

Round-Robin Arbiter Design

Jinming Ge
Engineering & Computing
Wilberforce University
Wilberforce, OH, U.S.A.

Abstract - Round-robin has been used as a fair (non starvation) scheduling policy in many computer applications. This paper presents a novel hardware design of a round-robin arbiter without any misses – It always grants an available resource to one of legitimate requests, which may be very unevenly generated from various sources. The design is modeled in HDL, logically verified and then synthesized targeting an ASIC technology. The speed and cost of this arbiter, compared with other designs, make it promise well for performance improvement in systems with potential non-uniform requests.

Keywords: Round-robin arbiter, Application Specific Integrated Circuit (ASIC), Synthesis, Logic Verification.

1.0 Introduction

Round-robin has been used as a fair (non starvation) scheduling policy [1] in many computer applications, either implemented in software as part of an operation system or in hardware, especially for real-time applications. Figure 1 shows round-robin arbiters used in resolving conflicted resource (channel) requests for one channel (positive direction along x dimension, x+) in a wormhole router [2]. There are three virtual channels (buffers) sharing the same physical channel and each may be requested to accept data (flit) from several input channels according to the routing algorithm used. So besides the arbitrator used to manage the shared physical channel from three virtual channels, there is one arbiter for each output virtual channel. All possible requests to a certain output channel are counted as the maximum requests, which contribute directly to the complexity of the arbitrator. When a fully adaptive routing algorithm is used, the output buffer such as an ejector can have as many as thirteen simultaneous requests for a two dimensional router. Under certain network traffic pattern, the requests can be very unevenly generated by the source (input) channels.

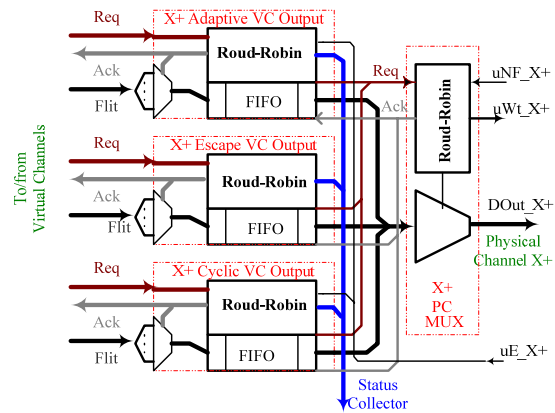


Fig. 1 Round-robin arbiters used in the X+ channel of a wormhole router.

A round-robin arbiter is used to resolve conflicting requests generated from various sources for a shared resource in a directional and cyclic order. The requests are generally tabulated by a fixed order in a multiple bit register with each bit corresponding to a specific request. An asserted bit meaning a request

is issued from the source (*candidate*). At most one request can be granted at any time – this is defined as a turn being given. If the turn is given in a circular manner, it can be viewed as an exclusive *token* cycling around. If the request is in the same position as of the token being asserted, it is called a *turn-hit*, otherwise it is a *turn-miss*. Fig.2 illustrates both the hit and the miss cases for an arbiter managing six candidates. The token is cycling around by shifting to the right each cycle. It gives turn to the second candidate but that one does not request so the turn is missed. Although along the shifting direction there are asserted requesting candidates, the nearest one is the fourth one, it must wait until its turn after two cycles later assuming the request still persist. Apparently there are much more cycles wasted in an arbiter managing more candidates especially when they generate requests very unevenly. The arbiter generates the acknowledgement or grant signal if both of the following conditions are satisfied:

- There is really a request, or the request bit is asserted;
- The turn is exclusively given to this request, or a turn-hit.

Although the design of the arbiter in this way is quite straight forward and fast, it may potentially degrade system performance in which existing requests for service are denied or delayed for a long time and may force the candidate to switch request for other possible resources (maybe already busy) – if not, the system basically denying service while the requested resource is available. If the arbiter is designed in such a way that it can skip the non-requesting candidate and find and grant the closest asserted candidate along its searching direction, it can improve overall system performance.

