Round-robin Arbiter Design and Generation

Eung S. Shin, Vincent J. Mooney III and George F. Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, 30332
{eung, mooney, riley}@ece.gatech.edu

ABSTRACT

In this paper, we introduce a Round–robin Arbiter Generator (RAG) tool. The RAG tool can generate a design for a Bus Arbiter (BA). The BA is able to handle the exact number of bus masters for both on-chip and off-chip buses. RAG can also generate a distributed and parallel hierarchical Switch Arbiter (SA). The first contribution of this paper is the automated generation of a round-robin token passing BA to reduce time spent on arbiter design. The generated arbiter is fair, fast, and has a low and predictable worst-case wait time. The second contribution of this paper is the design and integration of a distributed fast arbiter, e.g., for a terabit switch, based on 2x2 and 4x4 switch arbiters (SAs). Using a .25µ TSMC standard cell library from LEDA Systems [10, 14], we show the arbitration time of a 256x256 SA for a terabit switch and demonstrate that the SA generated by RAG meets the time constraint to achieve approximately six terabits of throughput in a typical network switch design. Furthermore, our generated SA performs better than the Ping-Pong Arbiter and Programmable Priority Encoder by a factor of 1.9X and 2.4X, respectively.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Design Aids – *Automatic synthesis, simulation*

General Terms: Design, Experimentation, Performance

Keywords: arbiter, distributed arbiter, round-robin token passing, synthesis, terabit switch

1. INTRODUCTION

As the era of a billion transistors on a single chip fast approaches, more Processing Elements (PEs) can be placed on a System-on-a-Chip (SoC). Most PEs in an SoC communicate with each other via buses and memory. As the number of bus masters increases in a single chip, the importance of fast and powerful arbiters commands more attention. Especially, a fast arbiter is one of the most dominant factors for high performance network switches [5]. Also, fast and efficient switch arbiters are needed to switch packets in a Network-on-Chip (NoC) [1]. However, to design with high performance and fairness in arbitrations is a very tedious and error-prone task for designers.

Fast arbitration schemes are intensively studied in computer networks. A major concern in computer networks today is the design of ultra high speed switches, which provide a high speed and cost-effective contention resolution scheme when multiple packets from different input ports compete for the same output port. This issue is extremely important in order to provide multimedia services for future Broadband Integrated Services Digital Networks (B-ISDN) [2, 15]. We will show how our Round-robin Arbiter Generator (RAG) can help in the design of a terabit switch.

ISSS'02, October 2–4, 2002, Kyoto, Japan.

2. TERMINOLOGY

In this section, we define terms to describe Figure 1. Figure 1 shows the inputs and outputs of the crossbar switch fabric in a 32x32 network switch. Note that we use the terms "switch" and "network switch" interchangeably throughout this paper.

1) An **MxN switch** is an M-input by N-output switch. For example, a 32-input by 32-output device is a **"32x32" device**. Thus, there are 1024 (32^2) different possible connections where a "connection" is between a particular input port and a particular output port.
2) **Virtual Output Queues (VOQs)** are typically employed in a packet switch to mitigate the head-of-line (HOL) block problem. HOL blocking occurs when a single FIFO input queue is used for each input port, and the packet in the head of the queue is blocked from being be forwarded to its corresponding output port due to port contention. By using separate input queues for each input/output port pair, the HOL blocking problem is solved [6].
3) **VOQ (*m, n*)**: *m* is the input port index, and *n* is the output port index. VOQ (*l, m*) implies VOQ at the l^{th} input port destined to m^{th} output port. VOQ (*1, 0*), for example, is the VOQ of input port 1 and queues packets destined to output port 0 as shown in Figure 1.
4) **(MxV)xN**: M is the number of input ports of an MxN switch. V is the number of VOQs per input port, and N is the number of output ports of an MxN switch. Note that the number of VOQs per input port (V) is typically equal to the total number of output ports (N) that can be requested from one input port. The multiplicative product of M multiplied by N is the total number of VOQs in an MxN switch. As the name of Virtual Output Queue (VOQ) implies, an input port considers its V VOQs as output ports. Also, the VOQs dedicated to a certain input port have the same input port index, as shown in Figure 2(a). For example, input port 0 as shown in Figure 2(a) has thirty-two VOQs with the same input port index: from VOQ (0, 0) to VOQ (0, 31). Theoretically, to completely remove the HOL block problem, each input port requires N dedicated VOQs.
5) **(MxV)xN crossbar switch fabric**: There are connections between (MxV) inputs (from VOQ (0, 0) to VOQ (M-1, V-1)) and N outputs, the number of output ports in the switch fabric. As an example, Figure 1 shows a (32x32)x32 crossbar switch fabric.
6) An **MxM Switch Arbiter (SA)** is a part of an (MxV)xN switch with M=V=N; thus, the number of requests (M) equals the number of grants (M). An MxM SA controls M specific transmission gates between M VOQs and a particular output port. At most one transmission gate is turned on at a time. In Figure 1, for example, signal grant (0, 31) from SA_31 turns on or off the transmission

gate between VOQ (0, 31) and output port 31. Signals grant (1, 31) through grant (31, 31) control the other thirty-one transmission gates. The total number of MxM SAs needed for an (MxM)xM switch is equal to the number of output ports, M.

