ARBITRATE-AND-MOVE PRIMITIVES FOR HIGH THROUGHPUT ON-CHIP INTERCONNECTION NETWORKS[*]

Aydin O. Balkan, Gang Qu, and Uzi Vishkin

Department of Electrical and Computer Engineering and Institute for Advanced Computer Studies (UMIACS)
University of Maryland, College Park, MD 20742
{balkanay, gangqu, vishkin}@umd.edu

ABSTRACT

An n-leaf pipelined balanced binary tree is used for arbitration of order and movement of data from n input ports to one output port. A novel *arbitrate-and-move* primitive circuit for every node of the tree, which is based on a concept of *reduced synchrony* that benefits from attractive features of both asynchronous and synchronous designs, is presented. The design objective of the pipelined binary tree is to provide a key building block in a high-throughput mesh-of-trees interconnection network for Explicit Multi Threading (XMT) architecture, a recently introduced parallel computation framework. The proposed reduced synchrony circuit was compared with asynchronous and synchronous designs of arbitrate-and-move primitives. Simulations with 0.18μm technology show that compared to an asynchronous design, the proposed reduced synchrony implementation achieves a higher throughput, up to 2 Giga-Requests per second on an 8-leaf binary tree. Our circuit also consumes less power than the synchronous design, and requires less silicon area than both the synchronous and asynchronous designs.

1. INTRODUCTION

In spite of the continuing fast increase in the number of transistors that can fit on a single chip (the *Billion-transistor chip era*), the speed of computer processors has entered a state of diminishing returns in offering growth beyond clock speed. This is happening as scientific and commercial workloads keep expanding at a fast rate.

The outreach of parallel computing has not met some expectations because of parallel computer systems' programmability shortcomings. The Parallel Random Access Model (PRAM) has been developed by numerous algorithm researchers during the 1980s and 1990s, as the key to coping with the programmability challenge. However, it had not been possible to build parallel machines using multi-chip multiprocessors, the only multiprocessors buildable in the 1990s, to support PRAM. The Explicit Multi-Threading (XMT) project ([16], [8]) at the University of Maryland is based on the insight that it is becoming possible to build a parallel processor, which can be programmed with a PRAM-like language on a single chip as silicon capacity continues to increase.

The performance of XMT depends on communication between processor clusters and memory modules. A memory subsystem has been proposed at the logic and physical level for the XMT architecture ([9]). It uses a *mesh of pipelined balanced binary trees* to implement a crossbar type interconnection network. At each node of each binary tree, data buffers and arbiter circuits will be used. Due to the large number of such arbitration circuits in the system, their throughput, speed and power consumption will have significant impact on the memory subsystem's performance. The goal of this paper is to describe a fast and power-efficient implementation of the arbitration circuit at circuitry level. Before elaborating on the difference between our approach and the state-of-the-art arbiters, we mention that our arbiter circuit is not limited to XMT architecture. It can be used to help implementing other on-chip high throughput and low power interconnection networks.

Research on state-of-the-art arbiter circuits were motivated by applications, where the performance of crossbar switches and bus-based networks is directly related to the latency of arbitration (e.g. [4], [10], [13] and [18]). These works focused on n-to-1 arbitration, where a *request vector* ('1' means a request) of n-bits is given, and an n-bit *grant vector* is generated with only one granting signal among n bits.

In cases of crossbar network applications ([4], [13] and [18]), the grant vector is used to configure the switches to connect input ports to output ports. Communication (moving data from input to output port) follows switch configuration. In case of bus network applications ([10]), the grant signal decides which source owns the bus in the next cycle to transmit data. In all of the above approaches, arbitration among n elements precedes the data movement. In contrast, we combine arbitration and data movement. We move the data one level up in the binary tree, towards the root (output port), as soon as we arbitrate between two neighboring requests. Arbitration and movement repeat at each stage until the data reaches the root. Each stage of the pipeline has the latency of a single (2-to-1) *arbitrate-and-move* primitive, which promises great increase in throughput.