The next section discuss three designs of the arbiter starting from a brief analysis of the requirement for skipping logic in a simple design, then developing a hierarchy to enhance arbitrator performance when skipping is used. A conclusion and further research direction is discussed in section three.

2.0 Round-Robin Arbiter Design

Three basic steps are involved in the design of an arbiter: modeling, logic verification, and synthesis. The design is first modeled and may be represented in schematic graphs. The model is then described in Verilog HDL [3] and the logic can then be verified by using simulation tools like SILOS. Finally, the design can be mapped into a specific process technology, LSI G11-p [4] here, to be synthesized and optimized for area cost and clock frequency [5].

The synthesis tool used is Synopsys, which optimizes a design constrained either by timing or chip area [6] [7]. The same design can achieve a higher clock frequency by using more space after optimization. When comparing different design strategies of the same logic implementation, timing is used as the primary target in this research.

2.1 A Simple Round-Robin Arbiter

A simple round-robin arbiter can be built with a

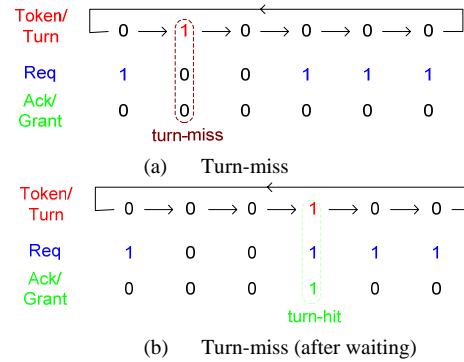


Fig. 2 Turn-hit and turn-miss in a simple round-robin arbiter.

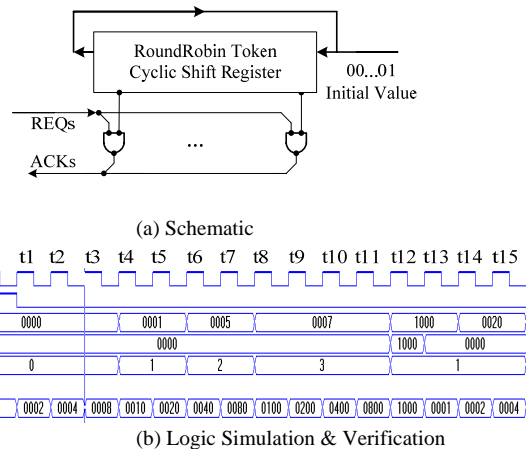


Fig. 3 A Simple Round-robin arbiter

shift register and several AND gates, as shown in Fig. 3. The turn is initialized to give the rightmost request only and then the exclusive turn-tag is cyclically shifted left at each cycle.

The arbitration designed by this way is fair, but will be suboptimal in the following situations:

- If there are not many candidates to compete for the resource (one non-uniform distribution of requests), then the probability that the token hits the request becomes very low. This leads to either turning the turn-missed request for another resource (if does exist) that may have been already very ‘hot’, or delaying its service.
- If the arbitrated resource is very non-uniformly ‘hot’, the asserted requests take a few consecutive bits but nothing in other bits. A miss in the consecutive blank bit may lead to prolonged service for each of the requests.

As shown in Fig.3b for 16-candidate arbitration, the request from the lowest-numbered candidate at t4 cannot get serviced for 9 cycles, even when there are no other pending requests at all, and has to turn away at t12.

2.2 Arbiter with Naïve Skipping Logic

To eliminate the performance loss of turn misses, the arbiter should keep searching one by one in the cyclic direction, from the last turn hit until another hit. The circuit, as shown in Fig. 4, consists of a token register RRT, like a barrel shifter [8], which keeps the token and the last turn given, and a combinational skipping logic (SL) circuit. This tries to find the nearest asserted request along a certain direction in case there are any asserted bits. If the nearest request bit is not asserted, it will check the next one until a nearest bit is asserted. The first asserted request will be granted service, and SL will refresh RRT with the current turn given.

