A VLSI Wrapped Wave Front Arbiter for Crossbar Switches

José G. Delgado-Frias
Dept. of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA 22904-4743
delgado@virginia.edu

Girish B. Ratanpal
Dept. of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA 22904-4743
ratanpal@virginia.edu

ABSTRACT

In this paper a novel VLSI CMOS implementation of a high performance wrapped wave front arbitration (WWFA) for Crossbar switches is described. Crossbars are one of the key components of communication switches used in networks. The transmission of packets is often delayed due to poor allocation of resources. Hence, switches must include an *arbiter* that efficiently allocates these resources. Arbitration time is one of the critical factors that affect the performance of networks. WWFA requires a two-dimensional arbitration that incorporates a rotating priority. In this paper we show the design and implementation of this arbiter. The arbiter is capable of performing an arbitration in `1.15ns` using `0.5`μm technology, for a 4x4 crossbar.

Keywords

Crossbar switch, arbiter, interconnection network, network router

1. INTRODUCTION

A computer network needs routers to receive, forward and deliver packets. To achieve high performance, it is necessary to have a router that provides high bandwidth and low latency. In general, a router can be considered a collection of network interfaces, some sort of bus or connection fabric connecting those interfaces and some logic that determines how to route packets among those interfaces [4]. A crossbar switch may serve as a switching fabric to provide a *non-blocking* network configuration [3]. Crossbars are one of the key components in most communication switches used in today's networks.

In order to better utilize the available input buffer space, the input buffers can use multi-queue buffers that have shown to significantly increase network throughput [2], [7] and [8]. Each multi-queue input buffer is able to transmit through the crossbar the packet at the head of any of its queues. If any one of these packets is transmitted the input buffer output bandwidth is fully utilized. Each queue can be considered to be a *user* and each internal crossbar bus (input and output) to be a *server* (resource). Hence, the switch consists of n^2 users and $2n$ resources. Figure 1 gives a user-resource perspective of a nxn crossbar switch matrix. The input port buses (IB_i) intersect the output port buses (OB_j) at crosspoints ($CP_{i,j}$). Any IB can connect to a OB by closing the appropriate crosspoint switch. A queue $q_{i,j}$ can transmit data to an output port j by acquiring IB_i and OB_j. These two resources are essential for a complete data path. This translates into a request to close the crosspoint switch $CP_{i,j}$. A conflict occurs when two or more queues request for the same input bus or output bus. For example, both $q_{1,1}$ and $q_{1,2}$ require the same input port bus (IB_1); we call this input-bus conflict. Another example, both $q_{3,2}$ and $q_{4,2}$ require the same output port bus (OB_2); we call this output-bus conflict. Only one user can be granted access to a resource. Hence, an *arbiter* needs to be employed to resolve these conflicts and assign resources to users. Arbitration schemes are needed to produce high throughput by providing a maximum user-resource utilization. Also, the arbiter grants requests to crosspoints so that at the most one grant is given per row per column of the crossbar matrix. Since there are n resource pairs, maximum throughput would require maximum number of connected pairs, i.e., $\sum_{i=1}^{n}\sum_{j=1}^{n} CP_{i,j(closed)} \Rightarrow n$. The arbiter can be decomposed into a group of arbiter cells with one arbiter cell associated with each crosspoint. The arbiter considers the requests for each crosspoint and depending upon the arbitration scheme determines the ones to be granted.

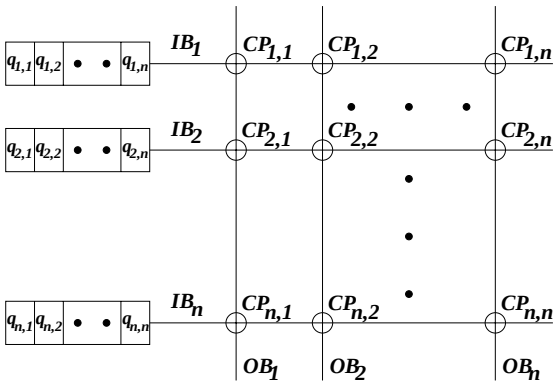

Figure 1: User-Resource model of a nxn crossbar.

