Index-based Round-Robin Arbiter for NoC Routers

Masoud Oveis-Gharan and Gul N. Khan
Department of Electrical and Computer Engineering
Ryerson University
Toronto, Ontario M5B 2K3 Canada
gnkhan@ee.ryerson.ca

***Abstract*—Scalable on-chip communication system such as Network-on-Chip (NoC) is needed to meet the communication demand of large number of SoC (System on Chip) cores. In the NoC router micro-architecture design, arbiter has become increasingly important due to its significant impact on the performance and efficiency of NoC systems. In this paper, we propose an Index-based Round Robin (IRR) arbiter that functions on the index format of input ports of the router. The microarchitecture of IRR arbiter scales logarithmically ($\log_2$) with the number of input ports as compared to a conventional round robin arbiter that scales with its input ports. The behavior and architecture of our arbiter leads to lower power consumption and chip area as well as higher performance characteristics.**

Keywords—arbiter micro-architecture; critical path delay; network on chip; router arbiter design

I. INTRODUCTION

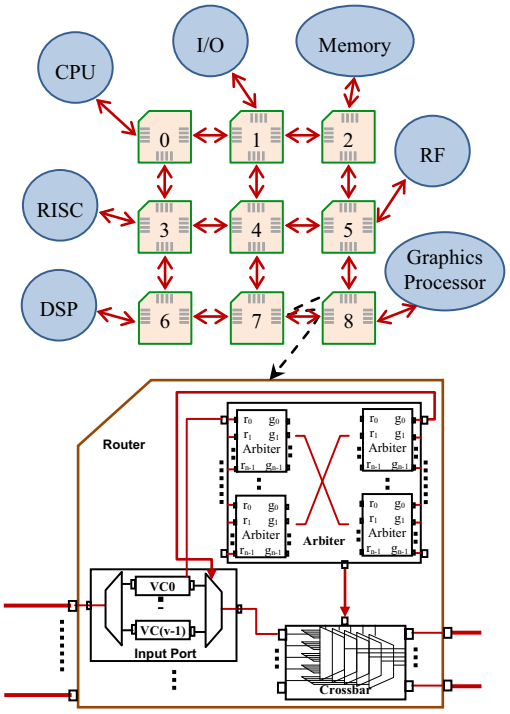
In digital system design, arbiters are used to allocate and access shared resources. Whenever a resource, such as a buffer, channel or a switch-port is shared, an arbiter is required to assign the access to the resource at a particular time. The most common usage of arbiters is the shared-bus arbitration of a bus-based system where multiple master modules can initiate their transactions. The modules must be arbitrated for access to the bus before initiating a transaction. In this paper, we investigate the arbiters used in NoC systems. An NoC system facilitates communication among IP cores of an SoC. It includes a network of switches (routers) that are interconnected by communication links as illustrated in Figure 1.a. Figure 1.b shows a typical NoC router that consists of some input and output ports, an arbiter and a crossbar switch [1].

The datapath of a router is made of port buffers, crossbar switch and interconnection structure, while the control unit of a router is mainly consists of arbiters. The structure of arbiter becomes even more complex with the utilization of Virtual Channel (VC) mechanism. Figure 2 shows two 4-input arbiter that can arbitrate the use of resources. The arbiter accepts n requests ($r_0, r_1, \ldots, r_{n-1}$), arbitrates among the asserted request lines, and selects an r_i for service, and then asserts the corresponding grant line, g_i. For example, assume the arbitration for the output port of a crossbar switch among a set of requests from the VCs of some input ports. The input-port VCs that have flits will issue request signals for having access to one of the desired output-port. Assume, there are 5 VCs and VCs 0, 2, and 4 assert their request lines, r_0, r_2, and r_4 respectively. The arbiter will then arbitrate and select one of these VCs for assigning the desired output-port. Assume the grant of VC2 (i.e. g_2) is asserted. VCs 0 and 4 lose the arbitration and must hold their requests active until they receive the grant signal for their output-ports.