The latency of traditional arbiter circuits is measured from the instant the request vector is modified to the instant when the grant vector is updated. N-bit arbiter circuits in each of [10], [13] and [18], were built in a tree structure, using 2-input or 4-input primitives. Critical delay path, which influences latency directly, extends from leaves to the root ([10]) and in some cases, from leaves to the root and then back to leaves ([13], [18]). [4] proposes a two-level design to reduce latency, which may work well for high number of inputs (n>2), however for 2-input arbiters a single level of logic gates would be sufficient. The arbiters, proposed in [4] and [13] are based on *priority encoder* circuits. Priority encoders are simpler and faster, compared to circuits in [10] and [18]; however, they require support of extra circuits to provide fairness among requesting inputs. The above n-to-one arbiter circuits are not as useful for our purpose of implementing the pipelined binary tree. Their underlying complex algorithms are optimized for n>2 inputs implying higher latencies for the primitives, and therefore lower performance when pipelining is used.

The rest of the paper is organized as follows. We state the problem and describe our approach in Section 2. The arbiter circuits are presented in Section 3 and simulation results are reported in Section 4. Section 5 concludes the paper with a discussion of our current and ongoing work.

For an extended version of this paper, see the XMT home page http://www.umiacs.umd.edu/users/vishkin/XMT/.

[*] This work is partially supported by NSF Grant 0325393.

0-7803-8251-X/04/$17.00 ©2004 IEEE

2. PROBLEM STATEMENT

Given is a balanced binary tree with n leaves (as described in Section 1), where n is a power of 2. At each *time unit*, up to *one* request is generated at *each* leaf, which does not already possess a request. The goal is to have the requests reach the root by advancing one step every time unit. Only one request can reside at each node of the tree at any point of time. Once a request exits a leaf, a new request can enter. In the XMT architecture, the root and each leaf in the tree correspond to a processor cluster and/or memory module. To create the full interconnection network between n processor clusters and n memory modules, the tree will be replicated at least n times.

A single request reaches the root in no less than $\log_2 n$ time units. In case of contention, a step may take more than one time unit, and the request reaches the root in at most $n + \log_2 n - 1$ time units. However, the rate of request arrivals at the root will be *one per time unit*, once the first request reaches the root until all the original requests arrive. This rate determines the overall throughput of the binary tree based interconnection network. Our main objective is to design a fast circuit, dubbed ***node circuit***, for each binary tree node to set the shortest time unit and thus increase the interconnection network's throughput.

The leaves generate requests independently. Therefore, when two or more leaves generate requests at the same time they will eventually compete to pass to the same node in the binary tree. Arbitration becomes necessary and an arbiter is required at each node because the competition may occur at any node. The rules for arbitrating and moving a request from child to parent are:

1. Each data generated at a leaf must reach the root.
2. Data generated from the same leaf reach the root in the same order as they are generated.
3. At the time of decision, the parent node will
   a. send a *non-request* signal to its own parent if none of its children has a request to pass forward;
   b. generate a request to its own parent if only one of its children has a request to pass forward, and pass that request;
   c. generate a request to its own parent if both its children have a request to pass forward, and perform the arbitration such that one request is passed at a time and the other request will be passed next.

From these requirements, we see that each node needs to generate signals and sends to its parent, and will receive and process signals from its children. This means that the input and output of a node are of the same type and therefore it becomes possible to build the complete binary tree with identical nodes. This will further increase the scalability of the design. In the rest of the paper, we focus on the design and implementation of a single node. Note that each node also needs to create a signal for the data multiplexer in the mesh of interconnection network. Data path can be implemented as described in [1] and [9].

High throughput is the main objective of our design. It is limited by two constraints: (1) request signals must be received and converted to digital logic values before any arbitration can be made; (2) request signals must be kept at the same state until a proper acknowledgement is received. Propagation delay increases and hence throughput decreases as the number of fan-out increases ([11]). Although buffers can be used to reduce delay, they are not effective for small fan-out circuits, such as ours, where the gates typically have fan-out of less than 4. Transistor resizing is another common technique to improve delay at the expense of increased area. Cascaded application of this technique would cause the node circuits at the leaves to have the largest transistors. Due to the large number of leaves, we limited transistor resizing to within a node circuit.