The crossbar arbitration policy has a significant impact on the overall performance of the crossbar switch [6]. Y. Tamir and H.-C. Chi have studied three arbiter schemes [6]: *Skewed Two-Step Arbiter* (STSA), *Wave Front Arbiter* (WFA) and *Wrapped Wave Front Arbiter* (WWFA). These arbiters differ primarily in the manner in which priorities to the individual arbiter cells are assigned. Throughput and average latencies of each arbiter have been evaluated using static probabilistic analysis and event driven simulations [6]. These evaluations have shown that WFA and WWFA achieve nearly same performance as more complex schemes. Further, WWFA has been shown to be approximately *twice* as fast as the WFA. With the additional advantage of being amenable to efficient VLSI implementation, WWFA is the arbiter of choice.

2. WRAPPED WAVE FRONT ARBITER

Figure 2 shows the schematic of WWFA for a crossbar switch. In this example a *4x4* crossbar switch is presented. The circles represent the basic building block of the arbiter. This basic block is called *Crosspoint Arbiter Cell* (CAC) in our study. The numbers associated with each CAC indicate the corresponding crosspoint on the switch matrix. The dashed lines indicate the wrapped diagonals of the arbiter. All the CACs on a same diagonal have equal priority. In this example, diagonal (2) has the highest priority. The following diagonals 3, 4 and 1 have priorities decreasing in that order. As explained earlier, two distinct conflicts can occur: input-bus conflict (x-direction) and output-bus conflict (y-direction) when two or more requests occur per input- or output-bus, respectively. Only one connection per input

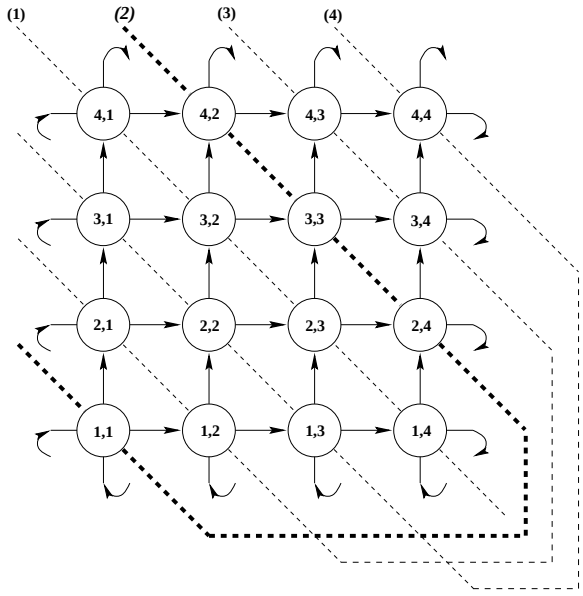

Figure 2: Wrapped Wave Front Arbiter (WWFA)

port and per output port can be established. This in turn requires the arbitration scheme to consider both x- and y-directions. The WWFA approach assigns priorities to each diagonal. With the priority diagonals of the WWFA arbiter being "wrapped around" to form wrapped diagonals, the CACs associated with any diagonal are guaranteed not to conflict since they all lie on separate rows and columns.

The arbitration *wave front* begins with n CACs that lie on the highest priority diagonal. It is important to point out that the priorities are rotated every arbitration cycle to provide fairness [1]. In the next arbitration cycle, priority is given to the diagonal following the one which held it last; in our example in Figure 2, the next diagonal would be 3.

2.1 Rules for WWFA

The fundamental rule of this arbitration scheme is that only one CAC per row, per column, can be granted. For example in Figure 2, if CAC *(1,1)* receives a grant, then no other CAC in row 1 (CACs *(1,j)*) and column 1 (CACs *(i,1)*) can be granted. On the other hand, if CAC *(1,1)* is not granted, then the priority is passed on to the CACs in its row and column, in the positive-x- and positive-y-direction respectively. Thus, when the wave front moves to diagonal (3), CAC *(2,1)* gets a grant if both CACs *(1,1)* and *(2,4)* do not. Also, a grant cannot be generated by a CAC unless the associated crosspoint switch is requested and the output port it services, is not busy. Summarizing these rules, it can be said that a CAC can be granted if and only if the following three conditions are met:

Request. The associated crosspoint switch is *requested*,

Priority. The CAC has *highest priority* or *none* of the CACs with higher priority in its corresponding row and column have been granted, and

Output port. The output port serviced by the associated crosspoint is *ready* to receive data.