Arbiters can be categorized in terms of fairness (weak, strong or FIFO) arbiters [1]. In a weak fairness arbiter, every request is eventually granted. The requests of a strong fairness arbiter will be granted equally often. The requests of FIFO fairness are granted in a first come first served basis. Moreover, arbiters in terms of priority can be grouped in two fixed and variable architectures. For a fixed priority arbiter, the priority of requests is established in a linear order. Figure 3a illustrates a 4-input fixed-priority arbiter where r_0 has the highest and r_3 has the least priority [1]. The architecture can be expanded to n-input arbiter where for each middle request, there is an arbiter cell consisting of two ANDs and an Inverter. The first and last arbiter cells can be simplified (see Figure 2a). For each request input ri, there is a carry input c_i, a grant output g_i, and a carry output, c_{i+1} where $i \in \{0,1, .. n-1\}$. Therefore, a low level c_i indicates that at least one of requests from r_0 to r_{i-1} was has been asserted. Moreover, in case that the request ri and carry c_i are high, the grant, g_i is set, and all the following grants i.e. g_{i+1} to g_{n-1} will become reset. It is obvious that the critical path of the circuit is from the first request, r_0 to the last grant, g_{n-1} due to propagation of carry from head to the tail of the arbiter. Fixed priority arbiters provide weak fairness arbitration because when a request is continuously asserted, none of its following requests will ever be served.

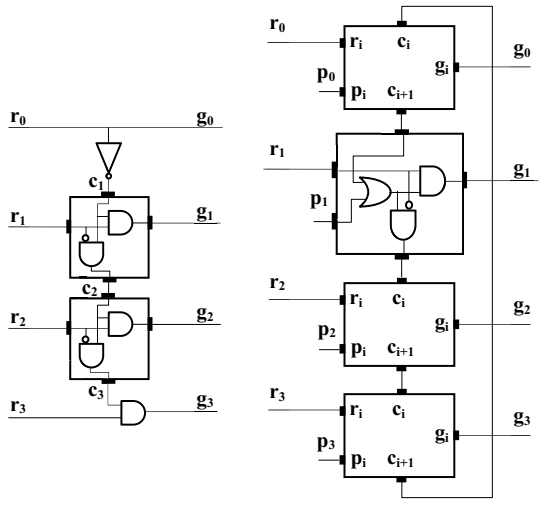
In order to have a fair iterative arbiter, we can use a variable priority arbiter as illustrated in Figure 2b. An OR gate and a priority input signal, p_i is added to each cell of the fixed priority arbiter shown in Figure 2a. When p_1 is set, its corresponding request, r_1 has high priority and the priority decreases from that point cyclically around the circular carry chain. Now we can create a fair iterative arbiter by changing its priority from cycle to cycle. In an n-input arbiter, if the grant, g_i (where $i \in \{0, 1, \ldots n-1\}$) is connected to the next priority vector p_{i+1}, a Round Robin (RoR) arbiter is created. Figure 3 illustrates a 4-input RoR arbiter. If a grant, g_1 becomes high at the current cycle, it causes p_1 to be set high on the next clock cycle. This leads the request, r_2 to become the highest priority at the next cycle, where the request, r_1 becomes the lowest priority. For the sake of simplicity, we assume that the arbitration cycle takes one clock cycle in all the architectures describe in this paper.

978-1-4799-8719-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ISVLSI.2015.27



(b)

Figure 1. (a) 2D SoC mesh. (b) Wormhole NoC router.



(a) fixed priority (b) variable priority

Figure 2. 4-input arbiter architectures.

The functionality of a round-robin arbiter can be explained as a request that is just granted will have the lowest priority on the next arbitration cycle [1]. The round robin arbiters are simple, easy to implement, and starvation free. When the input requests are large in numbers, the structure of round robin arbiter grows that leads to large chip area, higher power consumption, and critical path delay. In an NoC design, the critical path delay of arbiter usually dominates among the critical path delays of input-port and crossbar switch due to the architectural complexity of arbiter as compared to those of port and crossbar switch. Therefore, the arbiter circuit determines the maximum frequency (or the speed), F_{max} of an NoC router. The critical impact of arbiter on the performance of the NoC system and the characteristic behaviour of round robin architectures have created a lot of interest of NoC researchers.