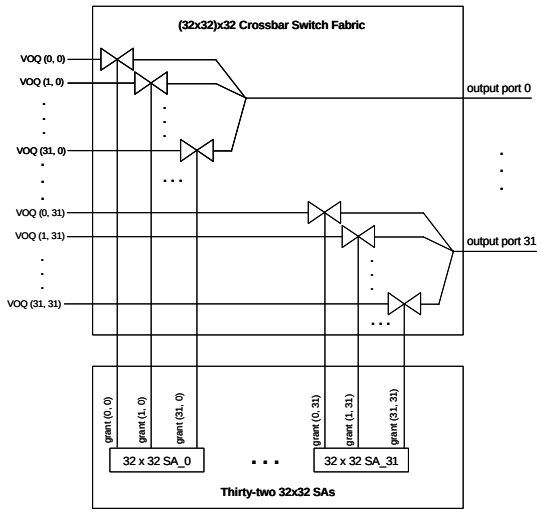

Figure 1. Internal structure of (32x32)x32 crossbar switch fabric and thirty-two 32x32 SAs of 32x32 network switch.

7) An **MxM distributed SA**, equivalently an **MxM hierarchical SA**, plays the same role as an MxM SA. However, an MxM distributed SA is composed of smaller SAs in the form of a hierarchical tree structure.
8) A **root SA** has the same input/output logic function as a regular SA except that there is no request signal output and no acknowledgement ("ack") input. A root SA is used as the "root" SA in the tree structure of an MxM hierarchical SA; this use will become more clear in Section 4.2.
9) A **Bus Arbiter (BA)** resolves bus conflicts when multiple bus masters request a bus in the same cycle. A BA allows access to a bus for the bus master whose request is granted. The input/output logic function of a BA and an SA are the same except that an SA has an extra "request" output. The use of this "request" output will become clear later in Section 4.2. The main difference between a BA and an SA is in typical use: a BA typically arbitrates buses while an SA typically resolves conflicts between input ports and output ports in a switch.

In addition to an (MxV)xN crossbar switch fabric, the internal structure of an MxN network switch consists of VOQs and arbiters (there may be additional hardware components such as memory at the input port in case of the occurrence of VOQ overflows). In Figure 1, we intentionally delete request connections to the 32x32 Switch Arbiters (SAs) from VOQs to present a more compact and easy-to-read diagram. In Figure 2, however, we show request connections to SAs in more detail.

3. RELATED WORK

Current designs in Network-on-Chip (NoC) typically use standard round-robin token passing schemes for bus arbitration [1]. In computer network packet switching, previous research in round-robin algorithms have reported results on an iterative round-robin algorithm (iSLIP) [3] and a dual round-robin matching (DRRM) algorithm [4]. Furthermore, Chao *et al*. describe a design of a round-robin arbiter for a packet switch [5]. Chao *et al*. refer to their hardware design as a Ping Pong Arbiter (PPA). In general, the goal of a switch arbiter in a packet switch is to provide control signals to the crossbar switch fabric as shown in Figure 2(a). In a packet switch design, one must keep in mind that each input port can potentially request connections to all output ports (e.g., in the case of broadcast). Theoretically, to avoid the HOL block problem, in a packet switch with M input ports and N output ports, each input is allocated N VOQs (one per output) for a total of N^2 VOQs in the packet switch. In general, an MxN switch can have fewer VOQs than N to save cost and area at some slight cost of occasional HOL blocking. However, we assume V=N VOQs in this paper.

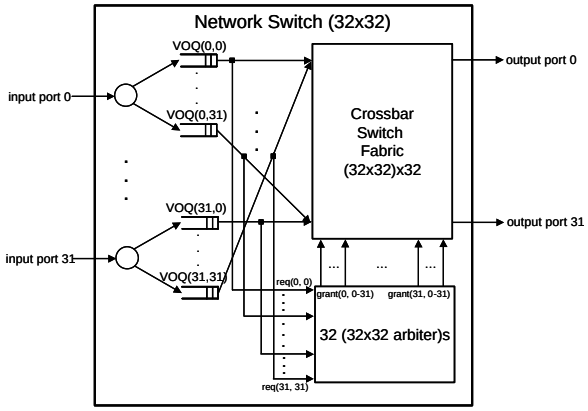

Figure 2(a). 32x32 network switch architecture.

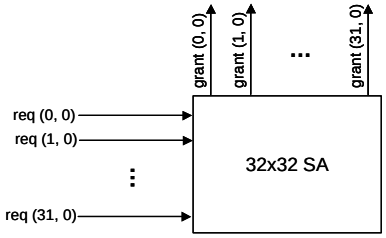

Figure 2(b). 32x32 Switch Arbiter (SA).

