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

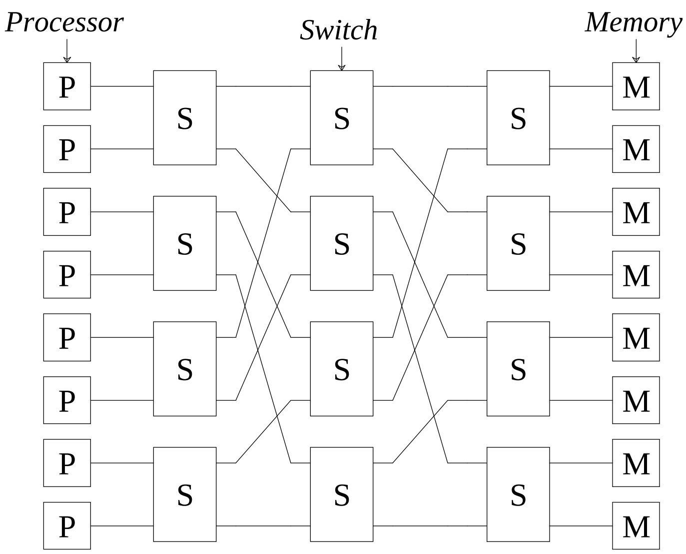

Figure 1.1: An indirect network for multiprocessors

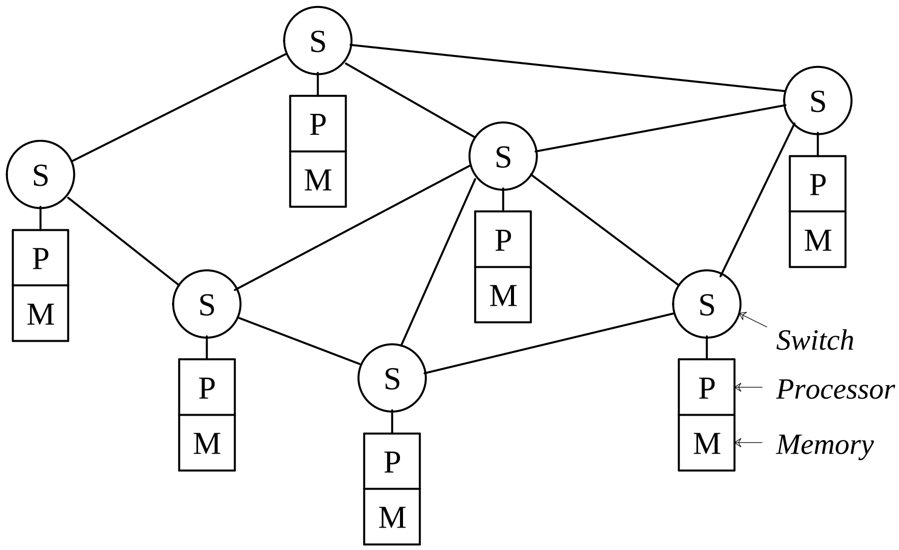

Figure 1.2: A direct network for multicomputers

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. *Arbiters* 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.

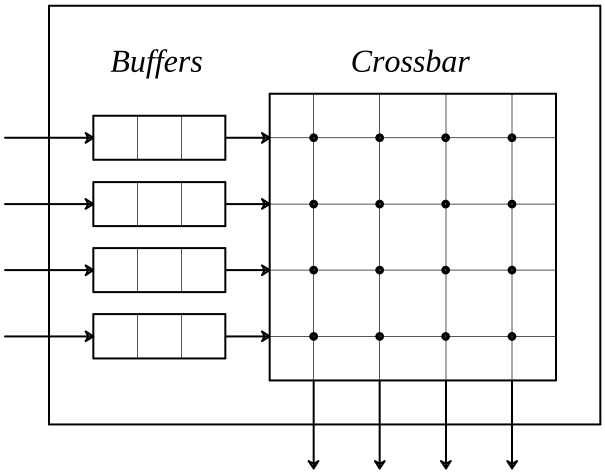

Figure 1.3: Organization of a typical communication switch

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 (**Com**munication **Co**processor **B**uilding-**B**lock) 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.

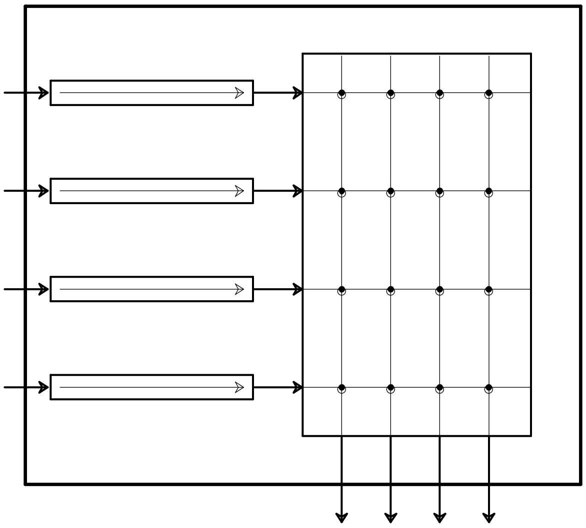

Figure 1.4: A switch with FIFO input buffers.

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

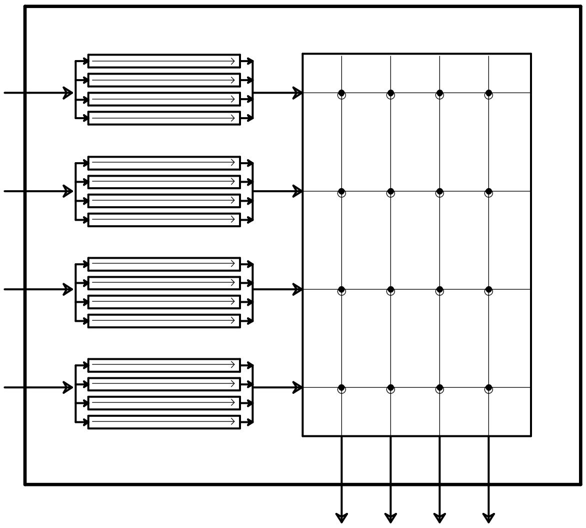

Figure 1.5: A switch with multi-queue input buffers.

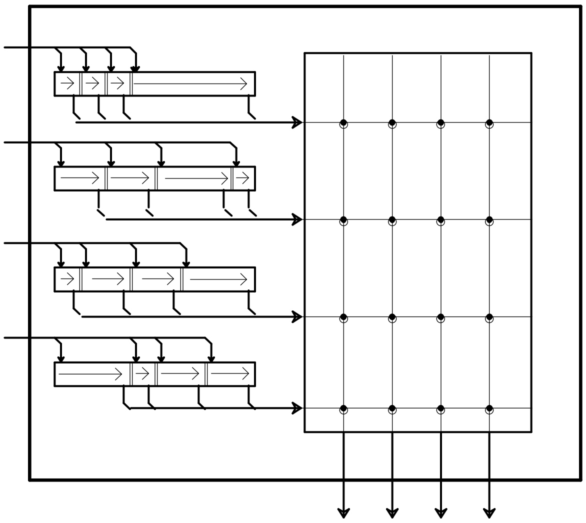

Figure 1.6: A switch with DAMQ input buffers.

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.