II. Related Work

The architecture of a popular Matrix round robin arbiter is presented by Dally and Towles [1]. A 4-input Matrix arbiter architecture is shown in Figure 4. It implements a least recently served priority scheme where a request, r_i wins an arbitration. It resets the bits of row i and sets the bits of column i to make itself the lowest priority where $i \in \{0,..3\}$. The Matrix arbiter is claimed to be useful for small number of inputs as it is fast, economical, and performs strong fairness arbitration. However, no evaluation is presented. Fu and Ling evaluated and compared the RoR and Matrix arbiters in terms of resource, performance and power consumption for an FPGA platform [2]. They concluded that the Matrix arbiter consumes more resource, same power but can process data more quickly than the RoR arbiter.

Zheng and Yang proposed a Parallel Round Robin Arbiter (PRRA) based on a simple binary search algorithm as illustrated for a 4-input PRRA in Figure 5 [3]. They further proposed an Improved PRRA (IPRRA) design where the output signals, *gL* and *gR* of PRRA are disconnected and directly ANDed with grant signals to generate new grant signals as shown in Figure 6. The IPRRA reduces the timing of PRRA significantly. A High speed and Decentralized Round robin Arbiter (HDRA) has been presented by Lee et al., which is illustrated in Figure 7 [4]. Each circuit enclosed with dash circle represents a filter circuit whose main components are a D flip-flop and a multiplexer. The filter circuit filters out the input without request or the one with request that has already been granted at that arbitration cycle. The un-filtered inputs with their requests participate in the arbitration again next cycle by setting its corresponding D-type flip-flops to 0 that are done by enabling the *ack* signals from higher lower level. The HDRA arbiter will reset itself asynchronously by the input *self_rst* from the root. The *sys_rst* indicates the system reset signal and is used initially before each arbitration cycle for all requests. A 4-input HDRA arbiter has a simpler circuit than a higher input HDRA architecture because the *act, rnext* and *self-rst* are connected together.

III. Index-based Round Robin Arbiter

In this paper, we present a new arbiter design called Index-based Round Robin (IRR) arbiter that employs a least recently served priority scheme and achieve strong fairness arbitration. The proposed arbiter has smaller arbitration delay, lower chip area and it also consumes less power as compared to the aforementioned arbiters. Before describing the IRR arbiter architecture, we introduce an inseparable and critical output in the arbiter design.

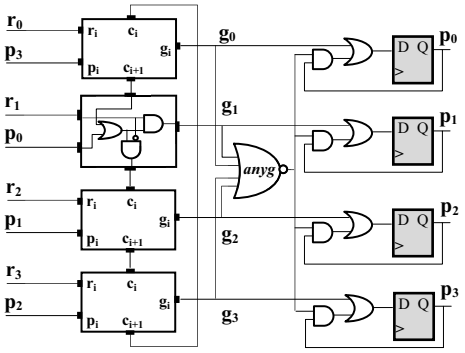


Figure 3. 4-input RoR arbiter architecture.

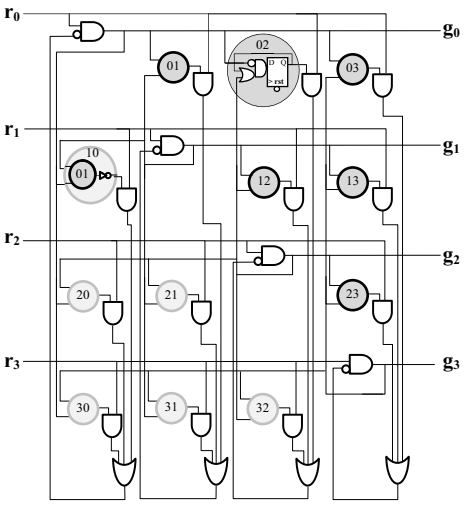


Figure 4. A 4-input Matrix arbiter.