Figure 2(a) shows a 32x32 network switch with thirty-two input ports and thirty-two output ports. Each input port can request between zero (none) and thirty-two (all) connections to output ports. To accomplish this, thirty-two 32x32 Switch Arbiters (SAs), shown in the bottom right hand side of Figure 2(a), take as input 32^2 requests (req (0, 0), req (0, 1), …, req (31, 30), req (31, 31) – 32 requests per input port, or one request per VOQ) and translates those requests into 32^2 grant signals (one grant signal per possible VOQ to output connection) where at most one grant signal per output port is set to '1' on each clock cycle (thus, of the 32^2 grant signals, at most 32 are set to '1' each clock cycle).

Figure 2(b) shows one 32x32 SA out of the thirty-two 32x32 SAs. Each SA grants one request out of at most 32 requests from thirty-two VOQs. Each input of the 32x32 SA in Figure 2(b) is connected to a specific VOQ (one per input port) which may request output port 0. The thirty-two outputs of the 32x32 SA are grant signals indicating which of the 32 VOQs is granted output port 0 (note that if no VOQ requests the output port, then all grant signals will be '0' in this case). For example, grant (31, 0) signals the crossbar switch fabric in Figure 2(a) to connect VOQ (31, 0) to output port 0. Since the performance bottleneck of an MxN network switch is the MxM SA [5], we show how our tool can generate a fast and efficient MxM SA.

The iSLIP algorithm uses in its implementation MxM SAs. The iSLIP authors implement an MxM SA in hardware which they call a Programmable Priority Encoder (PPE) [8]. In Section 6,

Experimental Results, we will compare a 128x128 SA generated by RAG to a PPE implementing a 128x128 SA and show a speedup of 2.4X. Similarly, we will compare a 128x128 SA generated by RAG to a 128x128 SA implemented by the PPA hardware described by Chao *et al.* [5], and we will show a speedup of 1.9X over PPA hardware.

4. ROUND-ROBIN ARBITER DESIGN

A round-robin token passing bus or switch arbiter guarantees fairness (no starvation) among masters and allows any unused time slot to be allocated to a master whose round-robin turn is later but who is ready now. A reliable prediction of the worst-case wait time is another advantage of the round-robin protocol. The worst-case wait time is proportional to number of requestors minus one. The protocol of a round-robin token passing bus or switch arbiter works as follows. In each cycle, one of the masters (in round-robin order) has the highest priority (i.e., owns the token) for access to a shared resource. If the token-holding master does not need the resource in this cycle, the master with the next highest priority who sends a request can be granted the resource, and the highest priority master then passes the token to the next master in round-robin order.

Section 4.1 shows the design of the Bus Arbiter (BA) generated by our tool, and Section 4.2 presents a sample design of a 32x32 Switch Arbiter (SA) using our tool. A BA generated by our Round-robin Arbiter Generator (RAG) tool can handle any number of masters, while MxM SAs generated by RAG have a hierarchical structure.

4.1 Bus Arbiter Design

Figure 3 show a BA generated to handle four requests. Figure 3(a) shows the BA block diagram for four bus masters. To generate a BA, RAG takes as input the number of masters and produces synthesizable Verilog code at the RTL level. Figure 3(b) shows the logic diagram for a 4x4 generated by RAG.

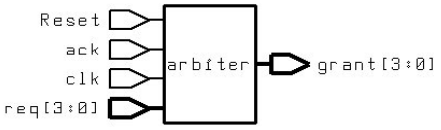

Figure 3(a). Bus arbiter block diagram.

The generated BA consists of a D flip-flop, priority logic blocks, an M-bit ring counter and M M-input OR gates as shown in Figure 3(b) where M=4. A 4x4 priority logic block is implemented in combinational logic implementing the logic function of Table 1. The priority of inputs are placed in descending order from in[0] to in[3] in the priority logic blocks (Priority Logic 0 through 3) shown in Figure 3(b). Thus, in[0] has the highest priority, in[1] has the next priority, and so on. To implement a BA, we employ the token concept from a token ring in a network. The possession of the token allows a priority logic block to be enabled. Since each priority logic block has a different order of inputs (request signals), the priority of request signals varies with the chosen priority logic block. The token is implemented in a 4-bit ring counter as shown in Figure 3(b).

The outputs (four bits) of the ring counter act as the enable signals to the priority logic blocks. Thus, only one enabled priority logic block can assert a grant signal. The ack signal to the bus arbiter is delayed by one arbitration cycle by a D flip-flop as shown in Figure 3(b). The delayed ack signal pulls a trigger to the ring counter so that the content of the ring counter is rotated one bit. Thus, the token bit is rotated left each cycle, with 4'b1000 rotating to 4'b0001 in Figure 3(b), and the token is initialized to one at the reset phase (e.g., 4'b0001 for four-bit ring counter) so that there is only one '1' output by the ring counter. In the round-robin algorithm, each master must wait no longer than (M-1) time slots, the period of time allocated to the chosen master, until the next time it receives the token (i.e., highest priority). The assigned time slot can also be yielded to another master if the owner of the time slot has nothing to send [12]. This protocol guarantees a dynamic priority assignment to bus masters (requestors) without starvation.