Figure 3 shows graphically, the rules for WWFA. Concentric circles indicate a requested crosspoint, while shaded concentric circles indicate that the request has been granted. A '0'

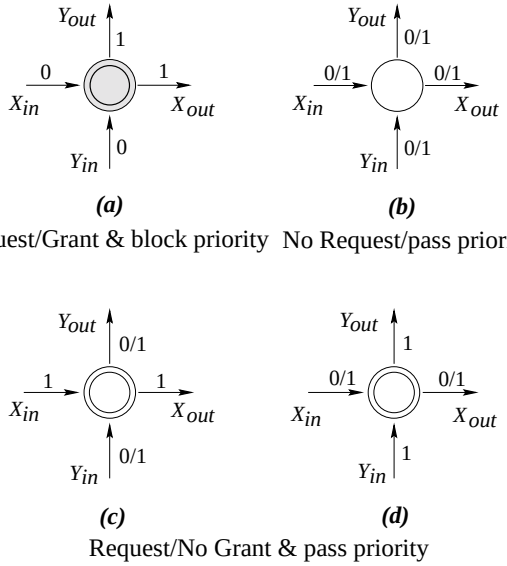

Figure 3: Conditions for Granting a Request.

at X_{in} or Y_{in} indicates that this CAC has priority in x- or y-direction, respectively. The grant output (G) is asserted

if, and only if, there is a request (R), output port ready signal (OR) is '1' and both X_{in} and Y_{in} are '0'. If the CAC grants the request, it pulls its X_{out} and Y_{out} signals to '1'. This case is shown in Figure 3(a). In this figure we have the arbitration cell's request input and its arbitration output(s) in the following format: *input/outputs*; where the input is either request or no request, while the outputs are grant or no grant and pass or block priority. No request for the crosspoint results in passing the priorities to the following CACs (Figure 3(b)). If the CAC does not receive a grant then it makes itself transparent and X_{out} and Y_{out} signals obtain the values of X_{in} and Y_{in} respectively. Two cases are shown in Figure 3(c) and (d).

Figure 4 illustrates the working of a WWFA. In this example it has been assumed that all the output ports are ready to receive data and diagonal (1) has the highest priority. It

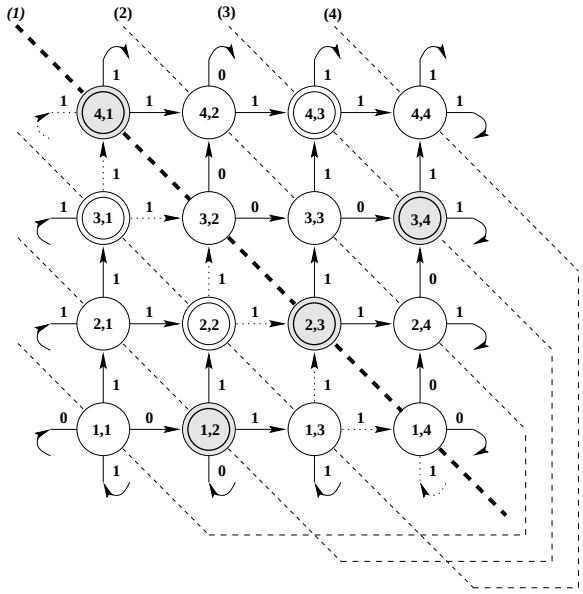

Figure 4: WWFA example.

should be pointed out that before the start of the arbitration cycle, the x and y inputs of all the CACs *except* the ones on the highest priority diagonal are initialized to '1'. In this example, the wavefront starts at diagonal (1) which has the highest priority. If a particular CAC receives a grant, it does not *disturb* the x and y inputs of the neighboring cells. As a result the priority is blocked. If a CAC does not receive a grant, then it passes on its x and y inputs.

Diagonal (1) has two requests at CACs *(4,1)* and *(2,3)* which are granted. This in turn denies priority to any request in rows 4 and 2 and column 1 and 3. Since diagonal (2) has no requests, the priority is passed to diagonal (3). This diagonal has three requests (CACs *(4,3)*, *(3,4)* and *(1,2)*). CAC *(4,3)* receives no priority in both directions, however, CACs *(3,4)* and *(1,2)* get priority and are granted. This in turn denies priority to any request in the following diagonal in rows 3 and 1 and columns 4 and 2. Any request in diagonal 4 will be denied. Figure 4 shows a configured arbiter at the end of the arbitration cycle.

