small and the switch size and buffer size are large, the difference in the quality of arbitration between a good arbiter and a poor arbiter is less significant than in synchronous networks. Since the differences among the various arbiters in terms of quality of arbitration is small, in many situations circuit speed and circuit size should be the deciding factors when choosing an arbiter.

With the advances in VLSI fabrication and packaging technology, larger crossbars are becoming practical. However, the time required to arbitrate conflicting requests to crossbar switches increases with the size of the crossbar. We have shown that by decomposing the arbitration process, the crossbar bandwidth can be increased while the average latency is reduced. The additional
implementation cost compared to a nondecomposed scheme is minimal. This
design lends itself to efficient modular implementation of large crossbars using
small crossbar building blocks.

Switches based on DAMQ buffers manage, at each input port, a separate
queue for each output port. The circuit complexity for managing multiple queues
in an input buffer increases as the number of queues is increased. Furthermore,
with the original DAMQ design, the number of request and grant lines between
each buffer and the arbiter increases linearly with the size of the switch. Based on
these considerations, for a large crossbar, it is undesirable to have at each input port
a separate queue for each output port. We have shown that this problem can be
alleviated by reducing the number of queues per input buffer. For large switches
with limited input buffer size, even if the number of queues is reduced
significantly, there is only a small reduction in performance compared to switches
that use full DAMQ buffers.

While high bandwidth and low latency are critical requirements from
interconnection networks, fairness and a guarantee of timely delivery of every
packet must also be provided. In an asynchronous network where the
communication switches use multi-queue input buffers, eventual packet delivery
cannot be guaranteed unless special starvation prevention mechanisms are
employed. We have presented a simple starvation prevention mechanism for the
wave front arbiter used in such communication switches. We have laid out in
custom VLSI wave front arbiters with and without the starvation prevention
mechanism. The overhead of the mechanism was found to be 11% in terms of
layout area and, in the worst case, 15% in terms of arbiter circuit performance.
The mechanism presented does guarantee eventual packet delivery. Furthermore, simulation studies have shown that, under certain nonuniform traffic patterns, the mechanism can improve performance for packets in what would otherwise be a particularly slow queue at a cost of a relatively small decrease in performance for packets in some other queues.

We have discussed the arbiter designs which can support different types of network traffic. One of the issues is the arbiter support for adaptive routing. Adaptive routing can be used to improve network performance and provide fault tolerance in multiprocessor interconnection networks. With adaptive routing, the interconnection network achieves better link utilization and fault tolerance by routing packets around congested or faulty areas. We have shown that the proposed symmetric crossbar arbiters can be used to support adaptive routing in communication switches efficiently. We have also discussed the issue of arbiter support in the communication switch for high-priority traffic in interconnection networks. A crossbar arbiter based on DAMQ buffers with dedicated high-priority queues is described. We showed that such an arbiter can be efficiently implemented by modifying the wave front arbiter.
Bibliography


[Dall88] W. J. Dally, “Fine-Grain Message Passing Concurrent


T. N. Mudge, J. P. Hayes, and D. C. Winsor, “Multiple Bus


