Programmable Weighted Arbiters for Constructing Switch Schedulers

Mei Yang[†], S. Q. Zheng[‡], Bhagyavati[†], and Stan Kurkovsky[†]
[†]Department of Computer Science
Columbus State University, Columbus, GA 31907, USA
{yang_mei, bhagyavati, kurkovsky_stan}@colstate.edu
[‡]Department of Computer Science
University of Texas at Dallas, Richardson, TX 75080, USA
sizheng@utdallas.edu

***Abstract*— As the basic building block of a scheduler based on maximal weight matching algorithms, the design of a weighted arbiter is vital to the performance of the scheduler. All existing weighted arbiter designs are based on the binary comparator tree structure and consume $O(bN)$ gates, where b is the number of bits needed to represent the weight. These designs are not desirable for implementing scheduling algorithms that require a large number of weighted arbiters. In light of the idea of radix sort, we propose a new weighted arbiter (WA) design and a programmable weighted arbiter (PWA) design, both with $O(N)$ gates. Through simulations, we show that the proposed WA design achieves significant improvement on area cost than existing WA designs. The proposed PWA design provides round-robin fairness for requests with the same weight. Both designs can be directly used to build schedulers based on maximal weight matching algorithms. They are also useful for other applications, such as the arbitration of a shared bus and control of real-time systems.**

I. Introduction

The cell scheduling problem for virtual output queue (VOQ) based switches can be modelled as a bipartite matching problem [1]. Although maximum weight matching algorithms are proved to achieve 100% throughput for all admissible identically independently distributed (i.i.d.) arrivals [2], they are infeasible for high speed implementation with their time complexity of $O(N^3 \log N)$ [3]. The most efficient maximum size matching algorithm has a time complexity of $O(N^{2.5})$ [3], [4]. However, maximum size matching algorithms are too complex for hardware implementation and can cause unfairness [2]. Most practical scheduling algorithms proposed in the literature are either maximal size matching algorithms, such as parallel iterative matching (PIM) [5], iSLIP [1], dual round-robin matching (DRRM) [6], or maximal weight matching algorithms, such as iterative longest queue first (iLQF) and iterative oldest cell first (iOCF) [1], the longest normalized queue first (LNQF) [7], and the dynamic DiffServ scheduling (DDS) [8]. Compared with maximal size matching algorithms, maximal weight matching algorithms achieve better performance under both uniform and nonuniform traffic [1].

All these maximal size or weight matching algorithms can be implemented by the scheduler architecture shown in Figure 1, in which each input/output port is associated with an arbiter. The function of an arbiter is arbitrating among multiple requests. As the basic building block of a scheduler, the design of an arbiter is important to the performance of the

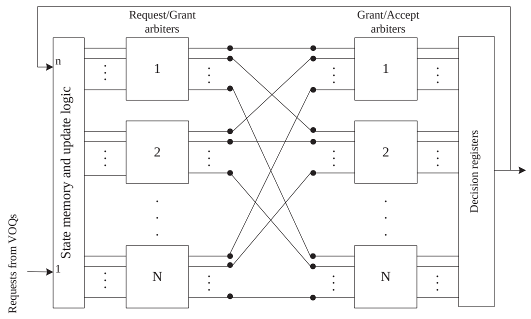


Fig. 1. Block diagram of a scheduler based on a maximal size/weight matching algorithm.

scheduler. In [9], Gupta and McKeown reviewed several well-known round-robin arbiter (RRA) designs and proposed two new RRA designs, named as programmable priority encoders (PPEs). In [10], we proposed a parallel RRA (PRRA) design based on a binary tree structure and showed that PRRA is faster and simpler than PPEs in practice [10]. However, all these three designs cannot arbitrate weighted requests. Hence, they are not suitable for implementing maximal weight matching scheduling algorithms.