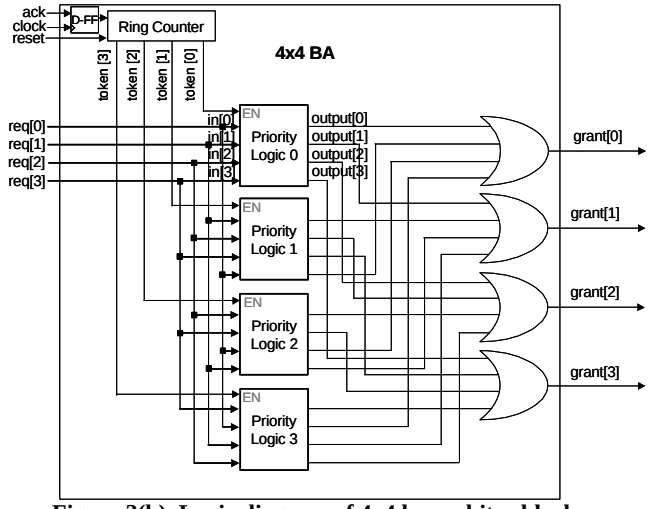

Figure 3(b). Logic diagram of 4x4 bus arbiter block.

Table 1. Truth table of a 4x4 priority logic block.

EN	in [0]	in [1]	in [2]	in [3]	output [0]	output [1]	output [2]	output [3]
0	X	X	X	X	0	0	0	0
1	1	X	X	X	1	0	0	0
1	0	1	X	X	0	1	0	0
1	0	0	1	X	0	0	1	0
1	0	0	0	1	0	0	0	1

Example. Consider a scenario with four processors as bus masters connected to the same bus with one large shared memory on the bus as a slave. Suppose the token is 4 (token=4'b0100, which means processor 2 has the token), and only processor 0 (which uses req[0]) and processor 1 (req[1]) want to access the memory at this cycle. Token=4'b0100 leads to the enabling of only Priority Logic 2 in Figure 3(b). In Priority Logic 2, the connection to in[0] (req[2] from processor 2) indicates the highest priority. Since req[3] is connected to in[1] of Priority Logic 2 in Figure 3(b), processor 3 has the next highest priority. However, since neither processor 2 nor processor 3 make a request, in[2] which is connected to req[0] is next in line in priority. Thus, processor 0 is granted access to the memory, and then the memory controller of the accessed memory sends an ack signal, whose connection to the BA is shown in Figure 3(a), indicating when the memory transaction is successfully completed. Next, which could be several processor clock cycles later, the token is passed to processor 3 (the 4-bit ring counter is rotated when the ack signal is received) in which case the token is 4'b1000. □

4.2 Switch Arbiter Design

The SA generated by RAG uses 2x2 and 4x4 switch arbiter blocks to implement an MxM switch arbiter. RAG is most efficient when M is a power of two. Figures 4(a) and 4(b) show how bus arbiters are modified for switch arbiter implementation by adding some AND and OR gates to a BA. Request signals of the current level are ORed together to generate just one request to the higher level, and grant signals are ANDed together with an ack input (active high) from the higher level so that the only granted switch arbiter block can grant the corresponding master.

A root SA is placed at the top level in the hierarchy. Since there is no higher SA in the hierarchy, root SAs have no ack input nor req output as shown in Figures 4(c) and 4(d). The input/output logic of a 2x2 root SA and a 4x4 root SA are the same as that of a 2x2 BA and a 4x4 BA except that there is no ack input and no D flip-flop in front of the ring counter: thus, the clock input is used to rotate the content (token bits) of the ring counter.

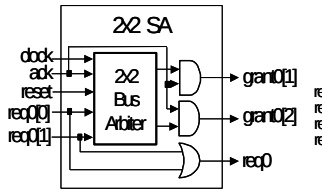

Figure 4(a). 2x2 SA.

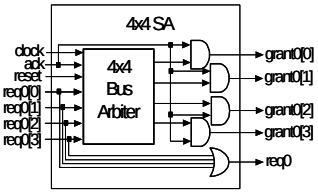

Figure 4(b). 4x4 SA.

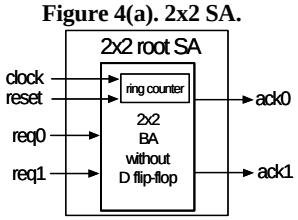

Figure 4(c). 2x2 root SA.