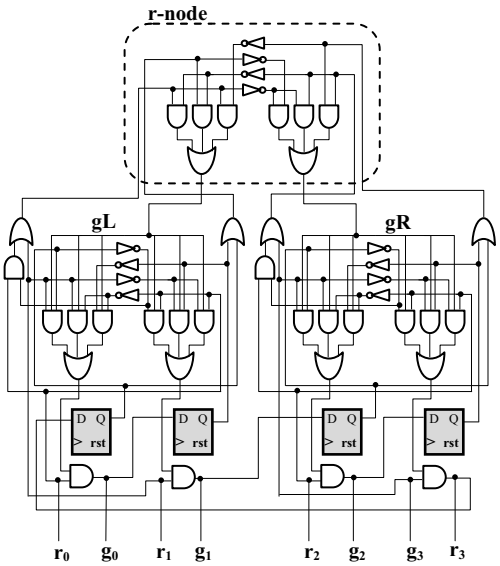


Figure 5. 4-input PRRA architecture.

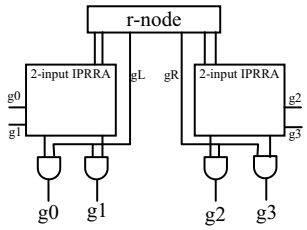


Figure 6. 4-input IPRRA structure.

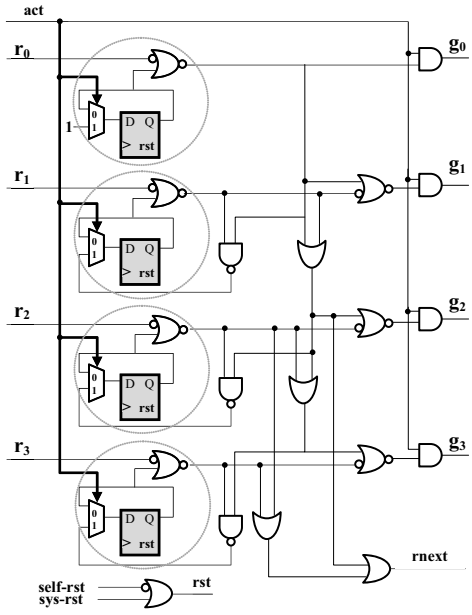


Figure 7. 4-input HDRA architecture.

A. Grant Index

All the arbiters have an output array, *grant* whose width is the same as that of input width. However, in practical designs, the index of *grant* signals, *g_id* is also generated that is used to address the granted request in some other components, such as control tables, multiplexers and memories used in NoC routers. When a router crossbar is made of multiplexers, the *g_id* can be connected to selection port of multiplexer to switch the granted input to the requested output port (see Figure 1.b). The width of *g_id* is the $\log_2$ of the width of *grant*. We used the *g_id* as the first output of our proposed design and due to lower width of *g_id*, our arbiter design is smaller and faster as compared to other arbiters. Due to the critical use of *g_id* in NoC design, we consider all the arbiters covered in this paper to generate both *grant* and *g_id* as outputs.

B. Fixed Priority Arbiter

Our fixed priority arbiter is simpler and economical as illustrated in Figure 8. The priority of requests is linear and in the ascending order where r_0 has the highest priority. The index of first asserted request is switched to the output as the index of *grant*, *g_id*. Then the *g_id* is decoded to create

the *grant* signals. The last request, r_{n-1} has a simplified circuit where instead of being multiplexed like other requests, it is ANDed by g_{n-1}. Therefore, in case that only r_{n-1} is high, then only g_{n-1} becomes high.

C. Variable Priority Arbiter

If the *g_id* output of the fixed priority arbiter of Figure 8 is connected to the last multiplexer, each request behaves as it has the highest priority through ascending order of the loop. For example, for four requests (r_0, r_1, r_2 and r_3) the output of multiplexer, *M1* generates an index where r1 has the highest priority then r_2, r_3, and r_0. Therefore, by further multiplexing of these outputs, we can choose an input as the highest priority request as shown in Figure 9. For example, when *P*=1, the output of multiplexer *M1* is selected and the request, r_1 has the highest priority. In the case of no request asserted, or r_0 is asserted, the *g_id* issue the same value (i.e. zero). In order to separate these two conditions, ORing of requests, *any_r* is ANDed with the g_0 so that when all the requests are zero, all the *grant*s will become zero.