We focus our study on the designs of weighted arbiters and programmable weighted arbiters. A weighted arbiter selects the first request with the maximum weight. A programmable weighted arbiter is a generalized weighted arbiter with its selection starting point programmable. It is very useful for providing fairness for requests with the same weight (referred as ties [2]). All existing designs of weighted arbiters are based on a (binary) comparator tree structure [1], [11], [12], [13]. Each node in the binary tree is a comparator which compares the weights of the requests coming from its children and sends the request with the maximum weight to its parent node. These designs have $O(\log b \log N)$-gate delay and consume $O(bN)$ gates, where b is the number of bits needed to represent the weight. For large b, these designs are not desirable for applications which require a large number of weighted arbiters, such as implementing the DDS algorithm [8].

In this paper, we propose a weighted arbiter (WA) design and a programmable weighted arbiter (PWA) design, both with $O(N)$ number of gates. The basic idea of our designs is similar to the idea of radix sort [14] but different from radix sort in finding the maximum weight by selecting the maximum value (‘1’ for binary digits) on the most significant bit, on the second significant bit, and so on. The arbitration of the lowest significant bit is performed by using either the priority encoder with a binary tree structure for the WA design, or using the PRRA [10] for the PWA design. The PWA design provides round-robin fairness for those requests with the same weight. Both designs have $O(b \log N)$-gate delay. Noticeably, the number of gates consumed by both designs is only $O(N)$. The product of the gate delay and the number of gates of our designs is $O(Nb \log N)$, which is better than that of the comparator tree design, $O(Nb \log N \log b)$.

Simulation results on Synopsys’ *design_analyzer* [15] conform to our analysis. The proposed programmable weighted arbiter designs can be directly used to build schedulers based on maximal weight matching algorithms. They are also useful for other applications, such as the arbitration of a shared bus and control of real-time systems.

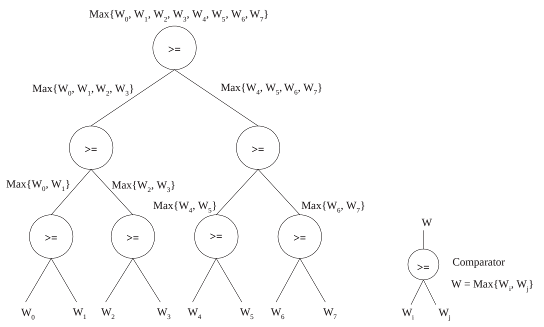


Fig. 2. Structure of a binary comparator tree with 8 weighted requests.

II. Related Work

Although the problem of finding the request with the maximum weight is of practical importance, only a few weighted arbiters have been proposed in the past. The variable round-robin arbiter proposed in [12] is based on a binary comparator tree structure. Each node in the binary tree is a comparator which compares the weights of the requests coming from its children and sends the request with the maximum weight to its parent node. To ensure round-robin fairness among requests with the same weight, each node is associated with a token bit to indicate its priority. Extra logic is needed to propagate the token information. In [1], the iLQF scheduler and the iOCF scheduler are all built by weighted arbiters based on the binary comparator tree structure. The arbiter designs proposed in [11] and [13] are also based on the binary comparator tree structure.

Figure 2 shows a binary comparator tree with 8 weighted requests. In general, a binary comparator tree with N weighted requests consists of $O(\log N)$ levels of nodes, each as a 2-input comparator. Assume that b is the number of bits used to represent the weight, the comparator tree design has $O(\log b \log N)$-gate delay and consumes $O(bN)$ gates since each node consumes $O(b)$ gates. Such a design is not scalable for large b. In next section, we introduce two scalable weighted arbiter designs which only consume $O(N)$ gates.

III. Programmable Weighted Arbiter Designs

Our goal is to design a simple yet fast weighted arbiter with a programmable selection starting point. The function of our *Programmable Weighted Arbiter* (PWA) is defined as follows. Given N binary requests R_i’s and their b-bit weights W_i’s, where $0 \leq i \leq N-1$, and one integer x, $0 \leq x \leq N-1$, select $R_{(x+m) \bmod N}$ such that $m = \min\{l \mid W_{(x+l) \bmod N} = M, R_{(x+l) \bmod N} = 1, 0 \leq l \leq N-1\}$, where $M = \max\{W_i \mid R_i = 1, 0 \leq i \leq N-1\}$. If R_i is selected, then the output grant signal $G_i = 1$; if $R_i = 0$, or $R_i = 1$ but it is not selected (in such a case, $W_i \leq M$), then $G_i = 0$. Note that ties are broken by selecting the first such request starting from position x in a circular manner. In the following, we first describe the basic idea of our designs and then discuss each design in detail.