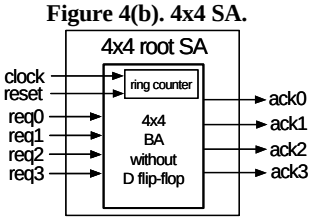

Figure 4(d). 4x4 root SA.

These 2x2 and 4x4 switch arbiter blocks can be composed into a tree structure as shown in Figure 5 (the leftmost blocks are the leaves and the rightmost block is the root). Non-root 2x2 and 4x4 switch arbiter blocks receive an acknowledgement from a switch arbiter block at the next higher level (which translates to being further towards the right hand side of Figure 5) for the next arbitration cycle. Since the root SA in the hierarchy does not receive an acknowledgement (because there is no higher level switch arbiter block), the root arbiter takes the clock input so that a token is passed to the next master in every arbitration cycle in round-robin order.

To reduce the number of levels in the hierarchical Switch Arbiter (SA), we use as many 4x4 switch arbiter blocks as possible because the area of a 4x4 switch arbiter block is less than the area of employing two levels of 2x2 switch arbiter blocks to handle four requests, for a total of three 2x2 switch arbiter blocks: two leaves and one root. Moreover, the delay of a 4x4 switch arbiter block is 0.34ns in a TSMC .25µm library from LEDA Systems [10, 14] which is less than the delay of implementing a hierarchical 4x4 SA using two levels of 2x2 switch arbiters: 0.46ns using the same library. This comparison is discussed in detail in [13].

Figure 5 shows the configuration of 32x32 SA for a 32 x32 fast switch. This hierarchical switch arbiter is a distributed switch arbiter whose individual 2x2 and 4x4 SAs operate in parallel with one another. In other words, the upper level switch arbiters (relatively located at the right side of Figure 5) arbitrate the ORed requests from the lower level switch arbiters in Figure 5, while a internal BA of a lower level SA grants one request regardless of an ack signal from the higher level. A lower level SA ANDs the outputs of its internal BA grant signals with its ack input. The end result is to guarantee that at most one grant out of 32 grants is logic '1'. Note that ack signals from higher levels are also input to a D-flip-flop in the BA internal to each SA in order to potentially rotate the token bit in the next arbitration cycle. More detail about Figure 5, including the critical path, is contained in [13].

This scheme of Figure 5 results in area savings and delay savings compared with a centralized arbiter. Even more, the design of a class of hierarchical SAs similar to Figure 5 is automated by the RAG tool.

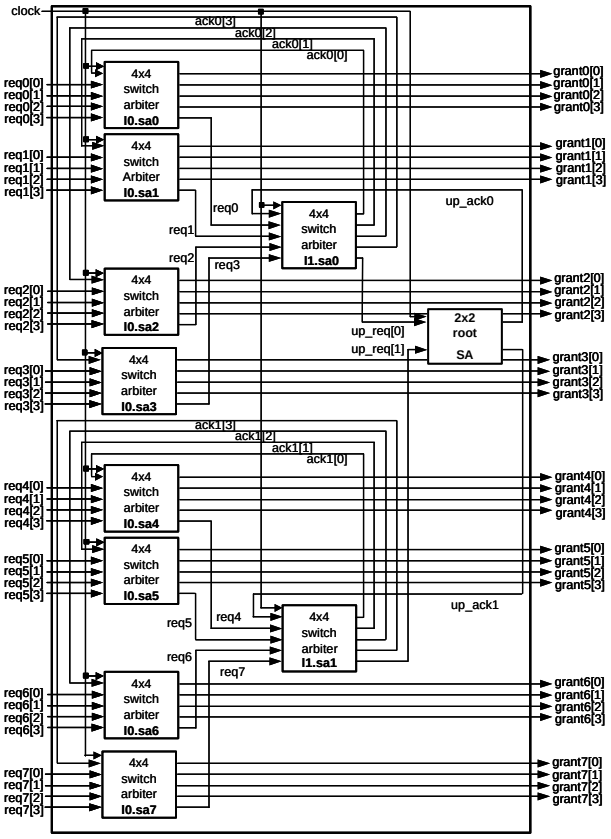

Figure 5. Hierarchical switch arbiter for 32 x 32 switch (Note: reset signal not shown).

5. IMPLEMENTATION OF THE RAG TOOL

Our Round-robin Arbiter Generator (RAG) tool can generate synthesizable Verilog for a BA able to handle any number of requests. RAG can also combine switch arbiter blocks (2x2 and 4x4 switch arbiters) to produce synthesizable Verilog for a hierarchical fast MxM switch arbiter.

5.1 RAG Tool

Figure 6 shows the flow of RAG. First, the user chooses the arbiter type (bus or switch) and the number of masters for the chosen arbiter type. The algorithm for a bus arbiter is straightforward because the tool just generates bus arbiter logic for the exact number of masters. The truth table shown in Table 1 shows the regular pattern as the number of masters, M increases and is implemented with simple logic equation in Verilog for the generation of a BA. From Figures 7 and 8, it turns out that employing a Switch Arbiter (SA) to implement the BA logic is better in terms of area and speed when M is greater than 4 at the possible waste of ports for the case that M is not a power of 2 [13]. In short, a parametrizable Verilog version of Table 1 suffices to generate a BA with any specific number of masters.