3. CIRCUITS

Traditionally, arbitrate-and-move circuits in an interconnection network can be implemented asynchronously or synchronously. In this section, we first summarize a fully asynchronous design based on the concept of Micropipelines [15]. We then study the fully synchronous design, which we will not elaborate because it is inferior to other options. We then introduce the *reduced synchrony* (RS) circuits and present two implementations with static CMOS gates (RS-Static) and dynamic TSPC [17] gates (RS-Dynamic) respectively. The novelty of the reduced synchrony circuit lies in the fact that only the root node is connected to external clock and all nodes will selectively propagate the clock pulse to their children to eliminate idle clock switching.

3.1. Asynchronous Arbitrate-and-Move Circuit

Single-input-single-output pipelines built by this approach operate at the speed of a single C-gate ([12], [14]). Arbiter and Call blocks ([15]) are required along with the pipeline segment, to build the node circuit (Figure 1). We implemented Arbiter and Call blocks as described in [6] and [5] respectively. The former generates two mutually exclusive grant signals, which, in turn, are converted to a single request by the latter. Due to space constraints, the reader is referred to [6] for the detailed operation of the arbiter circuit. Careful observation shows that a second request will be served at least 11-gate delays after the first one.

We ran a simulation of a ring oscillator ([11]) circuit to figure out the minimum gate delay for our design technology. 50ps propagation delay of an inverter suggests that our node circuit cannot operate at a rate faster than 550ps between consecutive requests. Detailed simulations show that the delay is much higher than 550ps, due to high fan-out of the gates.

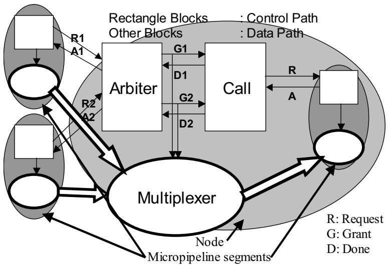

Figure 1: Asynchronous Arbitrate-and-Move Node

3.2. Synchronous Arbitrate-and-Move Circuit

In the synchronous implementation, the nodes will receive the clock signal in addition to sending and receiving request and acknowledgement signals. An acknowledgement signal is required to make the node accept new data or stall, similar to the control structures in instruction pipelines of current processors ([2], p. A-35). As the clock pulse arrives, if there is no stall signal received from the parent, the node can process the requests, and pass one of them. Otherwise, if the parent has sent a stall signal, the current state is preserved, and the stall signal is passed to the children, so that they keep their state as well.

The period of the clock needs to be adjusted according to the worst-case propagation delay of the stall signal from the root to the leaves, since if a node stalls, all other nodes in its subtree have to stall as well, in order not to overwrite other requests. In the worst case the stall signal propagates from the root to all leaves increasing the cycle time by O(log n) gate delays.

3.3. Reduced Synchrony Arbitrate-and-Move Circuits

The ***reduced synchrony*** circuits proposed below share some properties with the synchronous and asynchronous approaches above, yet they do not fit exactly into any of them. We unified acknowledgement signal and clock pulse as a new design approach, to simplify the implementation and conserve power.

These circuits are not connected to a global clock directly. An external *fast-clock* signal is connected to the root circuit only. The children receive clock pulses from their parents, only when needed. There is synchrony between a parent and its immediate children, but there is no global synchrony as in a fully synchronous circuit. Hence, we use the term, *reduced synchrony*.

We initially designed such a circuit using static CMOS gates. Further observations suggested that a design with dynamic gates could yield better performance in terms of speed and power.

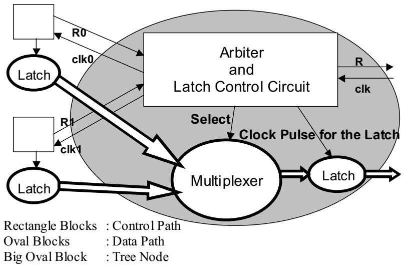

Figure 2: RS-Static Arbitrate-and-Move Node

3.3.1. Static Gate Implementation (RS-Static)

The node is connected to its children as shown in Figure 2. It operates as follows:

- If no requests come from the children, no request signal is sent to the parent, and both children receive a clock pulse.
- If a single request comes from a child, it passes, and the clock pulse is passed to both of the children.
- If two requests come, one passes first and then the other. Only the child with passing request receives a clock pulse.