A. Basic Idea

Inspired by the idea of radix sort, we find the maximum weight by selecting the request with the maximum value (‘1’ for binary digits) on the most significant bit, then the second significant bit, and so on, all starting from R_x. If there is at least one request with the maximum value (‘1’ for binary digits) on the current bit of its weight, we eliminate those requests with less value (‘0’ for binary digits) on the current bit of their weights. This process continues till either there is only one valid request left or the arbitration of the lowest significant bit is completed. Figure 3 illustrates an example with 4 requests. We assume that $R_i = 1$ for all $0 \leq i \leq 3$ and $x = 3$. We first check all digits on bit 3. Since there exist requests with ‘1’ on bit 3 of their weights, the second request is eliminated for it has ‘0’ on bit 3 of its weight. We then check all the remaining digits on bit 2, as enclosed in the eclipse in the figure. There exist requests with ‘1’ on bit 2 of their weights, therefore the first request is eliminated since it has ‘0’ on bit 2 of its weight. We then check all the remaining digits on bit 1. There is no request with ‘1’ on bit 1 of its weight, hence all remaining requests are valid for the process of bit 0, and R_3 wins finally.

Given R_i’s and their weights W_i’s, the general algorithm of finding the first $R_i = 1$ with the maximum weight starting from request R_x, FIND_MAX, is shown below.

```
Algorithm FIND_MAX( R, W, G )
    $c \leftarrow 0$, $k \leftarrow NIL$
    for $i \leftarrow 0$ to $N-1$
        do $G_i \leftarrow 0$, $T_i \leftarrow W_{i,b-1}$, $S_i \leftarrow R_i$
            if $S_i = 1$
                then $c \leftarrow c + 1$
    $j \leftarrow b - 1$
    while $j \geq 0$ and $c \geq 1$
        do find $K = \max\{T_i \mid R_i = 1, 0 \leq i \leq N-1\}$ and
            $k = \min\{l \mid T_{(x+l) \bmod N} = K, R_{(x+l) \bmod N} = 1,$
            $0 \leq i \leq N-1\}$
            if $j > 0$
                then $c \leftarrow 0$
                    for $i \leftarrow 0$ to $N-1$
                        do if $W_{i,j} \geq K$ and $S_i = 1$
                            then $T_i \leftarrow W_{i,j-1}$, $c \leftarrow c + 1$
                            else $S_i \leftarrow 0$
        $j \leftarrow j - 1$
    if $k \neq NIL$
        then $G_{(x+k) \bmod N} \leftarrow 1$
```

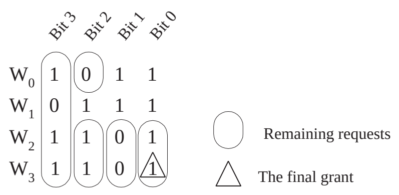


Fig. 3. An example with 4 requests.

In FIND_MAX, $W_{i,j}$ represents the j-th bit of W_i, where $0 \leq j \leq b-1$, T_i represents the valid digit on the current bit of W_i, S_i indicates the validity of request R_i and initially $S_i = R_i$, K represents the maximum value on the j-th bit, and k represents the index of the first non-zero request with $W_{k,j} = K$ starting from position x. FIND_MAX is composed of at most b rounds, each corresponding to the arbitration on bit j, $j = b-1, \cdots, 0$. Since W_i's are in binary digits, finding the largest value K on each bit is equivalent to OR all digits on the current bit of those weights of valid requests. In the round for bit j, if the result of OR is '1', which means that there exists at least one valid request with '1' on bit j of its weight, those requests with '0' on bit j of their weights will be set as invalid (eliminated). The logic functions of generating valid digits on bit j are derived as follows, where $A = 1$ if $j > 0$ and $A = 0$ otherwise.

