An Efficient Round-Robin Arbiter

Himashinii M
Dept of ECE
Saveetha Engineering Chennai

Abstract- *Round robin arbiter (RRA) is a vital block in these days designs. it is widely located in machine-on-chips and community-on-chips. The need of an efficient RRA has elevated considerably as it's miles a limiting performance block.In this paper, we supply comparative assessment between distinctive RRA architec- tures determined in literature. We also suggest a singular green RRA architecture. The FPGA implementation consequences of the previous RRA architectures and our proposed one are given, that display the improvements of the proposed RRA.*

I. INTRODUCTION

Scheduling algorithms are required when multiple requestors require access to a shared resource. In a System on Chip (SoC), multiple devices in the chip are needed to work together. As a result, an SoC may have multiple bus masters. A fast and powerful arbiter becomes important to service all the bus masters. Another example of arbitration system application is network switches. In a network switch, packets from multiple input ports need to go through a single output port. As the number of parallel processes increases, accessing a shared resource becomes the bottleneck in performance. One of the goals of a scheduler is to maximize throughput. The advantages of utilizing arbiters include access fairness for the requestors to the resources, utilization without wasting cycles, re- usability, arbitration speed, power and resource overhead.Different types of Round- robin arbiters such as baseline arbiters, time speculative arbiters, acyclic arbiters, parallel prefix arbiters, priority based arbiters, etc... are used in various applications.

In this paper, a Round-robin arbiter is designed using weight decoder, next grant pre- calculate logic and granting logic. The weight of each granted requestor is decoded using a weight decoder logic. Based on the current grant, the next possible grant is precalculated in Round-robin mechanism. Finally, the granting logic checks for requests and precalculated next grant mask to select a single grant.

A Round-robin arbitration mechanism fits the Weight decoder, next grant precalculator.The number of requestors and the bit width of the weight can be set before synthesis.

II. ORGANIZATION

This paper is organized as follows:

- Discusses different arbitration algorithms.
- Discusses the design and implementation of weighted Round-robin arbiter.
- Discusses the test results.
- The conclusion and possible future work.

III. BACKGROUND RESEARCH

Arbiters play an important role when multiple requests are sent to access a single resource. In a network switching router, the packets received on input ports are sent out to the respective output ports. The arbiter acts a middle man to direct which input gets to send its packet to the designated output. The arbitration speed of the arbiter has a large factor in determining the speed of switching performance. Of the many metrics to benchmark an arbiter, fairness is a good unit to measure the performance, and there are a few types typically utilized:

$$\begin{bmatrix} X & W_{1,2} & W_{1,3} & W_{1,4} \\ W_{2,1} & X & W_{2,3} & W_{2,4} \\ W_{3,1} & W_{3,2} & X & W_{3,4} \\ W_{4,1} & W_{4,2} & W_{4,3} & X \end{bmatrix}$$

- Weak Fairness
- Weighted Fairness
- Last Served Lowest Priority (LSLP)

Weak Fairness means a request may have to wait indefinitely until it gets served. There may be higher priority requestors holding on to the grant. Section 2.1 discusses a Fixed Priority Arbiter that demonstrates the weak fairness

metric. The Lottery Arbiter discussed in Section 2.2 updates the weight of the lottery ticket as it arbitrates. The weight of the ticket increases the chances of winning the grant. This algorithm has the property of Weighted Fairness. The Matrix Arbiter in Section 2.3 has the property of LSLP. The last served requestor will have the lowest priority in the arbiter.

Fixed Priority Arbitration

Fixed priority arbiter is the simplest form of arbiter. It is also known as per-emptive arbiter due to the nature of its scheduling algorithm. Each master is given a priority from high to low. As shown in Eqn 2.1, master $i - 1$ has higher priority than master i[7, 9]. For master i to get the grant, all the masters higher priority than master i must not be requesting to the arbiter.

$$grant_i = req_0 \cdot req_1 \cdot req_2 \cdot \cdot \cdot req_{i-1} \cdot req$$

For example, if there are 3 masters, master 0 is given priority 0, master 1 priority 1 and master 2 priority 2. Grant is given to the master that has the highest priority. If master 0 and master 1 request at the same time, master 0 will get the grant since it has higher priority. As a result, a higher priority master can starve other masters by monopolizing the bus. However, due to the simplicity of the design, fixed priority arbiters are very useful in applications where high priority tasks need immediate servicing and low priority tasks can wait indefinitely to get grant.