3. CROSSPOINT ARBITER CELL DESIGN

Based on the requirements of the scheme we have designed a crosspoint arbiter cell (CAC). This cell should take into account priority in both x- and y-directions.

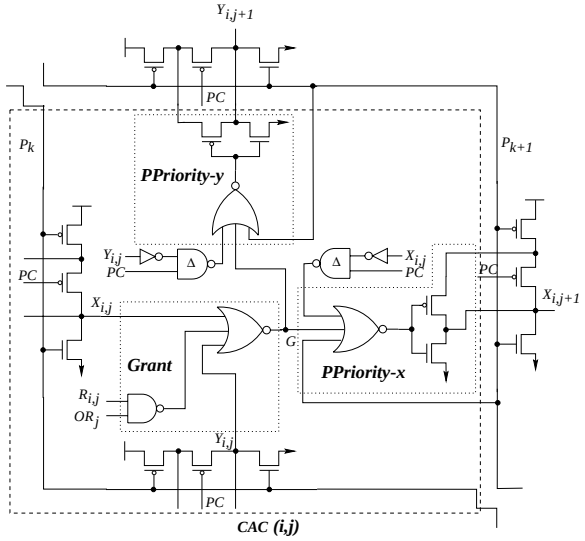

Figure 5: Complete circuit diagram of the CAC (i, j)

Figure 5 provides a circuit diagram of our CAC *(i,j)*. The *Grant* and the *PPriority* form the primary blocks of the CAC. There are two *PPriority* blocks, one per direction. The directional inputs ($X_{i,j}$ and $Y_{i,j}$) of all the cells, except the ones on the priority diagonal, are precharged to '1'. The directional inputs of the cells on the priority diagonal are discharged to '0'. Priority to a diagonal is indicated by P_k = '1'. The P_k (= '1' or '0') and PC (='0') signals arrive at approximately the same time which assign the appropriate value to $X_{i,j}$ and $Y_{i,j}$. This operation initializes the wavefront.

The *Grant* block determines the status of grant signal $G_{i,j}$. As stated earlier, a CAC can receive a grant if and only if its associated crosspoint is requested ($R_{i,j}$), it had priority in both x and y directions ($X_{i,j}$, $Y_{i,j}$) and the output port serviced is ready to receive data (OR_j). This is expressed as:

$$G_{i,j} \quad = \quad R_{i,j} \cdot OR_j \cdot \overline{X_{i,j}} \cdot \overline{Y_{i,j}} \qquad (1)$$

The primary function of the *PPriority* block is to pass priority to the following CAC. Both the *PPriority-x* and *PPriority-y* have similar operation; for simplicity, only the former is explained here. Since directional inputs of the cells are precharged to '1', a priority is passed by discharging them. A priority is passed only if $X_{i,j}$ is '0', the CAC does not receive a grant ($G_{i,j}$ = '0') and the following cell does not lie on the priority diagonal ($P_{k+1} \neq$ '1'). Once these conditions are met the output of the NOR gate goes to '1' discharging (passing the priority to) $X_{i,j+1}$.

Both the *Grant* and *PPriority-x* blocks require the value of signal $X_{i,j}$. If $X_{i,j}$ were fed to the *Grant* and *PPriority-x* blocks at the same time, the *PPriority-x* would see that the

cell is not granted and pass the priority to CAC *(i,j+1)*. Although the output of the *Grant* block may be affirmative, the priority has been already passed to CAC *(i,j+1)*. In the worst case, all the CACs in the associated row and column may generate a grant resulting in violation of the arbitration rules. Thus the input $X_{i,j}$ is delayed by an amount equal to that of the *Grant* block through NAND gate Δ.

In case the accidental discharge of $X_{i,j+1}$ does occur, its correct value is restored through the p-type transistor on the *PPriority-x* block. Here also, the signal $X_{i,j+1}$ is disturbed only if it does not lie on the highest priority diagonal.

