A VLSI Self-Compacting Buffer for DAMQ Communication Switches

José G. Delgado-Frias and Richard Diaz
Department of Electrical Engineering
State University of New York
Binghamton, NY 13902-6000

Abstract

This paper describes a novel VLSI CMOS implementation of a self-compacting buffer (SCB) for the dynamically allocated multi-queue (DAMQ) switch architecture. The SCB is a scheme that dynamically allocates data regions within the input buffer for each output channel. The proposed implementation provides a high-performance solution to buffered communication switches that are required in interconnection networks. This performance comes from not only the DAMQ approach but also the pipelined implementation and novel circuitry. The major components of the SCB are described in detail in this paper. The system has the capability of performing a read, a write, or a simultaneous read/write operation per cycle due to its pipelined architecture.

1 Introduction

An n by m buffered switch is a critical component in many interconnection networks. The performance of these networks is closely related to the architecture of the buffered switch. This paper describes a VLSI design and implementation of a switch architecture that uses a self-compacting buffer [1].

A router is composed of input controllers, a (n by n) switch, and output controllers. The input controller receives incoming packets and determines the appropriate output channel number according to a routing algorithm. The (n by n) switch delivers the packets from n input controllers to the n output controllers and the output controller sends the packet to the neighboring node. Figure 1 show an example of a block diagram for a router and an input controller. The input controller has three major functions. First, the input controller is responsible for receiving the packet and distributing the header part of the packet to the routing algorithm handler and to the packet flow controller. Second, it determines the output channel number based on the header information which is received from the input controller [3, 4]. This task is carried out by the routing algorithm handler. Third, it allocates and deallocates the buffer space for incoming and outgoing packets. This function is performed by the packet flow controller.

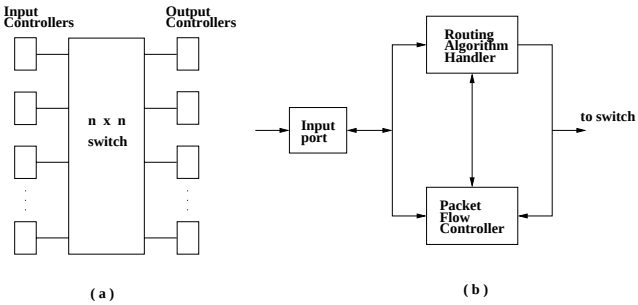

Figure 1: (a) Logical blocks of a router. (b) Logical blocks of an input controller

Tamir and Frazier [2] have classified the buffered switch architectures into four major types. This classification is based on how the input queues are manipulated and how data is stored. The four types are: first-in first-out (FIFO), statically allocated fully connected (SAFC), statically allocated multi-queue (SAMQ) and dynamically allocated multi-queue (DAMQ). FIFO, SAFC, and SAMQ buffered switches do not use efficiently the buffer space [2]. A better way of using the buffer is to dynamically allocate buffer space as is done with a dynamically allocated multi-queue (DAMQ). Space allocated for each buffer changes dynamically to fulfill the buffer space demands at a particular time. It has been reported that the DAMQ switch achieves the best performance among four switch types [1, 2]. The self-compacting buffer implements the DAMQ using a small amount of hardware and taking advantage of VLSI technology.

This paper has been organized as follows. Section 2 introduces the self-compacting buffer architecture and its properties. In Section 3, the cell designs and implementation of the buffer, buffer controller and channel pointers are described in detail. The timing of the self-compacting buffer is explained in Section 4. Some

concluding remarks are provided in section 5.

2 Self-Compacting Buffer

The self-compacting buffer (SCB) architecture has been organized to implement the dynamically allocated multi-queue scheme for buffer management. The SCB consists of a buffer, buffer controller, channel pointers and channel update. The overall organization of the SCB is shown in Figure 2.