Lottery Arbiter

Lottery arbitration scheduling is based on the weighted probabilistic distributions. The algorithm utilizes a lottery manager to mange the drawing of grants. As in Figure 2.1,

lottery manger gives a numbered ticket/request to each master. The weight of the ticket number is increased each time a specific master requests

Assuming a non-empty set of weights $\{w_1, w_2, ..., w_n\}$, the probability of winning a ticket can be calculated as in Eqn 2.2.

The manager draws the highest numbered ticket as a winner. The ticket count of the granted master is reset on winning the lottery. The reset makes the current winner less likely to be chosen on the next draw. In case of a tie, the manager may choose any master. If there is only one master requesting, the manger will choose the trivial solution. As a result of this pseudo-randomization, the masters get a fair share of bus time dictated by the weight of the lottery ticket.

Matrix Arbiter

Matrix arbiters are designed to enforce last served master to have the lowest priority on the shared resources. It keeps track of the priority in a square matrix form. The rows and columns of the matrix represents the requestors. The i^{th} row can be linked to requestor i and j^{th}column requestor j. Figure 2.2 shows the 4 requestors mapping in a 4 by 4 matrix.

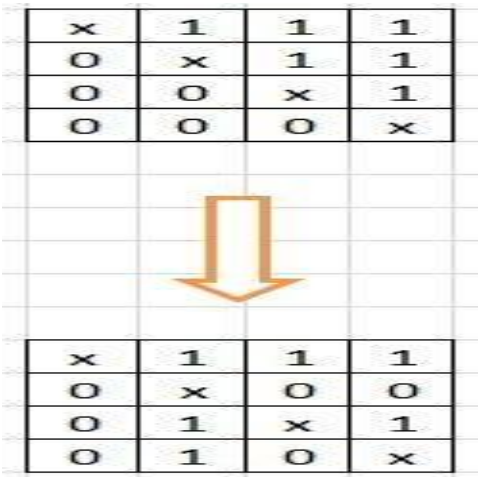
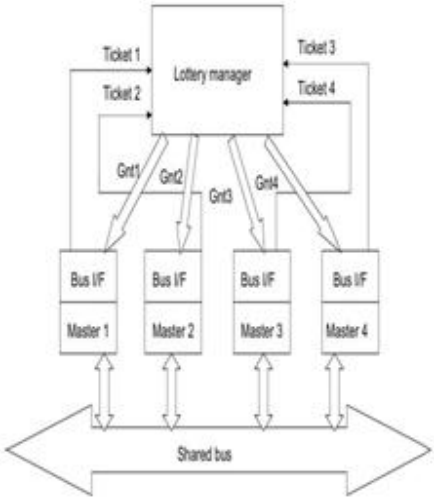


Figure 2.2: Priority Matrix

The rule of the matrix arbiter is if there is a 1 in i^{th} row and j^{th} column, requestor i has priority over requestor j. As in Figure 2.3, if requestor 2 sends a request, the grant will be issued to requestor 2. The elements in row 2 are set to zero. It forces requestor 2 to have the lowest priority. At the same time the elements in column 2 are set to 1. It makes other requestors beat requestor 2 in the next iteration. Matrix arbiters are useful when the number of inputs are small. If the number of requests increases, the structure of the arbiter increases leading to larger area overhead.