D. IRR Arbiter

If the next index of granted request is chosen for the next priority selection, the current granted request receives the least priority, and its next request receives the highest priority among all the requests. To accomplish it, the *g_id* array is stored in a register, and the output of the register is incremented and connected to the selection port of multiplexer, *MP* as shown in Figure 10. In this way, the arbiter follows the least recently served priority scheme or a round robin scheme. Figure 12 illustrates our proposed IRR arbiter that takes one clock cycle for arbitration. To keep the priority unchanged, the output of priority register, *next_g* is fed back through the SF multiplexer to the register to cater for no request. It guarantees a strong fairness arbitration in our design we discuss later.

E. IRR Arbiter Functional Behaviour

We present the functionality and behaviour of our round robin arbiter illustrated by its timing diagram in Figure 11. During time 1-6, a fixed input request, "1111" is applied and granted bit by bit per clock cycle. At time 6, the request is changed to "0000", i.e. no request is asserted. For no request situation, the priority of last granted request is recorded and applied when a new request is asserted. For example, at time 6, the priority of second bit of request is recorded and applied at time 8. Consequently, the forth bit of request is granted at time 8. We tested our arbiter along with some past arbiters (RoR, Matrix, PRRA, IPRRA, and HDRA) with the same testbench and same request scenario as illustrated in Figures 11 and 12. When no request is asserted, the RoR, Matrix and our IRR arbiters record the current priority shown in Figure 11. However, the PRRA, IPRRA and HDRA couldn't record the priority and show different waveforms shown in Figure 12. For no request condition for PRRA, IPRRA and HDRA waveforms, the highest priority is given to the least significant bit of the request. This arbitration behavior of PRRA, IPRRA and HDRA is due to no circuit to handle the no request condition. Keeping the last priority during no request condition has direct impact on the fairness of an arbiter. The first advantage of our IRR arbiter is aits stronger fairness arbitration.

F. Simpler Incrementing Hardware

The adder in IRR architecture performs an incrementing operation and it can be built in a simple form. For incrementing an *n*-bit operand, one can connect *n* full adder cells in serial form, and feed 1 to carry input of the first cell and zero value as the second operand. Zero value for the 2nd operand enables us to use half adders instead of full adders.

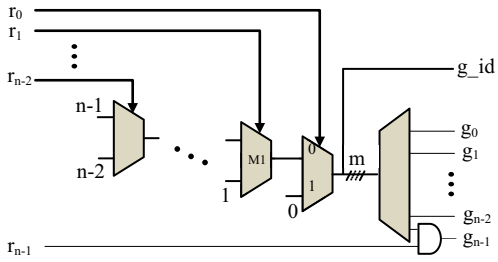


Figure 8. *n*-input fixed priority arbiter, where $m = \log_2(n)$

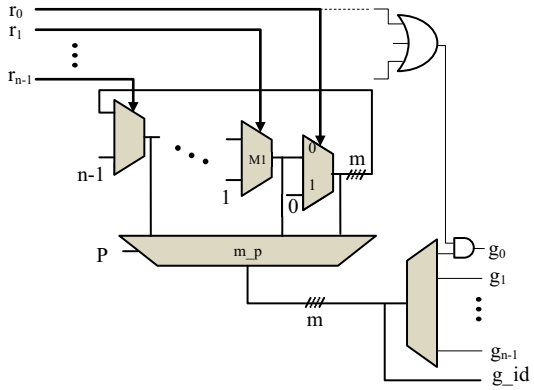


Figure 9. *n*-input variable priority arbiter.