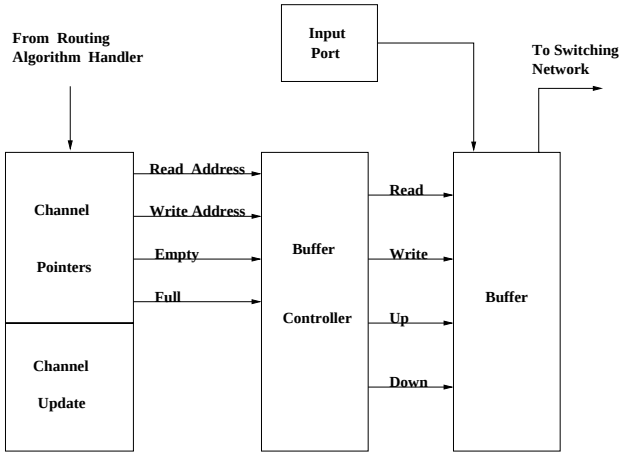

Figure 2: Self-compacting buffer organization

The function of the SCB is to store incoming packets from the input port and transfer outgoing packets to the switching network. An output channel number, received from the routing algorithm handler, points to an address in the buffer where its data is held. The channel pointer determines the buffer address for that channel number and updates the channel's address depending on the action (read or write). The buffer address is passed to the buffer controller along with the empty (selected channel contains no valid data) and full (all locations contain valid data) signals. With this information, the buffer controller sets the corresponding lines to move data from and/or to the buffer and shift data within the buffer.

The self-compacting buffer is divided dynamically into regions with every region containing the data associated with a single output channel. This scheme supports the dynamically allocated multi-queue (DAMQ) buffer management method introduced by Tamir and Frazier [2]. The self-compacting buffer scheme has the following properties:

Property 1. If two channels are denoted as i, k with $i < k$, then the dynamically allocated region for channel i and k always resides in a space with addresses A_i and A_k respectively where $A_i < A_k$.

Property 2. There is no reserved space dedicated for a channel i. If no data currently require the output channel i, then no region is reserved for channel i.

Property 3. Within the space for each channel, the data are stored in a FIFO manner.

Property 4. For every output channel i, there is an integer number δ_i, denoting the number of entries present in the region reserved for that output channel.

The set of properties of the buffer suggests that when an insertion/deletion in the buffer occurs via a write/read operation, there should be a mechanism to access arbitrarily the region associated with a channel. In particular, if the insertion of the packet requires space somewhere in the middle of the buffer, the required space must be created by moving all the data which reside below the insertion address. Furthermore, a reading from the top of the region for output channel data may create empty spaces in the middle of the buffer. The data below the read address is shifted up to fill the empty spaces. The buffer space maintained under the self-compacting buffer scheme is shown in Figure 3. Below, we discuss in detail the proposed high performance self-compacting buffer organization and VLSI implementation.

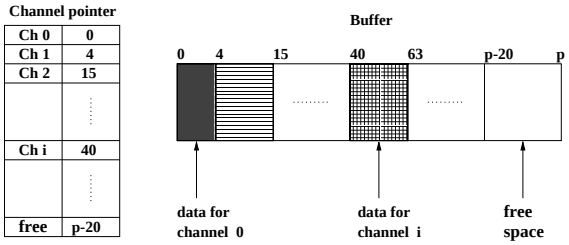

Figure 3: Buffer space

3 Buffer Implementation

In this section we describe the VLSI implementation of the buffer architecture described in Section 2. The requirements and circuitry of the three major components, buffer, buffer controller, and channel pointers, are described in detail here.

3.1 Buffer Organization and Cell Design

The buffer consists of a finite number of storage locations. The organization of the buffer is shown in Figure 4, where p storage locations are considered. For a location k, the following actions can occur:

- Shift up: row k moves its contents up to row $k-1$,
- Shift down: row k moves its contents down to $k+1$,
- Hold (no action): row k holds its data,
- Write: row k moves the write bus contents onto the cell, and
- Read: row k moves its contents to the read bus.

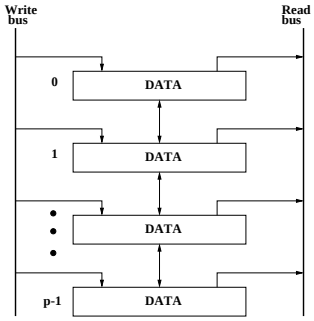

Figure 4: Buffer organization

These actions have to be performed by the buffer once the proper paths are set up. The buffer cell has to implement shift up, shift down, hold, write, and read actions as described above. A buffer cell in this buffer organization shares the up, down, read and write signals with cells in the same row, while the read and write bus lines are shared by cells in the same column. Figure 5 shows a CMOS circuit that implements the proposed buffer.