Weighted Round-Robin Arbiter

Round-robin arbitration has multiple flavors to fit the desired application. In some applica- tions two-pick Round-robin arbiters are used instead of one pick arbiters.However, the final goal, starvation prevention and statistical fairness, is the same. The algorithms introduced in Chapter give the grant to the master that has the higher priority. It means a master has the ability to monopolize the bus for a long time. This causes bus starvation to the masters with lower priorities. Weighted Round-robin arbiter design is based on the algorithm that the scheduling of grants must go on in a Round-robin manner. This work is based on a two-step approach. The arbiter monitors the requests and give them grants in the next clock. In best case condition, the request at time t_i will get serviced at time t_{i+1}. This scheduling algorithm makes sure each master gets its share of time slice in a fair amount of time. A good analogy would be if there are 4 masters in x cycle arbiter, each master will get a quantized time slice of x/4 cycles. However, in some applications, one bus master may require more bus time than others. Figure 3.1 shows top level view of Round-robin arbiter in a network packet switching system.

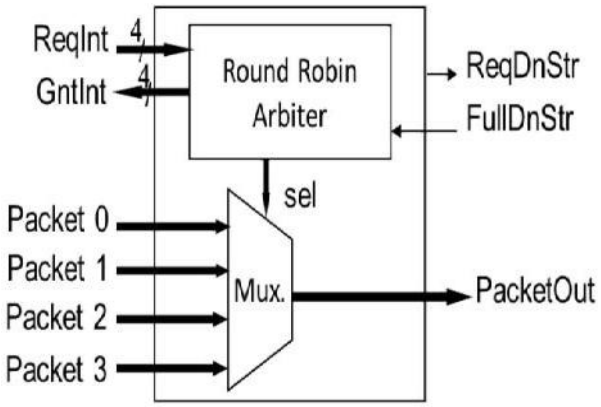


Figure 3.1: Packet Switching Architecture

This paper introduces another configurable variable called weight. The weight of each master can be defined as the grant time slice that the master can configure in the arbiter. If all the masters have the same amount of weight, each master will get an equal time share of the pie. If master A requests 20 cycles and master B requests 10 cycles, master A will get grant 2 times longer than master B. One disadvantage of letting the masters to configure the weight is a master may configure a very large weight. To reduce this large weight monopoly, another global configurable maximum allowed weight is added. A master may request a very large weight value, but the arbiter will only grant up to the maximum allowed weight if there are other masters waiting.

Weight Decoder

Weight decoder decodes one-hot grant to decode the correct weight of the granted master. As shown in Figure 3.2, the weight of the masters are concatenated to form a weight bus. As in Eqn 3.1, the width of the bus can be calculated by the width of a single weight and the total number of masters in the arbiter.

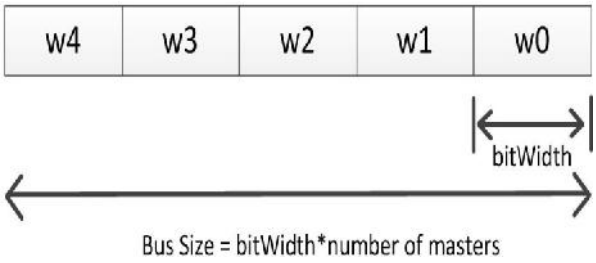


Figure 3.2: Weight Bus Size

$$busWidth = weightW\ idth * numOf\ Masters$$

Weight decoder takes current grant as an one-hot input. The input grant is decoded to produce an index for correct bit slice positions of the weight bus. Figure 3.3 shows the flowchart to produce the correct index. For example, if the grant is $b0010$, the index output is 1. Index of 1 stands for the master no. 1. The weight of master 1 is decoded as an output. If the grant is $b0100$, index output is 2. The weight of master 2 is decoded as an output.