For an MxM Switch Arbiter (SA), the tool divides M by 4 to decide the maximum allowable number of 4x4 switch arbiters that can be used in an MxM SA. First, 4x4 switch arbiters are employed as many as the quotient of M divided by 4. If M modulo 4 is not equal to 0, a 2x2 switch arbiter is employed when the remainder is less than 3. Otherwise another 4x4 switch arbiter is employed with one

unused request (one request signal, say req[0] in Figure 4(a), is set to '0'). To get the optimum performance, M is preferred to be a power of two.

After the tool decides the number of 2x2 and 4x4 switch arbiter blocks for each level in the tree structure, the tool will integrate switch arbiters for each level and produce an MxM SA. More algorithm detail, including pseudo code, is available in a technical report [13].

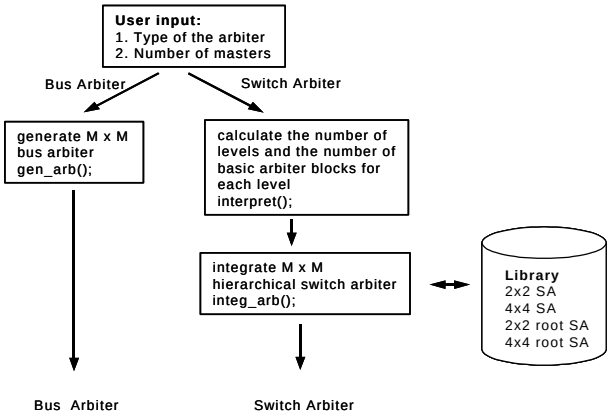

Figure 6. Flow of RAG tool.

5.2 Area and delay considerations

Figures 7(a) and 7(b) show the area and delay, respectively, of BAs generated by RAG. The area increases more than linearly as the number of masters (M) increases. As can be seen in Figure 7(b), delay increases linearly as M increases. Our tool can generate a bus arbiter that can handle the exact number of masters (including non-powers of two). From Figures 7 and 8, for minimal area and delay it is better to employ SA when the number of bus masters is greater than 4 and is a power of two. However, when the number of masters is not a power of two, a user might prefer to choose a bus arbiter option in our tool because of possible waste of ports in the SA. Figure 7 and Figure 8 help a user to choose between BA and SA options with consideration of area and delay.

As shown in Figure 7(b), the increasing delay for our generated BA as M increases will limit the achievable switching speed in a fast switch. We found that limiting the size of the switch arbiter blocks, used as the components of a SA, to 2x2 and 4x4 yielded the fastest switching speeds. Our RAG tool is favored to utilize 4x4 switch arbiter blocks rather than 2x2 switch arbiter blocks. Employing 4x4 switch arbiter blocks gives 16% area saving and 36% gate delay reduction compared with using three 2x2 switch arbiter blocks (two for leaves and one for a root to implement 4x4 switch arbiter block).

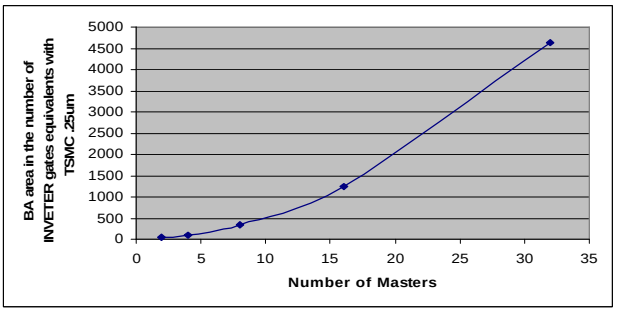

Figure 7(a). Area of MxM bus arbiter

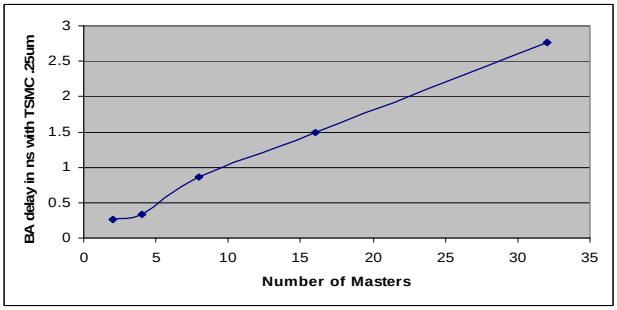

Figure 7(b). Delay of MxM bus arbiter

6. EXPERIMENTAL RESULTS

In this section, we first compare area and delay between a Ping-Pong Arbiter (PPA) [5], a Programmable Priority Encoder (PPE) [8] for iSLIP [3], and our generated Switch Arbiter (SA).