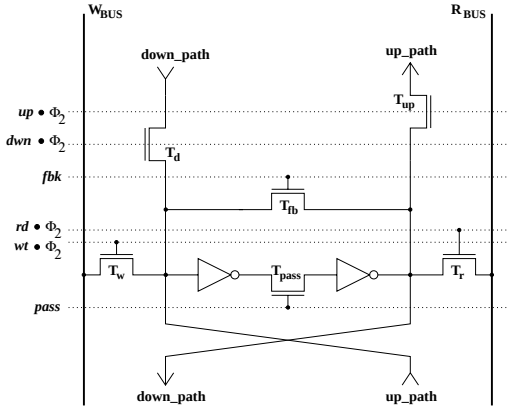

Figure 5: Buffer cell

This cell has the capability of performing all the required data moves. These data transfers are implemented as follows:

Hold data: If there is no action to be taken, cells should hold their information. Thus, feedback in the storage cells should occurs. During feedback, transistors T_{fb} and T_{pass} are on. This in turn allows the storage cells to hold their data.

Read/write data: If a read and/or write is requested transistors T_r and/or T_w are turned on, respectively. It should be pointed out that at this time transistors T_{fb} and T_{pass} are both off to isolate the incoming data from the outgoing data. The proposed buffer cell allows read and write from the same cell to take place at the same time; as the previous data leaves the cell the new data can be stored. This capability is also required to implement shift up and shift down as explained below.

Shift up/down data: When shifting down or up, the cell must be able to separate the incoming data from the outgoing data. Transistor T_d or T_{up} is on when shifting down or shifting up occurs, respectively. While a shifting operation takes place, transistors T_{fb} and T_{pass} are set off to isolate data in and data out. When shifting data down from a storage cell k to $k+1$, the path is set as follows. Data comes from the second inverter in cell k, passing through transistor $T_d(k+1)$ and into the first inverter of cell $k+1$. Similarly, when shifting data up from storage cell k to $k-1$, data comes from the second inverter in cell k, passing through transistor $T_{up}(k)$ and into the first inverter of cell $k-1$.

3.2 Buffer Controller

The self-compacting buffer operations include read and write and simultaneous read/write. There are four distinct cases by which the actions of storage cells in the buffer are determined. These four cases are explained below.

case 1) Single Write (Insertion). For a given address to write data in, all storage locations whose addresses are less than the write address retain their data. The storage locations whose addresses are greater than or equal to the write address must shift their data contents down to open a space in the buffer for incoming data.

case 2) Single Read (Deletion). All storage locations whose addresses are less than the reading address hold their data. The rest of the storage locations shift the contents of their storage location up.

case 3) Simultaneous Read/Write (address of read < address of write). In this case, the storage locations with addresses smaller than the read address are not affected. The storage location with addresses which are greater than the read and less than or equal to the write address should shift up their contents. The rest of the storage locations take no action.

case 4) Simultaneous Read/Write (address of write < address of read). In this case, only the storage location whose addresses are greater than or equal to the write address and less than the read address, shift their contents down. Other storage locations require no action.

The buffer controller determines how the buffer's data is moved within the buffer as well as from the read bus (R_{BUS}) and to the write bus (W_{BUS}). Based on the case of the current request(s), the controller will

set the path for the data to be moved.

Case selection is determined just after a write and/or read request(s) is (are) received. When a single write (or read) occurs, the buffer controller decodes the address and selects the corresponding write (or read) line for that row of cells. The rest of the rows with address larger than the selected one are set to shift down (or shift up) the stored data. These cases correspond to case 1 and 2. If a simultaneous read and write occurs, the decoding of both addresses is done at the same time. The distinction between cases (3) and (4) is in which direction the data is shifted. This shifting depends on where the read and write actions are located within the buffer. This needs to be determined to set the buffer's down and up lines.

When the buffer is full (i.e. all storage locations contain valid data), the buffer controller prevents data from being written to the buffer unless a simultaneous read/write occurs. In this case, writing data is allowed since the simultaneous read creates a space in the buffer. When the selected channel is empty, an empty signal informs the buffer controller that this channel (buffer address) contains no data. If a channel is empty, the buffer controller cancels read request for that channel. Thus, the full and empty input signals cancel write and read operations, respectively.