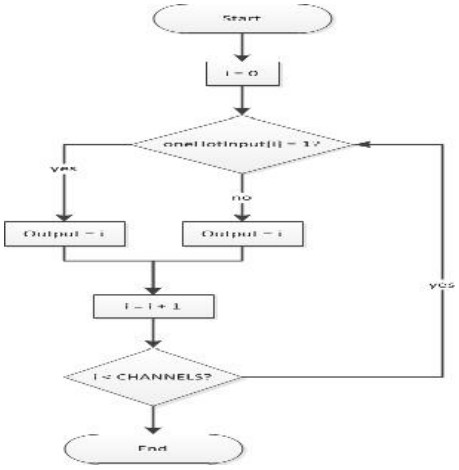


Figure 3.3: One-hot Index Flowchart

Next Grant Precalculator

Next Grant PReCalculator(NGPRC) calculates the next possible grants mask based on current grant. By precalculating the next possible grants, NGPRC dictates the Round- robin arbitration of the arbiter. As in Figure 3.4, if all 4 masters in the arbiter are requesting and current grant is master 1, next possible grant is restricted to be in the order of master 2, master 3 and master 0. The arbiter cannot skip master 2 to grant master 3. It would violate Round-robin scheme and it is not allowed. By giving next possible grant priority to the Grant State-machine, it forces the grant to be in strict Round-robin order.

Figure 3.5 shows the calculation steps NGPRC takes to compute the next possible grant priority. For example, if current grant is *b*'0010, rotate left gives *b*'0100. After inversion, the bits become *b*'1011. After increment by 1, the next possible grant becomes *b*'1100. It means the leftmost 2 bits are in line in priority.

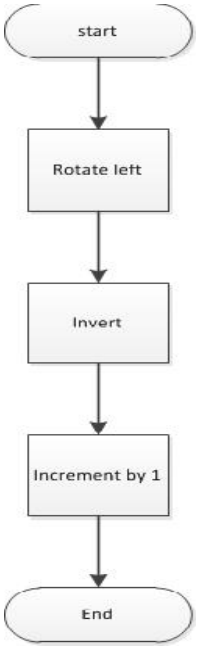


Figure 3.5: NGPRC Calculation Steps

Grant State Machine

Grant state machine is the logic to calculate which master gets the grant and for how long based on the weight. The grant logic is based on the requests and next grant priority mask created by NGPRC. Figure 3.6 shows the state flow diagram of grant state machine. "Grant Process" state masks requests using precalculated mask to grant the next requesting master. After the grant is decided, it moves to "Get Weight" state to fetch the weight of the grant from Weight Decoder. After that, it moves to "Count" state to count the clock cycles until local counter reaches the desired weight.

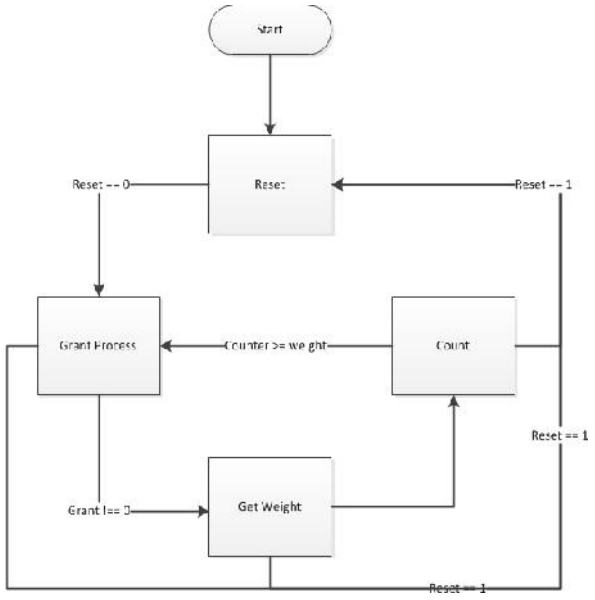


Figure 3.6: Grant State Machine

Tests and Results

This section discusses the simulation results of the arbiter, as well as the area and power overhead of the top level design.

IV. SIMULATION

Weight Decoder

Figure 4.1: Weight Decoder

Figure 4.1 shows the simulation of weight decoder. Input dataInBus contains the weights of the channels preconfigured. Input selOneHot port/grant the input used to decode the weight of current grant. The decoded weight is outputted to the dataOut port.