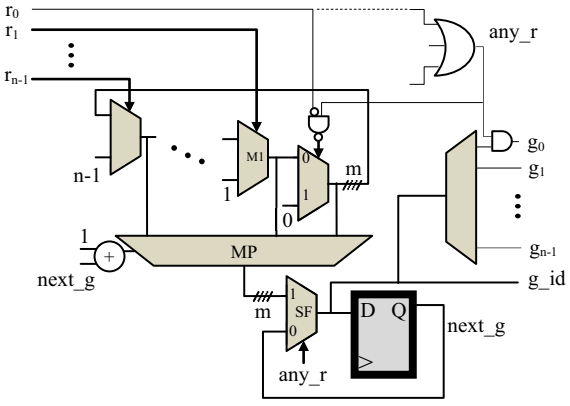


Figure 10. n-input IRR arbiter, *where m=log₂(n).*

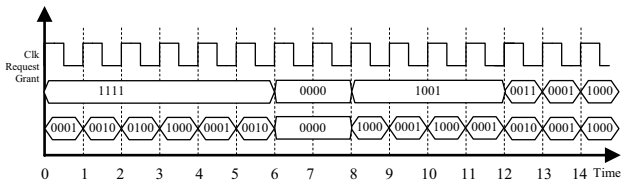


Figure 11. Input request scenarios for strong-fairness RoR arbiters.

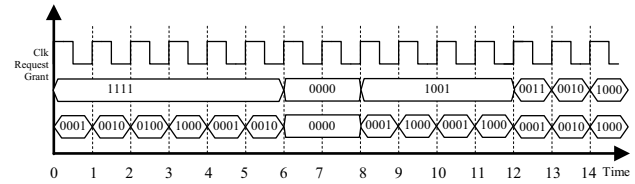


Figure 12. Input request scenarios for weak fairness RoR arbiters.

G. Analytical Comparison

We perform a hardware analysis to compare the expected performance and hardware overhead of the aforementioned round robin arbiters with our proposed IRR arbiter. We don't apply any algorithm to optimize the circuits as an Electronic Design Automation software does. The main figures of merit of an arbiter circuit are speed, area and power consumption. The usual measure for speed of an arbiter circuit is the delay time or maximum clock frequency (F_{max}). The clock frequency of an arbiter depends on the longest delay (critical path) between two registers clocked at the same time. The circuits of 4-input arbiters, which are shown in Figures 3, 4, 5, 6, 7 and 10, are decomposed at the gate level. The electrical parameters of the logic gates are derived from Synopsys 90nm Digital Standard Cell Library as listed in Table I. We calculated the summation of the areas and powers of all the cells of each arbiter to estimate their power and area. The power includes both static and dynamic powers. For speed estimation, the critical path delay between two registers of each circuit is calculated. We have further illustrated the critical path of each circuit through the number in parentheses for the last column of Table I. The increment is done by a half adder (XOR2x1) propagation delay.

As discussed earlier, RoR and Matrix arbiters have strong fairness, and the HDRA, PRRA and IPRRA has weak fairness. Therefore, we introduce a weak fairness version of IRR named IRR_WF for comparison with the HDRA, PRRA and IPRRA arbiters. The only difference between IRR and IRR_WF is the SF multiplexer shown in Figure 10 and it affects the critical path delay. We expect a faster IRR_WF arbiter than IRR. Table II list the characteristics of all the arbiters. IRR has the optimum performance and hardware overhead among all the listed arbiters. In terms of speed, the IRR can run 15% to 50% faster than the other arbiters. Moreover, the IRR saves the area from 10% to 47% and power from 1% to 44%. We also evaluate our IRR arbiter with the other arbiters in terms of area, power and timing by using the EDA tool.

TABLE I. ELECTRICAL PARAMETERS GATES FOR SYNOPSYS 90nm

Gate name	Propagation Delay (ps)	St. Power (nW)	Dy. Power (nW/MHz)	Area (μm^2)
INVX1	38	88	12	6.5
AND2X1	85	298	19	7.4
AND3X1	119	297	34	8.3
NAND2X1	51	336	15	5.5
OR2X1	85	226	23	7.4
OR3X1	114	250	39	9.2
OR4X1	137	261	56	10.1
NOR2X1	64	170	15	6.5
MUX21X1	107	815	43	11.1
MUX41X1	168	827	58	23.0
DEC24X1	119	1238	66	29.5
XOR2X1	133	454	26	13.8
DFFARX1	217	620	100	32.2