Figure 6 shows the CMOS circuit diagram for selection of a down line in row k. The internal write signal (W_{int}) is generated using the full signal (that prevents writing when the buffer is full) and the external write request signal. Therefore, signal W_{int} determines when writing to the buffer is allowed. If there is an allowable write request (i.e. $W_{int} = 1$), the $down - w_k$ line is precharged through transistor T_{pre}. At the same time, transistor T_b is on to set transistor T_{kill} off, preventing a short circuit between T_{kill} and T_{pre} since T_c is on. After precharge occurs, transistor T_{pass} allows the $write_k$ signal to pass to the buffer control cell. Writing to row k sets a 1 in signal $write_k$, this signal turns T_{kill} on and T_c off. All down lines above row k will be discharged and all down lines below will remain charged (i.e. set to a logic 1). When there is no allowable write request (i.e. $W_{int} = 0$), transistor T_a and T_{kill} are on and transistor T_c is off. This in turn causes a discharge of the $down - w_{k-1}$ line through T_{kill} and the $down - w_k$ line through cell k+1. Thus, all the *down* signals are set to logic 0. A similar circuit (as the one in Figure 6) is used for the read request to set the $up - r$ lines. In this case signals $read_k$ and R_{int} are used.

Once the down and up lines have been set, they are allowed to pass to the buffer depending on the

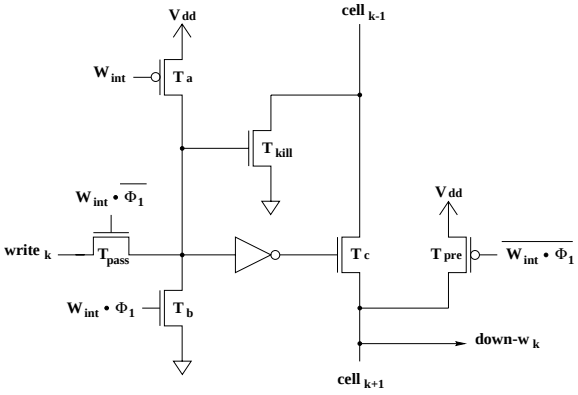

Figure 6: Controller's down signal generator cell

selected case. For a single write, all of the down lines below the selected row are set to 1 (all others are set to 0) and all up lines are set to 0. A similar setting is done for a single read. In both cases all down and up lines are passed unchanged. When a simultaneous read and write occurs, there are two possible settings of the down and up lines. Figure 7 illustrates these settings for $read < write$ and $write < read$. The down and up (*dwn* and *up*) signals that the buffer receives are modified when both down and up signals ($down - w$ and $up - r$), generated by the allowable write and read requests, are set to a logic 1.

Read < Write (case 3)

Address	Action	Signal generator down-w	Signal generator up-r	Buffer dwn	Buffer up
0		0	0	0	0
1		0	0	0	0
		0	0	0	0
		0	0	0	0
j-1		0	0	0	0
j	read	0	0	0	0
j+1		0	1	0	1
		0	1	0	1
		0	1	0	1
k-1		0	1	0	1
k	write	0	1	0	1
k+1		1	1	0	0
		1	1	0	0
		1	1	0	0
p		1	1	0	0

Write < Read (case 4)

Address	Action	Signal generator down-w	Signal generator up-r	Buffer dwn	Buffer up
0		0	0	0	0
1		0	0	0	0
		0	0	0	0
		0	0	0	0
j-1		0	0	0	0
j	write	1	0	1	0
j+1		1	0	1	0
		1	0	1	0
		1	0	1	0
k-1		1	0	1	0
k	read	1	1	0	0
k+1		1	1	0	0
		1	1	0	0
		1	1	0	0
p		1	1	0	0

Figure 7: Settings of down and up lines

The proper selection of the lines is implemented by the CMOS circuit shown in Figure 8. For a row k, if the down or up line is a 1 or both are 0, the circuit lets the input to pass on; transistors T_{pd} and T_{pu} are on and transistors T_{gd} and T_{gu} are off. However, when the down and up lines are both 1, T_{pd} and T_{pu} are off and T_{gd} T_{gu} are on. Thus, both lines are set to logic 0 preventing any data movement for that row. This occurs during a simultaneous read/write (as well as when the controller cells are precharged).