$$K = \sum_{i=0}^{N-1} T_i \tag{1}$$

$$S_i = K \cdot T_i \cdot S_i + \overline{K} \cdot S_i \tag{2}$$

$$T_i = A \cdot S_i \cdot W_{i,j} + \overline{A} \cdot (K \cdot T_i + \overline{K} \cdot S_i) \tag{3}$$

This process continues till either there is only one valid request left or T_i's are generated in the last round (i.e., $j = 0$). For the first case, we simply set the grant signal to the valid request as '1' and the grant signals to other requests as '0'. For the second case, we need to find $T_{(x+k) \bmod N}$ such that $k = \min\{l \mid T_{(x+l) \bmod N} = 1, 0 \leq l \leq N-1\}$ (i.e. find the index of the first request with $T_i = 1$ starting from R_x), and set $G_{(x+k) \bmod N} = 1$ and $G_i = 0$ for $i \neq (x+k) \bmod N$. Notice that Equation (3) ensures that there exists at least one $T_i = 1$ in the last round. For rounds of bits $j > 0$, we simply generate K, S, and T according to Equations (1) to (3), which can be implemented by $O(N)$ number of gates and in $O(\log N)$-gate delay each round. The operation of the last round can be performed by either a priority encoder for $x = 0$ or the parallel round-robin arbiter for $x \geq 0$. We name these two designs as the weighted arbiter and the programmable weighted arbiter respectively.

B. Weighted Arbiter

We first consider the design of a weighted arbiter, i.e., a programmable weighted arbiter with $x = 0$. We propose a priority encoder (PE) design which finds the index of the first non-zero input taking T_i's as the inputs. Figure 4(a) shows a 4-input priority encoder based on a binary tree structure. Each node of the tree, named as priority encoder node (PEN), is a priority encoder with two inputs, as shown in Figure 4(b). Each PEN has two inputs r_0 and r_1, two outputs g_0 and g_1 with $g_i = 1$ indicating r_i is selected, where $i = 0, 1$, and another output r_o to indicate its upper layer PEN that there exists at least one non-zero input in the subtree rooted at it. The input and output relations of a PEN are specified by the following equations.

$$\begin{aligned} r_o &= r_0 + r_1 \\ g_0 &= r_0 \\ g_1 &= \overline{r_0} \cdot r_1 \end{aligned}$$

The final grants of the 4-input priority encoder are generated by using the root node's grants to mask out the grants of the subtree that is not granted by the root node, i.e.,

$$g_i = \begin{cases} g_i \cdot tg_0 & \text{for } i = 0, 1, \\ g_i \cdot tg_1 & \text{for } i = 2, 3, \end{cases}$$

where g_i is the grant signal for input r_i, $0 \leq i \leq 3$, and tg_0 and tg_1 are 1-bit grant signals generated from the root node. This design makes it possible to construct PEs recursively. An 8-input PE is built by one PEN, two 4-input PEs, and four AND gates. In general, as shown in Figure 4 (c), a 2^N-input PE is built by one PEN, two 2^{N-1}-input PEs, and N AND gates. It is easy to derive that the PE design has $O(\log N)$-gate delay, and consumes $O(N)$ gates since $N-1$ PENs are used in the binary tree. Hence the weighted arbiter has $O(b \log N)$-gate delay and $O(N)$ gates.

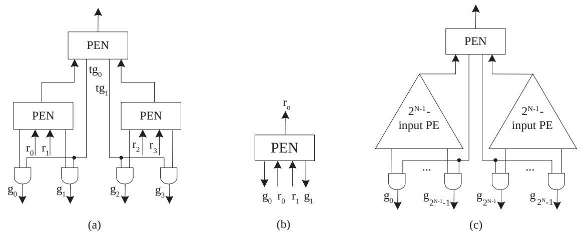
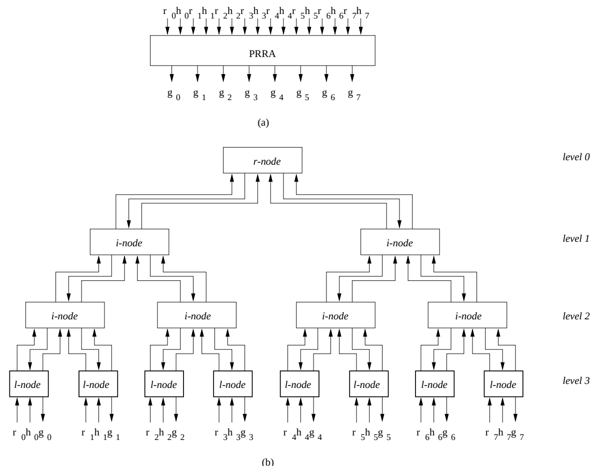