Then will show a speedup for our generated SA over a PPA and a PPE. First, we explain areas and delays of the three switch arbiters, then the speedups achieved by the SA generated by RAG.

6.1 Areas and Delays

PPA uses a 2x2 switch arbiter as a basic switch arbiter block. PPA applies 2x2 switch arbiters to a binary tree structure to form an MxM switch arbiter. Whenever one master is granted by a 2x2 switch arbiter, the other master is guaranteed to be granted for the next cycle. The treatment of upper level grant signals in the binary tree is similar to SA in Figure 4(a). One difference is that every 2x2 switch arbiter of PPA receives acknowledgements from two higher levels and ANDs them together with the current level grants.

A PPE for iSLIP is a centralized switch arbiter. PPE adds a programmable functionality to the priority encoder so that the grant pointer can point to the next request after the current arbitration [8]. The delay and area of priority encoder in PPE becomes larger as M increases and is shown in Figure 8 (a) and Figure 8(b).

In [5], the authors compare PPA with iSLIP [3] and DRRM [4]. Note that DRRM is not implemented in hardware. The performance of PPA is very competitive with speedup c=2. The speedup c of the switch fabric is the ratio of the switch fabric bandwidth and input link bandwidth. Since the arbitration protocol of SA is almost the same as that of PPA, we assume that the performance of a packet switch using SA and a packet switch using PPA are comparable with each other, with the major difference only in logic (arbitration) delay which was found to be the critical path. Thus, we just compare the area and the longest delay of SA with those of PPA and PPE for iSLIP in different MxM configurations.

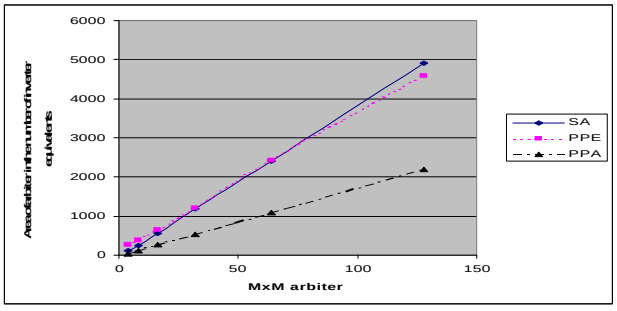

Figure 8(a). The area of MxM switch arbiter.

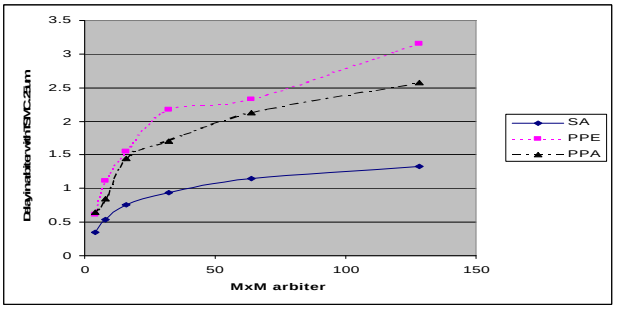

Figure 8(b). The longest delay of MxM switch arbiter

Figures 8 (a) and 8(b) show the area and the longest switch arbiter logic delays of SA, PPE[1] and PPA[2], respectively. The areas of all switch arbiters increase linearly as M increases. SA shows the shortest logic delay when compared with PPE and PPA, and the logic delay increases the slowest (compared with PPE and PPA) as M increases from 32 to 128. This is because we first limit the size of basic switch arbiters to 2x2 and 4x4 to reduce the critical path delay due to the expansion of priority logic blocks as M increases and apply the 2x2 and 4x4 switch arbiters to a distributed structure so that the number and delay of the switch arbiter block(s) in the critical path is minimized. Since PPE is a centralized switch arbiter, its delay is longest. Even though PPA is also a distributed switch arbiter, it has more levels than SA causing more gate delays to connect switch arbiters in different levels.

6.2 Speedup for a Terabit switch

For comparison purposes with PPA and PPE, suppose we have a 128x128 switch with eight-byte cell size, and the speed of the serial link is wholly determined by the arbitration cycles. Thus, the serial link capacity is equal to 64-bit/128x128 switch arbiter delay. Then the aggregated bps capacities of SA, PPA, and PPE are 6.16Tbps, 3.18Tbps, and 2.59Tbps, respectively. This can be verified by using the delays for SA, PPA, and PPE shown in Figure 8(b). Thus, for this comparison, the RAG generated arbiter achieves throughput 1.9X higher than PPA and 2.4X higher than PPE.