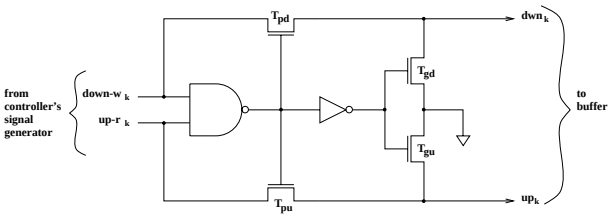

Figure 8: Case selector cell

3.3 Channel Pointers

If there are n output channels in the router system, the buffer should be able to accommodate data for these many channels. Data for each channel is dynamically allocated into the buffer; there is an address that corresponds to the beginning of the data for such channel. As a write or read operation occurs the amount of data in each channel increases or decreases. This makes the buffer to expand or compact, changing the location (address) of data for a set of output ports. This address is kept and updated by a set of channel pointers. Each channel pointer holds the address of the data for the corresponding output channel.

When a single read (or write) occurs, data is read from the top (or written to the bottom) of the selected channel decreasing (or increasing) its space. All channels greater than the selected channel update their addresses, decrementing (or incrementing) by 1. A read and write can occur simultaneously to the same channel; no channel is updated for a simultaneous read/write request in this case.

When a read and write requests are for different channels, the channels that are between these two channels will be updated. This is done in the same fashion that the buffer is updated. If the read channel number is less than the write channel, then all channels greater than the read channel and less than or equal to the write channel decrement their addresses by 1. Similarly, when the write channel number is less than the read channel number, all channels greater than the write channel and less than or equal to the read channel increment their addresses by 1.

When a channel contains no data (i.e. empty) its starting address is the same as the starting address of the next channel. The empty signal can be generated by comparing the addresses of adjacent channels; when these addresses are identical the empty signal for the particular channel is set. The channel update (increment or decrement) is canceled if the read or write operation for the given address is canceled due to an empty or full condition.

A CMOS circuit diagram for a channel pointer cell is shown in Figure 9. Reset of the circuit through transistor T_{reset} causes all channels to set their starting address to 0 (i.e. they are all empty). It should be noted that the first channel (i.e. channel 0) always contains the first address in the buffer. An output channel address is provided to the channel pointers with every read and/or write request(s). The channel address is decoded and the corresponding channel is selected, turning on transistors T_r and/or T_w respectively. This allows the buffer address to be passed to the read and/or write address bus. The sum (or difference) when incrementing (or decrementing) a channel's starting address is generated through an XOR of the stored address bit and the carry in C_{in}. This result passes through T_{clock} and T_{sum} which are both clocked transistors. However, T_{sum} only turns on if the sum_k signal is set to logic 1. The sum_k signal is generated when an add or subtract occurs. In addition, transistors T_{clock} and T_{sum} are never on at the same time since this would form a closed loop through the inverters and the output of the XOR. Adding or subtracting determines whether C_{in} is propagated to C_{out} or killed (set to logic 0). If the add_k signal is set to logic 1, transistor T_{add} is on allowing the $\overline{address\ bit}$ to pass. Therefore, C_{out} is equal to C_{in} when the address bit is set to logic 1, otherwise C_{out} is killed. Setting the sub_k signal to logic 1, allows the address bit to pass through transistor T_{sub}. In this case, if the address bit is set to logic 0, C_{out} is equal to C_{in}. Finally, while no operations are being performed transistor T_{fb} is turned on to provide feedback.

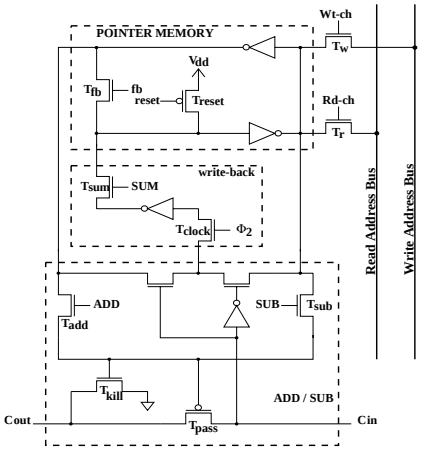

Figure 9: Channel pointer cell

The channel update has a similar function and organization as the buffer controller. It utilizes the circuits shown in Figures 6 and 8 to generate the add_k and sub_k lines that are passed to the channel pointers.