TABLE II. CHARACTERISTICS OF 4-INPUT ARBITERS BASED ON TABLE I

Type of 4-input arbiters	Area (μm^2)	Power (μW)	Critical Path Delay (ps)
IRR	294	296(282[d])	625 (217+133+168+107)
RoR	328	298(289[d])	1242(217+5*(85+85)+137+38)
Matrix	556	479(465[d])	747 (217 +2*38 +3*85+114+85)
IRR_WF	280	274(262[d])	518 (217+133+168)
HDRA	431	360(348[d])	609 (217 +64+51+85+85+107)
PRRA	510	493(479[d])	861(217+2*38+3*85+85+2*114)
IPRRA	528	488(473[d])	747 (217+2*38 +3*85+85+114)
Arbiters	Saving	Saving	Faster
IRR/RoR	10%	1%	50%
IRR/Matrix	47%	38%	16%
IRR_WF/HDRA	35%	24%	15%
IRR_WF/PRRA	45%	44%	40%
IRR_WF/IPRRA	47%	44%	31%

[d] dynamic power

IV. EXPERIMENTAL RESULTS

IRR (IRR_WF) arbiter is evaluated and compared with other arbiters. We consider IRR_WF for comparison with the HDRA, PRRA and IPRRA arbiters. To analyze the speed, area and power overhead, all the arbiters are implemented in structural RTL Verilog and synthesized using the Synopsys Design Compiler (DC) for 90nm Synopsys Generic Library and Altera FPGAs (Stratix V). The resulting designs operate at 400 MHz and 1.2 Volt. Different structures of each arbiter are synthesized, and their results for total cell area, critical path delay, and power (dynamic and static) consumption are listed in Tables III and IV. The Synopsys tool runs different algorithms iteratively to find an optimum architecture for a design. There is always a trade off among the three characteristics i.e. power, area and critical path of a design. This trade off behaviour dictates us to the efficient arbiter design that has smaller power, area and critical path delay. The critical path is the limiting factor preventing us from decreasing the clock period.

For a fair comparison, we group the arbiters into strong and weak fairness arbiters and present their results in Tables III and IV. Table III lists the IRR, RoR and Matrix arbiter characteristics. We listed two synthesised versions of IRR at 16 and 32-input configurations indicating the IRR having efficient area, power and timing characteristics than RoR and Matrix arbiters. It consumes on average 19% and 70%

less chip area, 44% and 74% less power (static and dynamic), and 43% and 7% shorter critical path delay as compared to RoR and Matrix arbiters respectively. The results of 4-input arbiters follow the analytical results of Table II.

Table IV data indicates that IRR_WF is more efficient as compared to PRRA and IPRRA arbiters for all the input configurations. It saves in average 16% and 22% less chip area, 46% and 45% less power, and 34% and 24% shorter critical path delay as compared to PRRA and IPRRA arbiters. The IRR_WF is also more efficient as compared to HDRA for input structures. It saves 13% less chip area, 44% less power, and 15% shorter critical path delay on average. Overall, IRR and IRR_WF arbiters consume the least power among all the arbiters due to its usage of fewer registers. The fewer number of registers of IRR also leads to simpler design and chip layout due to simple clock tree organization. To illustrate the clock tree advantage, we measured the F_{max} of IRR_WF and HDRA for 32-input configuration in FPGA. We have chosen the IRR_WF and HDRA because they are very competitive in ASIC design. The IRR_WF arbiter had 15% higher F_{max} as compared to HDRA in FPGA design.

V. Conclusion

We have presented a strong fairness round robin arbiter design, IRR. We proved that our design achieve a strong fairness arbitration for all input patterns, which is not guaranteed by some other previous designs such as HDRA [3], PRRA and IPRRA [4]. The key difference of our approach is its operation on the basis of index format of the input ports. This index based arbitration is simple, fast and with small hardware overhead. We did analytical comparison of the arbiters illustrated in Figures 3, 4, 5, 6, 7 and 10 and concluded that the IRR can execute 15% to 50% faster, requires 10% to 47% less area and consumes 1% to 44% less power as compared to the other arbiters. In ASIC and FPGA designs, the IRR arbiter is more economical and faster than other arbiters. Simulation results with the 90nm Synopsys Generic Library show that the IRR saves up to 70% area improvement, up to 74% power improvement, and up to 43% timing improvement over RoR and Matrix arbiters. The distinctive feature of our IRR arbiter design is its lower power consumption as compared to other arbiters due to its usage of fewer registers.