Next Grant PreCalculator

Figure 4.2: Next Grant PreCalculator

Figure 4.2 shows the simulation waveform of Next Grant PreCalculator. Based on the input request and grant, next grant mask is created to dictate Round-robin order to restrict the grant order.

Round-Robin Top Level/Grant State Machine

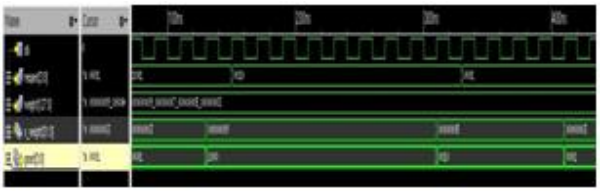


Figure 4.3: Top Level Simulation

The Figure 4.3 shows the simulation top level Round-Robin Arbiter. Grant is serviced based on the requests and precalculated mask from NGPRC. The grant is given the access time for the number of weight cycles before servicing the next request.

V. FUTURE WORK

Since the arbiter are application specific, for future work, this implementation of Round- robin arbiter can be modified to suit the intended usage. One of the applications of Round-robin arbiter is system-on-chip shared memory. In this application, two independent Round-robin arbiters are used, one for address and one for data. For read access, the two arbiters can operate independently.However, for write back operations, both the data and address needs to go together. It might be beneficial to implement a modified version that is aware of the condition when address or data arbiter needs to freeze in order to write back.

Another applications is communication arbiter for Network-On-Chip (NOC), where communication between IP cores are usually non-uniform or hot-spot in traffic.The arbiter in this work only allow a fixed time slice preconfigured. It would be beneficial to implement logic to detect the load of the inputs and adjust priority dynamically. By adjusting priority or grant time based on the traffic would make sure that busy master/re- questor traffic is well balanced and not starved.

REFERENCES

[1] K. Warathe, D. Padole, and P. Bajaj, “A Design Approach to AMBA (Advanced Microcontroller Bus Architecture) Bus Architecture with Dynamic Lottery Arbiter,” in *2009 Annual IEEE India Conference*, Dec 2009, pp. 1–4.

[2] Z. Fu and X. Ling, “The design and implementation of arbiters for Network-on-chips,” in *2010 2nd International Conference on Industrial and Information Systems*, vol. 1, July 2010, pp. 292–295.

[3] C. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. New York NY: Springer, 2006.

[4] Y. Li, N. Zeng, W. N. N. Hung, and X. Song, “Enhanced symbolic simulation of a round-robin arbiter,” in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Oct 2011, pp. 102–107.

[5] S. Q. Zheng and M. Yang, “Algorithm-Hardware Codesign of Fast Parallel Round- Robin Arbiters,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 1, pp. 84–95, Jan 2007.

[6] M. Abdelrasoul, M. Ragab, and V. Goulart, “Impact of Round Robin Arbiters on router’s performance for NoCs on FPGAs,” in *2013 IEEE International Conference on Circuits and Systems (ICCAS)*, Sept 2013, pp. 59–64.

[7] Y. Yang, R. Wu, L. Zhang, and D. Zhou, “An Asynchronous Adaptive Priority Round- Robin Arbiter

Based on Four-Phase Dual-rail Protocol," *Chinese Journal of Electron- ics*, vol. 24, no. 1, pp. 1–7, 2015.

[8] K. A. Helal, S. Attia, T. Ismail, and H. Mostafa, "Priority-select arbiter: An efficient round-robin arbiter," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, June 2015, pp. 1–4.

[9] R. Kamal and J. M. M. Arostegui, "RTL implementation and analysis of fixed pri- ority, round robin, and matrix arbiters for the NoC's routers," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, April 2016, pp. 1454–1459.

[10] E. S. Shin, V. J. Mooney, and G. F. Riley, "Round-robin Arbiter Design and Gener- ation," in *15th International Symposium on System Synthesis, 2002.*, Oct 2002, pp. 243–248.