The implementation can be seen in Figure 3. The critical delay, which determines the clock period of this circuit, is the amount of time between the generation of the clock pulse at the parent node and the update of the request signal at the child node. The former consists of a 2-input *nand* gate and two *inverter*s, and the latter consists of a D-latch, implemented as a *transmission gate* and 2 *inverter*s. Since some of these gates have a fan-out of 3 or 4, we cannot achieve the delay of a gate of the reference ring oscillator as described for the asynchronous circuit in Section 3.1.

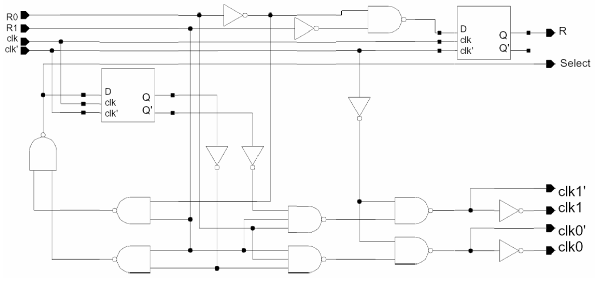

Figure 3: Schematic of RS-Static Arbitrate-and-Move Node Circuit

3.3.2. Dynamic Gate Implementation (RS-Dynamic)

The high-level node structure is the same as in Figure 2 except that the clock signals to the children are unified to *clk-out* signal. Dynamic logic gates from True Single Phase Clocking (TSPC) family ([17]) are used for this implementation (Figure 4). Each gate executes a simple logic function and latches the result for one clock period. This allows the parent node to modify signals of its children. The high-level algorithm is as follows:

- If no requests come from the children, no request signal is sent to the parent, and both children receive a clock pulse.
- If a single request comes from a child, it passes, and the clock pulse is passed to both of the children.
- If two requests come, the one from Child 0 passes, then the parent kills that request. The clock signal is not passed to the children. (At the next cycle, the request at Child 1 remains but Child 0 does not generate a new request)

If both children send a request at a given cycle, the request of child 1 passes as the *only remaining request* at the next cycle. Therefore, arbitration is fair, despite of the built-in priority of Child 0. (Figure 5)

The critical delay path is similar to that of RS-Static, however fewer gates are used: Clock pulse is generated through two half gates (dynamic gates) and a buffer. Request is generated through one dynamic gate.

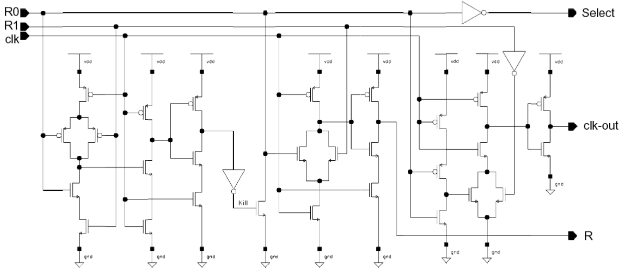

Figure 4: Schematic of RS-Dynamic Arbitrate-and-Move Node Circuit

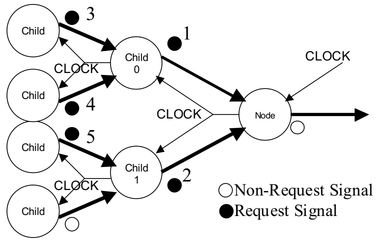

Figure 5: As the Node receives continuous clock pulses, requests pass in order *1, 2, 3, 5, and 4* through the Node.

4. SIMULATION RESULTS

In this section, we describe our simulation setup for the study of the proposed circuits and report our preliminary results. We used Cadence tools (SPECTRE simulator) for all simulations and delay and power measurements. SPICE parameters for 0.18µm technology are obtained from [7]. A 1.8V source is used.

We simulated a single node of the asynchronous arbitrate-and-move circuit with minimum sized transistors, and an 8-leaf-7-node binary tree for both RS-Static and RS-Dynamic circuits. RS–Dynamic (Figure 4) is optimized using Cadence Analog Circuit Optimizer. RS-Static (Figure 3) is optimized only manually, because our preliminary estimations did not suggest that it is promising in outperforming RS–Dynamic. For the same reason and due to space constraint, we will not elaborate on the synchronous arbiter circuit.