References

[1] W. J. Dally and B. Towles. “Arbitration,” In: Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers, 2004, pp. 349-362.

[2] Z. Fu and Xiang Ling, "The design and implementation of arbiters for Network-on-chips," Proc. 2nd Int. Conf. Industrial and Information Systems, Dalian, 2010 pp. 292-295.

[3] S. Q. Zheng and Mei Yang, "Algorithm-Hardware Codesign of Fast Parallel Round-Robin Arbiters", IEEE Trans Parallel and Distributed Systems, vol. 18, January 2007, pp. 84-95

[4] Yun-Lung Lee, Jer Min Jou, and Yen-Yu Chen, "A High-Speed and Decentralized Arbiter Design for NoC," Proc. IEEE/ACS Int. Conf. on Computer Systems and Applications, Rabat, 2009 pp. 350-353.

TABLE III. HARDWARE CHARACTERISTICS OF STRONG FAIRNESS ROUND ROBIN ARBITERS

Input	Design	ASIC design (90 nm Generic Library) *(2.5ns)*			FPGA design (Stratix V)	
		Total Cell Area *(μm^2)*	Power *(μW)*	Critical path*(ns)*	Comb. Element	Reg. *(bits)*
4	RoR	299	76	0.99	13	4
	IRR	295	53	0.54	9	2
	Matrix	431	80	0.56	16	6
8	RoR	1066	204	0.90	25	8
	IRR	705	105	0.66	37	3
	Matrix	1838	313	0.73	58	28
16	RoR	1846	251	1.46	56	16
	IRR	1390	155	1.24	95	4
		2140[a]	182[a]	0.71[a]		
	Matrix	7817	1242	0.73	238	120
32	RoR	3175	384	2.39	109	32
	IRR	2859	219	1.23	205	5
		4224[a]	351[a]	0.89[a]		
	Matrix	31498	4663	0.92	958	496
IRR/RoR		19% saving	44% saving	43% shorter	28% extra	73% saving
IRR/Matrix		70% saving	75% saving	7% shorter	48% saving	90% saving

[a] the second version of IRR for comparison with the Matrix arbiter

TABLE IV. HARDWARE CHARACTERISTICS OF WEAK FAIRNESS ROUND ROBIN ARBITERS

Input	Design	ASIC design (90 nm Generic Library) *(2.5ns)*			FPGA design (Stratix V)	
		Total Cell Area *(μm^2)*	Power *(μW)*	Critical path delay *(ns)*	Comb. element	Register *(bits)*
4	IRR_WF	278	45	0.48	9	2
	HDRA	352	117	0.6	11	4
	PRRA	344	85	0.79	9	4
	IPRRA	373	74	0.69	9	4
8	IRR_WF	667	100	0.61	37	3
	HDRA	754	172	0.72	28	8
	PRRA	777	158	1.02	24	8
	IPRRA	848	153	0.87	28	8
16	IRR_WF	1478	179	0.77	95	4
	HDRA	1588	249	0.85	62	16
	PRRA	1683	262	1.25	56	16
	IPRRA	1809	267	1.05	73	16
32	IRR_WF	2801	211	1.11	205	5
		3831[b]	315[b]	0.93[b]		
	HDRA	3204	398	0.94	125	32
	PRRA	3466	442	1.48	126	32
	IPRRA	3719	454	1.27	159	32
IRR_WF/HDRA at 4, 8 and 16-inp.		Saving 13%	Saving 44%	Shorter 15%	Extra 12%	Saving 63%
IRR_WF/PRRA		16%	46%	34%	33%	73%
IRR_WF/IPRRA		22%	45%	24%	23%	73%

[b] IRR_WR for comparison with the HDRA arbiter