When the number of candidates being arbitrated increases, the skipping logic will not only take more chip area, but also significantly slow down. The arbiter has two basic operations, matching and loading. The matching is to confirm whether the tabulated request bit, one by one along pre-defined direction starting from the last token position, is valid. If the nearest asserted request is found, the RRT is re-loaded with a new token to specify the next turn-given. If the matching operation on each of the n candidates takes T_m ns and the re-loading takes T_l ns, the arbiter speed is limited by the time T_a ,

$$T_a = (n-1) T_m + T_l,$$

because in the worst case, the arbiter has to skip (n-2) times.

2.3 Effective Hierarchic Skipping

The cycle time of the naïve skipping approach is of $O(n)$, which may be on the critical path of the overall router design. To speed up the arbiter, the arbitrated candidates can be re-tabulated properly in groups and skipped hierarchically, as described in Fig. 5.

The hierarchy has a topology similar to a tree. For example, a total number of $n = 2m$ candidates can be partitioned into two groups and each has m candidates to be arbitrated. The top level then just needs to handle 2 requests, and has a 2-bit turn register. Each of these two requests is a reduction of a group, that is, if any of the request bit in the

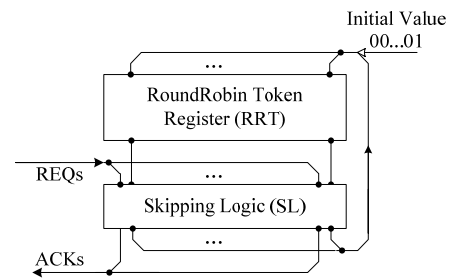


Fig. 4. A round-robin arbiter with skipping logic.

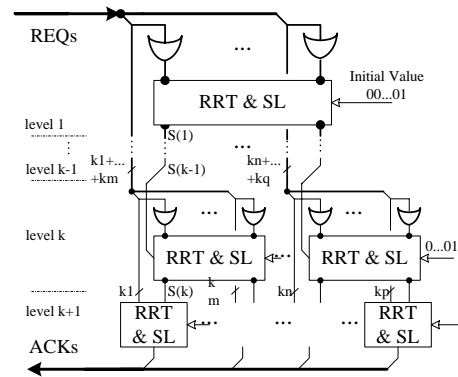


Fig. 5. An arbiter designed by grouping and hierarchic skipping

group is valid, the top level corresponding request is set. The naïve skipping logic works independently at each hierarchic level, and at each smaller group of the same level. After finishing its own operation, one of the lower level groups will get the turn (token) passed from its parent node and its generated turn is passed to its child node. The logic works in a manner of a chain, and the leaf node will finally grant the request.

The logic is verified as shown in Fig. 6 for 13 candidates partitioned into three levels. Each level has 1, 2, 4 node(s) respectively from top to low level, and the candidates are grouped as $\{\{4\} \{3\}\} \{\{3\} \{3\}\}$. The register RRu, RRm1, RRm0, RRd3, RRd2, RRd1, RRd0 is used to log the token position in each node, and all of them are initiated with a token on the lowest position. At t4, only the lowest numbered request is asserted. After the request is granted, the corresponding token registers RRu, RRm0 and RRd0 are refreshed by shifting the token to the next. All others remain unchanged as the given turns have not been used and there are no requests under their control. RRd0 at t6 is unchanged because the tokenized request is not asserted, but it is skipped to the nearest asserted request, 4, and the token is shifted to its next position (cyclically), the first position at next cycle. At t7, the request is granted because of the turn hit.