As reference, we built a 7-stage ring oscillator using inverters with a W_p/W_n ratio of 13/6 at the same technology. Each inverter showed a 49ps propagation delay (50% of the input to 50% of the output) of full voltage swing) and a 73ps rise time (10% to 90% of full voltage swing). We thus targeted a rise time of 80ps for the clock pulse when we simulate the arbitrate-and-move circuits.

Table 1 reports the performance, measured by speed, area, and power, of different circuit implementations for a test design explained above. The root node of RS circuits is connected to a clock generator with various clock periods for our simulation. Speed is measured as the fastest clock (or equivalently, the shortest clock period) that the circuit can keep pace with for the RS circuits, and the time interval between the request and done signals for the asynchronous circuit. Area is measured as the total transistor area (width x length) and the number of transistors required per node circuit. Power consumption of each circuit is measured as the average power consumed by the global power source on the same request pattern.

The shortest clock period for the RS-static circuit to work correctly is 800ps, while the RS-dynamic circuit operates correctly at a clock period as fast as 500ps. (The critical delay path of RS-dynamic circuit is even shorter, but we observed that the clock signal may not be able to complete full voltage swing occasionally). The asynchronous circuit completes one arbitration cycle in 2.5ns. No other requests can be processed during this time. The synchronous circuit was not measured as it is clearly dominated by the proposed reduced synchrony circuits.

Node Circuit	Speed	Transistor Area	No. of transistors	Power consumption
RS-Dynamic	500 ps	28.4 μm^2	34	16.4 mW
RS-Static	800 ps	38.4 μm^2	72	8.8 mW
Asynchronous	2.5 ns	$\geq 35\mu m^2$	188	N/A
Synchronous	N/A	N/A	~90 (est.)	N/A

Table 1: Performance comparison of the four different arbitrate-and-move circuit implementations.

5. DISCUSSION

All circuits described above contain state holding elements. In the asynchronous one, these elements may go into a metastable state as the node receives two request signals concurrently. Extra circuitry, which reduces the speed of the circuit, is required to prevent this from happening. In the synchronous circuit, the inputs of the state holding elements have to be ready within the same clock period and a stalling condition at the root may delay inputs at the leaf circuits. Both of these properties cause degradation of throughput in an overall system.

Power consumption is proportional to switching frequency, and load capacitance. Although we have not implemented the synchronous arbiter circuit, we expect that it will consume more power because of the clock network (clock generator and the clock tree) and the fact that all clocked gates will operate at the highest frequency. On the other hand, in the RS circuits, a node sends clock pulse to its children only when it is ready for a new request. Therefore the inner nodes will not always receive the fastest clock signal. In the case when the interconnect tree is fully loaded with requests, there will be only one single branch driven by the clock, from the root towards the leaves, and the remaining part of the tree will not consume any power. Table 1 shows that RS–Static is slower but consumes less power compared to the RS-Dynamic circuit. A more realistic result could be observed if we would let the input pattern change over time, as will be the case in the XMT implementation. Future research on XMT will need to examine load characteristic on the interconnection network.

In sum, we conclude that the dynamic gate implementation of reduced synchrony circuit best suits our design objectives, while the static gate implementation is also useful when compared with traditional asynchronous or synchronous designs. As for future work, we first mention that opportunities for less power consumption still exist; for example, an extra circuit to avoid unnecessary precharging of the dynamic gates. Also, note that the total silicon area of the circuit should also include diffusion areas, overhead areas, pins and metal wires among others. At the current stage of this project, we are unable to determine these values without the silicon layout. However, we expect that the circuit with larger transistor area will have larger total silicon area. Finally, we mention that, when we compare area and power consumption with the synchronous design, any clock driver required by the synchronous arbiter must also be included for a fair comparison, because the clock circuit of the reduced synchrony arbiter circuits is built into the nodes.