Fig. 4. The priority encoder design. (a) A priority encoder with 4 inputs. (b) A priority encoder node. (c) The recursive construction of a 2^N-input priority encoder.

C. Programmable Weighted Arbiter

To implement the programmable weighted arbiter, we use the parallel round-robin arbiter [10] to find the index of the first non-zero input starting from position x taking T_i's as the inputs, where $0 \leq x \leq N-1$. As shown in Figure 5, a PRRA with N inputs is based on a binary tree structure consisting of one level of leaf nodes (l-nodes), $\log N - 1$ levels of internal nodes (i-nodes), and one root node (r-node). Each input r_i, $0 \leq i \leq N-1$, is associated with a one-bit head information, h_i, which is used to indicate if r_i is the selection starting point. In [10], we proved that PRRA achieves round-robin fairness. It can be verified that PRRA design has $O(\log N)$-gate delay and consumes $O(N)$ gates. Hence, PWA has $O(b \log N)$-gate delay and consumes $O(N)$ gates.



$r_0 h_0 r_1 h_1 r_2 h_2 r_3 h_3 r_4 h_4 r_5 h_5 r_6 h_6 r_7 h_7$
PRRA
g_0 g_1 g_2 g_3 g_4 g_5 g_6 g_7
(a)
r-node
level 0
i-node
i-node
level 1
i-node
i-node
i-node
i-node
level 2
l-node
l-node
l-node
l-node
l-node
l-node
l-node
l-node
level 3
$r_0 h_0 g_0$ $r_1 h_1 g_1$ $r_2 h_2 g_2$ $r_3 h_3 g_3$ $r_4 h_4 g_4$ $r_5 h_5 g_5$ $r_6 h_6 g_6$ $r_7 h_7 g_7$
(b)

Fig. 5. Structure of a parallel round-robin arbiter with 8 inputs. (a) Inputs and outputs. (b) The tree structure.

Design	**N=8**	**N=16**	**N=32**	**N=64**
WA	80	170	346	698
PWA	147	311	639	1295
CTA	133	285	589	1197
Improvement	40%	40%	41.3%	41.7%

TABLE I

AREA RESULTS OF DESIGNS OF WA, PWA, AND CTA IN TERM OF THE NUMBER OF 2-INPUT NAND GATES.

Design	**N=8**	**N=16**	**N=32**	**N=64**
WA	18.61	19.67	24.36	29.05
PWA	19.16	21.91	27.26	32.61
CTA	6.62	7.72	9.01	10.30

TABLE II

TIMING RESULTS OF DESIGNS OF WA, PWA, AND CTA IN TERMS OF *ns*.

IV. SIMULATION RESULTS AND COMPARISONS

To evaluate the performance of our designs and compare them with existing weighted arbiter designs, we have conducted simulations for designs of the weighted arbiter (WA), the programmable weighted arbiter (PWA), and the comparator tree arbiter (CTA) on Synopsys' design tools. We have written the Verilog HDL [16] code for each design and synthesized them on Synopsys' *design_analyzer* [15]. All these designs are optimized under the same operating conditions and the tool is directed to optimize the area cost of each design.

Table I lists the area results (in terms of the number of 2-input NAND gates) of WA, PWA, and CTA with different number of inputs assuming $b = 4$. The bottom line shows the area improvement of WA to CTA. The area results of both WA and PWA are proportional to N, conforming to our analysis. The area improvement of WA over CTA is 40% or more for all N's.