Currently some commercial terabit switches are available. One is from Mindspeed, M21155 [9, 16]. M21155 is a 144-port x144-port switch, each port delivering a data rate of up to 3.125Gbps; the aggregate capacity is 0.45Tbps. The other is PetaSwitch [11] from PetaSwitch Solutions, Inc. PetaSwitch claims that their chipset allows configuration of a switch for data rates from Gigabit Ethernet/OC-48 to OC-3072 and port numbers from 2x2 to 256x256. The aggregate bandwidth, PetaSwitch claims, can be configured from 40 Gbps to 10.24Tbps depending on the number of ports and the data rate of port. Unfortunately, no information about the switch arbitration logic nor the process technology (e.g., .25µm) used is publicly available for either of these chips.

[1] The area of PPE is synthesized and complied with Texas Instruments TSC5000 0.25µm technology [8], while SA and PPA with TSMC 0.25µm technology. We modeled the PPE as shown in Figure 11 of [8] to measure the delay with TSMC 0.25 technology. The delay measured here is well matched with Table 2 in [8] except M=32. More details about the comparisons are contained in [13]

[2] We modeled PPA by writing Verilog code based on the logic diagram in [5] and synthesized using the Design Compiler [7] to estimate the area and delay.

7. CONCLUSION

In this paper, we introduced a Round–robin Arbiter Generator (RAG) tool. RAG can generate a BA to handle the exact number of bus masters for both on-chip and off-chip buses. RAG can also generate a parallel hierarchical MxM switch arbiter. We discussed the BA logic and showed the logic of 2x2 and 4x4 SA components. We also presented how RAG uses 2x2 and 4x4 switch arbiter blocks to produce a hierarchical MxM switch arbiter. The first contribution of this paper is the automated generation of a round-robin token passing Bus Arbiter (BA) to reduce bus design time. The generated BA is fair, fast, and has a low and predictable worst-case wait time. The second contribution of this paper is the automated generation of an MxM SA. We compared the area and delay of our generated SAs with PPA and PPE. It turns out that our distributed switch arbiters generated by RAG lead to significant area and delay improvements when compared with other switch arbiters such as PPA and PPE. Specifically, RAG can be used to generate a switch arbiter for a 128x128 terabit switch which, assuming as stated in [5] that the critical path is the switch arbitration logic, achieves throughput 1.9X higher than PPA and 2.4X higher than PPE for the same 128x128 configuration.

8. ACKNOWLEDGMENTS

This research is funded by NSF under INT-9973120, CCR-9984808 and CCR-0082164. We acknowledge donations received from Denali, Hewlett-Packard, Intel, LEDA, Mentor Graphics, Sun, and Synopsys.

9. REFERENCES

[1] W. J. Dally and B. Towels, "Route, Packets, Not Wires: On-Chip Interconnection Networks," *Proceedings of IEEE Design Automation Conference*, 2001, pp. 684-689.

[2] F. A. Tobagi, "Fast Packet Switch Architecture for Broadband Integrated Services Digital Networks," *Proceedings of IEEE*, January 1990, pp. 133-167.

[3] N. Mckeown, P. Varaiya, and J. Warland, "The iSLIP Scheduling Algorithm for Input-Queued Switch," *IEEE Transaction on Networks*, 1999, pp. 188-201.

[4] H. J. Chao and J. S. Park, "Centralized Contention Resolution Schemes for a Larger-capacity Optical ATM Switch," *Proceedings of IEEE ATM Workshop*, 1998, pp. 11-16.

[5] H. J. Chao, C. H. Lam, and X. Guo, "A Fast Arbitration Scheme for Terabit Packet Switches," *Proceedings of IEEE Global Telecommunications Conference*, 1999, pp. 1236-1243.

[6] Y. Tamir and H-C. Chi, "High Performance Multi-queue Buffers for VLSI Communications Switches," *IEEE Transaction on Communications*, 1987, pp. 1347-1356.

[7] Synopsys, Design Compiler, Available HTTP: http://www.synopsys.com/products/logic/design_comp_cs.html.

[8] P. Gupta and N. Mckeown, "Designing and Implementing a Fast Crossbar Scheduler," *IEEE Micro*, 1999, pp. 20-28.

[9] P. Rigby, "Mindspeed unveils terabit switch chip," Network World Fusion Newsletter, 12/12/01, Available HTTP: http:// www.nwfusion.com/newsletters/optical/2001/01142734.html.

[10] TSMC, "IP Services," Available HTTP: http://www.tsmc.com/design/ip.html.

[11] PetaSwitch Product, Available HTTP: http://www.peta-switch.com.

[12] A. Silberschatz, P. Galvin, G. Gagne, *Applied Operation System Concepts*, NY: John Willey and Sons, Inc., 2000.

[13] E. S. Shin, V. J. Mooney III, G. F. Riley, "Round-robin Arbiter Design and Generation," Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-CC-02-38, 2002, Available HTTP: http://www.cc.gatech.edu/tech_reports.

[14] LEDA Systems, Available HTTP: http://www.ledasys.com.

[15] W. Stallings, *Data and Computer Communications*, Fifth Edition, NJ: Prentice Hall, 1997.

[16] Mindspeed, Available HTTP: http://mindspeed.com.