6. CONCLUSION

We reviewed *asynchronous* and *synchronous* arbiter circuits, and proposed *reduced synchrony* arbiter circuits. Simulations show that a binary tree shaped arbitration network can be implemented by our circuit in a fast (2 Giga-Requests/second at 0.18μm) and power effective way. The modularity of our approach, where the input characteristics of the primitive circuit match its output characteristics, provides design scalability.

We believe that future work on XMT could greatly benefit from the throughput provided by this circuit. Furthermore the proposed an arbitration scheme may prove useful in areas where performance is held back due to high latency of present arbitration and scheduling schemes, such as issue-logic of wide-issue superscalar processors [3].

7. REFERENCES

[1] S. Dutta, “VLSI Issues and Architectural Trade-Offs in Advanced Video Signal Processors”, *Ph.D. Thesis*, Princeton University, 1996

[2] J.L. Hennessy, D.A. Patterson, *Computer Architecture A Quantitative Approach*, 3rd ed., Morgan Kaufmann Pub., 2003

[3] D.S. Henry et al. “Circuits for wide-window superscalar processors”, *Proc. 27th International Symposium on Computer Architecture*, IEEE, pp. 236-247, 2000

[4] C.K. Hung, M. Hamdi, C. Tsui, “Design and Implementation of High-Speed Arbiter for Large Scale VOQ Crossbar Switches”, *Proc. of Int. Symposium on Circuits and Systems (ISCAS)*, vol. 2, pp. 308-311, 2003

[5] L. Lloyd et al., “Asynchronous Microprocessors: From High Level Model to FPGA Implementation”, Department of Computing Science, University of Newcastle upon Tyne, 1997

[6] C.E. Molnar and I.W. Jones, “Simple Circuits that Work for Complicated Reasons”, *Int. Symposium on Advanced Research in Asynchronous Circuits and Systems*, Eilat, Israel, April 2000

[7] MOSIS web site, http://mosis.org/cgi-bin/params/tsmc-018/t29b_mm_non_epi-params.txt

[8] D. Naishlos, J. Nuzman, C. Tseng and U. Vishkin, “Towards a First Vertical Prototyping of an Extremely Fine-Grained Parallel Programming Approach”, *Theory of Computer Systems*, Online First Edition, 2003 (special issue for SPAA’01)

[9] J. Nuzman, and U. Vishkin, “Memory Subsystem Design for Explicit Multithreading Architectures”, in preparation, also J. Nuzman’s MS Thesis, Univ. of Maryland, 10/2003.

[10] F. Petrot, D. Hommais, “A Generic Programmable Arbiter with Default Master Grant”, *Proc. of Int. Symposium on Circuits and Systems (ISCAS)*, vol. 5, pp 749-752, Geneva, 2000

[11] J.M. Rabaey, *Digital Integrated Circuits*, Prentice Hall Inc., 1996

[12] M. Shams, J.C. Ebergen, and M.I. Elmasry, “A Comparison of CMOS Implementations of an Asynchronous Circuits Primitive: the C–Element”, *Proc. Int. Symposium on Low Power Electronics Design*, pp 93-96, 1996

[13] E.S. Shin, V.J. Mooney III, G.F. Riley “Round-Robin Arbiter Design and Generation”, *15th Int. Symposium on System Synthesis*, pp 243-248, 2002

[14] J. Sparsø and S. Furber, *Principles of Asynchronous Design*, Kluwer Academic Publishers, 2001

[15] I. Sutherland, “Micropipelines”, *Communications of the ACM*, June 1989

[16] U. Vishkin, S. Dascal, E. Berkovich and J. Nuzman, “Explicit Multi-Threading (XMT) Bridging Models for Instruction Parallelism (Extended Abstract*)”, Proc. 10th ACM Sym. on Parallel Algorithms and Architectures (SPAA)*, 1998

[17] J. Yuan and C. Svensson, “New Single-Clock CMOS Latches and Flipflops with Improved Speed and Power Savings”, *IEEE Journal of Solid-State Circuits*, Jan 1997

[18] S.Q. Zheng et al. “A Simple and Fast Parallel Round-Robin Arbiter for High-speed Switch Control and Scheduling”, *45th Midwest Sym. on Circuits and Systems*, vol. 2, pp 671-674, 2002