Table II lists the timing results (in terms of ns) of WA, PWA and CTA with different number of inputs assuming $b = 4$. The timing results of WA and PWA are proportional to $b \log N$ and the timing results of CTA are proportional to $\log b \log N$. The function complexity of PWA makes its timing results and area results worse than that of WA and CTA.

Though the timing results of WA are not as good as CTA, the area improvement of WA over CTA is very promising. For example, to construct a DDS scheduler for a 64×64 switch, without counting all the state memory and update logic, it needs 128 arbitration components, each associated with an input or output port. One arbitration component consists of 6 arbiters, each dedicated for the arbitration of a traffic class. Totally it consumes $919,296$ gates using CTA, but it only consumes $537,072$ gates using WA. The advantage of PWA is its support of round-robin fairness for requests with the same weight, which is very useful for implementing maximal weight matching scheduling algorithms.

V. SUMMARY

In this paper, we proposed a weighted arbiter design and a programmable weighted arbiter design for constructing switch schedulers. The basic idea of our designs is finding the maximum weight by selecting the request with the maximum value ('1' for binary digits) on the most significant bit, on the second significant bit, and so on, all starting from a given position x. The WA and PWA designs are based on the proposed priority encoder design and the PRRA design [10] respectively. We showed that WA achieves significant area improvement over the comparator tree arbiter and PWA provides round-robin fairness for requests with the same weight. Both designs can be directly used to construct schedulers based on maximal weight matching scheduling algorithms, such as DDS [8], iLQF, iOCF [1], and LNQF [7].

REFERENCES

[1] N. McKeown, "Scheduling algorithms for input-buffered cell switches", Ph. D. Thesis, Univerity of California at Berkeley, 1995.

[2] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand., "Achieveing 100% throughput in an input-queued switch", *IEEE Trans. Commun.*, vol. 47, no. 8, Aug. 1999.

[3] R. E. Tarjan, *Data Structures and Network Algorithms*, Bell labs, Murray Hill NJ, 1983.

[4] J. E. Hopcroft and R. M. Karp, "An $n^{2.5}$ algorithm for maximum matching in bipartite graphs", *Soc. Ind. Appl. Math. J.*, vol. 2, pp. 225-231, 1973.

[5] T. Anderson, S. Owicki, J. Saxie, and C. Thacker, "High speed switch scheduling for local area networks", *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319-352, Nov. 1993.

[6] H. J. Chao, "Saturn: a terabit packet switch using dual round-robin", *IEEE Commun. Mag.*, pp. 78-84, Dec. 2000.

[7] S. Li and N. Ansari, "Provisioning QoS features for input-queued ATM switches", *Electronics Letters*, vol. 34, no. 19, pp. 1826-1827, Sept. 1998.

[8] M. Yang, E. Lu, and S. Q. Zheng, "Scheduling with dynamic bandwidth allocation for DiffServ classes", to be presented on *ICCCN 2003*, Dallas, TX, Oct. 2003.

[9] P. Gupta and N. Mckeown, "Designing and implementing a fast crossbar scheduler", *IEEE Micro.*, vol. 19, no. 1, pp. 20-28, Jan.-Feb. 1999.

[10] S. Q. Zheng, M. Yang, J. Blanton, P. Golla, and D. Verchere, "A simple and fast parallel round-robin arbiter for high-speed switch control and scheduling", in *Proc. 45th IEEE MWSCAS*, 2002, pp. 671-674.

[11] A. Bystrov, D. J. Kinniment, and A. Yakovlev, "Priority arbiters", in *Proc. ASYNC 2000*, pp. 128-137.

[12] K. C. Lee, "A variable round-robin arbiter for high speed buses and statistical multiplexers", in *Proc. ICCCN 1991*, pp. 23-29.

[13] F. Petrot and D. Hommais, "A generic programmable arbiter with default master grant", in *Proc. ISCAS 2000*, vol. 5, pp. 749 -752.

[14] T. H. Cormmen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 2nd Edition*, McGraw-Hill, 2001.

[15] Synopsys, Design Analyzer Datasheet [Online], Available: http://www.synopsys.com/products/logic/deanalyzer_ds.html, 1997.

[16] IEEE Standards Board, *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*, IEEE, New York, 1995.