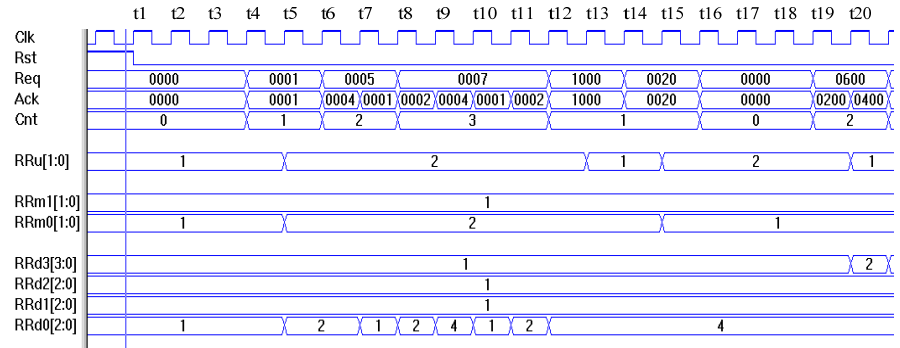


Fig. 6. Logic verification of arbitration of 13 candidates

The cycle time of the arbiter (T_{a_g}) depends on the maximum time of its node operations ($T_{a_{gx}}$) and the maximum time for a token passing along the chain (T_p):

$$T_{a_g} = T_{a_{gx}} + T_p.$$

$T_{a_{gx}}$ is much smaller than T_a . Also, T_p can be small if the hierarchy depth is properly designed. Therefore T_{a_g} can also be smaller than T_a .

2.4 Comparison of Three Designs

Grouping and hierarchy skipping can be faster than naïve skipping if constrained by the same cost (chip-area). Table 1 shows the cost and speed of three different designs for arbiters in the router. To be consistent with each other, all the synthesis optimizations are based on the timing constraint. The partition for the cases with 6, 10, 13 candidates is optimized as $\{\{3\} \{3\}\}$, $\{\{3\} \{2\}\} \{\{3\} \{2\}\}$ and $\{\{4\} \{3\}\} \{\{3\} \{3\}\}$ respectively. It can be seen that

- Skipping logic, although necessary, not only increases the chip area cost, but will also slow down the arbiter.
- Grouping and hierarchy skipping is more cost-effective than naïve skipping. Although the initiative to speed up is not as much as expected in the case of 13-candidate arbitration, the cost is reduced. To view this in another way, if under the same budget, it can be much faster.

TABLE 1
THE COST AND SPEED OF ROUND-ROBIN ARBITERS

# Candidates	Simple		Skipping		Hierarchy Skipping	
	Area (cells)	Timing (ns)	Area (cells)	Timing (ns)	Area (cells)	Timing (ns)
2	80	.20	204	.34		
3	187	.27	444	.49		
6	473	.30	1,908	.72	1,523	.59
10	906	.35	3,875	.88	3,830	.69
13	1,232	.39	7,324	.94	6,524	.76

3.0 Conclusion

Skipping unasserted requests may potentially improve system performance, but introducing skipping into hardware design slows down a round-robin arbiter's clock cycle and increases its chip area cost. A novel design by using hierarchy grouping and skipping is a much better approach as illustrated in this paper in terms of chip cost and speed. As the skipping approach is targeting resource scheduling in systems with non-uniform distribution of request, using this approach for system with uniform request distribution is not recommended due to the cost increase and speed decrease.

A general design guideline for hierarchy grouping, in terms of both mathematic relation and circuit model will be a further topic to be investigated. Another topic is to implement a reconfigurable arbiter for both skipping and non-skipping operations based on either user's specification or recent history of request distributions.

References

- [1] Silberschatz, A., "Operating System Concepts". John Wiley & Sons, 2003.
- [2] Ge, J., "Cost-effective Buffered Wormhole Routing". Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, Vol. 3358, pp666-676, 2004.
- [3] Palnitkar, S., "Verilog HDL: A Guide to Digital Design and Synthesis". SunSoft Press, Palo Alto, CA, 1996.
- [4] LSI Logic Corporation, "G12 Cell-Based ASIC Library Release Notes", Oct. 1999.
- [5] LSI Logic Corporation, "Using LSI Logic and Synopsys Design Tools and Libraries", Oct. 1998.
- [6] Synopsys, Inc., "Design Compiler Reference Manual: Optimization and Timing Analysis", 1997.
- [7] Synopsys, Inc., "Design Compiler Reference Manual: Constraints and Timing", 1997.
- [8] Katz, R.H., "Contemporary Logic Design". The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1995.