4. PERFORMANCE

The wavefront starts from the highest priority diagonal and progresses towards the last diagonal. The arbitration cycle is said to be completed when the wavefront reaches the diagonal with the lowest priority, generating all possible grants. Figure 6 shows an example where diagonal (1) has the highest priority. It shows the advancement of the wavefront as seen in the CACs of row *4*. CAC *(4,1)* passes the priority to CAC *(4,2)* in time $t_{pass-pd}$. The priority moves further through CAC *(4,2)* and CAC *(4,3)*, each with a delay of $t_{pass-npd}$. The same occurs simultaneously in the y-direction from CAC *(1,4)* to CAC *(3,4)*. Once the priorities are received by CAC *(4,4)*, the grant signal $G_{4,4}$ is asserted after a delay of t_{NOR}. We could generalize the grant time ($t_{grant(n)}$) for an *n*x*n* arbiter as:

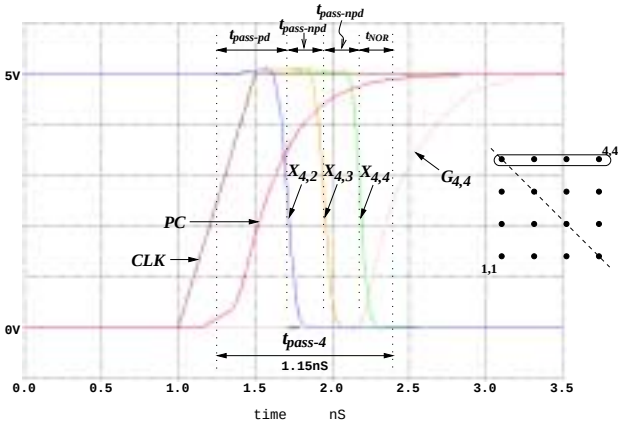

Figure 6: SPICE Simulation: Arbitration time of an *4*x*4* arbiter.

$$t_{grant(n)} = t_{pass-pd} + d \cdot (t_{pass-npd}) + t_{NOR}$$

Where d is the number of diagonals (or CACs) between the lowest and the highest priority diagonal. It should be pointed out that the magnitudes of $t_{pass-pd}$, $t_{pass-npd}$ and t_{NOR} would depend on technology and implementation. From our SPICE simulations, using a `0.5`μ`m` technology, the values of these delays are: $t_{pass-pd} = 0.47nS$, $t_{pass-npd} = 0.23nS$, and $t_{NOR} = 0.22nS$. From this data the total arbitration time $t_{grant(4)}$ of a *4*x*4* arbiter can be calculated as follows:

$$t_{grant(4)} = t_{pass-pd} + 2 \cdot (t_{pass-npd}) + t_{NOR}$$
$$t_{grant(4)} = 1.15nS$$

5. CONCLUDING REMARKS

In this paper we have presented a novel VLSI design of a wrapped wave front arbitration scheme for communication switches. Our arbiter fulfills the two-dimensional arbitration requirements needed in WWFA. It provides a flexible priority setting; often a rotating priority is used to provide a fair arbitration. Our design achieves high performance due to minimal circuitry in the critical path [5] [9]. Using a `0.5`μ`m` technology and a *4*x*4* crossbar switch, the arbitration scheme produces a valid configuration in `1.15ns`.

6. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under contract CCR 9900643.

7. REFERENCES

[1] L. N. Bhuyan, "Analysis of Interconnection Networks with Different Arbiter Designs," *J. Parallel Distributed Computing*, vol. 4, no. 4, pp. 384-403, Aug. 1987.

[2] J. G. Delgado-Frias and R. Diaz, "A VLSI Self-Compacting Buffer for DAMQ Communication Switches," *IEEE Eighth Great Lakes Symposium on VLSI*, pp. 128-133, Feb 1998.

[3] J. Duato, S. Yalamanchili and L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society, Los Alamitos CA. 1997

[4] C. Partridge et al, "A 50-Gb/s IP Router," *IEEE/ACM Transactions on Networking*, vol. 6, no. 3, June 1998.

[5] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall Inc., NJ, 1996.

[6] Y. Tamir and H.-C. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches," *IEEE Transactions on Parallel And Distributed Systems,* vol. 4, no. 1, pp. 13-27, 1993.

[7] Y. Tamir and G.L. Frazier, "The Design and Implementation of a Multi-Queue Buffer for VLSI Communication Switches," *Proc. Int. Conf. Computer Design*, Cambridge, MA, Oct. 1989, pp. 466-471.

[8] 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.

[9] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Second Edition, Addison-Wesley, Reading, Mass., 1993.