4 System Timing

The current self-compacting buffer implementation uses a two-phase clock scheme. In this implementation the operations of the channel pointers, buffer controller, and buffer are pipelined; thus, these operations can be overlapped. A detailed description of the timing approach follows.

A timing diagram for some operations that can occur is shown in Figure 10. The operations shown are simultaneous read/write, single read (when the selected channel is empty) and single write (when the buffer is full). The channel pointers receive the output channel number at the beginning of clock ϕ_2. At this time, the channel address is decoded and subsequently, the buffer address is passed on to the buffer controller at ϕ_1. Updating of the channel pointers begins at the trailing edge of ϕ_2 and generation of the read, up, write and down lines at the trailing edge of ϕ_1. In the next ϕ_2 cycle the data is written, read and shifted in the buffer. This timing is used for a single read or single write except that the operations followed are for the read or write, respectively.

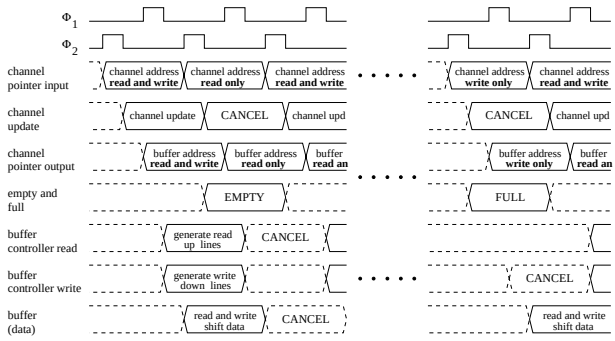

Figure 10: Self-compacting buffer timing

If the selected channel is empty when a read is requested or the buffer is full when a write occurs, the channel update, buffer controller and buffer operations are canceled during that cycle. This is shown in the timing diagram for a read only and write only. For a simultaneous read/write with an empty and/or full signal, the read and/or write operations would be canceled. The empty and full signals in a given cycle occur due to the channel update from the previous cycle.

The channel and buffer addresses remain valid for a complete clock cycle although they are not needed the entire cycle. However, the next address cannot be presented until the end of the cycle since updating the channel pointers requires a full cycle. Therefore, the packet flow controller can receive a channel address at every cycle. Data can also be read and written every cycle except if a channel is empty during a single read or the buffer is full during a single write.

5 Concluding Remarks

A novel VLSI CMOS implementation of a self-compacting buffer (SCB) for the dynamically allocated multi-queue (DAMQ) switch architecture has been presented in this paper. The DAMQ switch has been shown to provide the best performance among the buffered switch architectures [2]. The SCB is a novel scheme to dynamically allocate data regions for each output channel as required in the DAMQ switch. The SCB allocates only the required buffer space per channel allowing data expansion as needed to accommodate data storage demands.

We have presented the SCB architecture major components as well as the VLSI CMOS circuitry associated with these components. The components of the SCB are capable of performing a read, a write, or simultaneous read/write operations. The major blocks of the SCB architecture include the buffer that stores, moves in and out, and shifts data; the buffer controller that selects a case for each operation which determines data movements in the buffer; and the channel pointer which keeps track of the data dynamic changes in the buffer. For each of these components we have developed novel circuitry.

The proposed SCB VLSI implementation has been extensively simulated and fabricated. The SCB system has been pipelined to further enhance its high-performance. This system has the capability of performing a read and/or write operations per cycle as it was shown in the system's timing.

References

[1] J. Park, B.W. O'Krafka, S. Vassiliadis, J.G. Delgado-Frias, "Design and Evaluation of a DAMQ Multiprocessor Network With Self-Compacting Buffers," *IEEE Supercomputing '94*, pp. 713-722, Washington D.C., November 1994.

[2] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, Vol. 14, No. 6, pp. 725-737, 1992.

[3] D.H. Summerville, J.G. Delgado-Frias, and S. Vassiliadis, "A Flexible Bit-Pattern Associative Router for Interconnection Networks," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 477-485, 1996.

[4] A.A. Chien, "A Cost and Speed Model for k-ary n-cube Wormhole Routers," *Hot Interconnects '93*, Palo Alto, California, August 1993.