justify why the broad-side method should be the primary scan-based delay test strategy. It should not, however, be dismissed completely, because it might have some incremental value when used in a hybrid delay test scheme.

There are several problems with the broad-side delay test method. The primary problem is its limited capability to generate a rich set of two-pattern tests, especially in single-section networks. What aggravates the situation even further is that generally there is nothing that can be done in terms of design for testability, to improve its delay test performance. When compared to skewed-load, this is a major disadvantage. The performance of skewed-load can be substantially improved by performing latch re-ordering, or even made to be ideal by performing a complete input separation [12]. The input separation method re-assigns the combinational circuit inputs to scan latches, so that no two inputs belonging to the same output cone emanate from adjacent latches on the scan chain. This "separation" of inputs completely breaks the shift dependency associated with the skewed-load method, and allows its transition fault coverage to assume the maximum possible value (equal to the stuck-fault coverage). Input separation, however, does not come without a cost (extra latches and performance degradation due to possible longer wires), and this is where the incremental value of the broad-side method may come into play. For cases were latch re-ordering is infeasible, a hybrid test methodology, that uses skewed-load as the primary mode, and broad-side as a secondary mode, is a viable alternative.

There is also a secondary problem associated with the simulation of broad-side patterns. Since the second vector of the transition test is generated through the logic, the simulation is generally a two-time-frame event. This event slows down the process, and is quite profound in simulation of large structures.

Still another problem with broad-side, when compared to skewed-load, is its unbounded (from below) fault coverage. The minimum transition fault coverage attainable by skewed-load is conjectured to be 50%. Unfortunately, there is no such minimum attainable by broad-side. It is quite easy to come up with examples were the broad-side transition fault coverage is below 50% (see Fig. 3).

This paper has studied the problem of broad-side delay test generation. It has provided a Boolean-difference-based approach to calculate broad-side delay test vectors. It has shown how to modify (in concept) existing test generation tools so that they be able to generate broad-side delay test vectors. As a final chapter, it has shown the results of an extensive experiment, conducted on the ISCAS sequential benchmark circuits, to assess the capability of the broad-side method, and other hybrid methods that involve broad-side and skewed-load.

References

[1] S. B. Akers, "On a theory of boolean functions," *J. Soc. Industrial Math*, vol. 7, no. 4, pp. 487–497, 1959.

[2] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*. New York: Wiley Intersci., 1987.

[3] Z. Barzilai and B. Rosen, "Comparison of AC self-testing procedures," in *Proc. Int. Test Conf.*, Oct. 1983, pp. 89–94.

[4] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. Int. Symp. Circ. and Syst.*, May 1989, pp. 1929–1934.

[5] A. C. L. Chiang, I. S. Reed, and A. V. Banes, "Path sensitization, partial boolean difference, and automated fault diagnosis," *IEEE Trans. Comput.*, pp. 189–195, Feb. 1972.

[6] H. Fujiwara, *Logic Testing and Design for Testability*. Cambridge, MA: MIT, 1985.

[7] V. S. Iyengar, B. K. Rosen, and J. A. Waicukauski, "On computing the size of detected delay faults," *IEEE Trans. Computers-Aided Design*, pp. 229–312, March 1990.

[8] S. Patil and J. Savir, "Skewed-load transition test: Part II, coverage," in *Proc. Int. Test Conf.*, Sept. 1992, pp. 714–722.

[9] S. M. Reddy, C. J. Lin, and S. Patil, "An automatic test pattern generator for the detection of path delay faults," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1987, pp. 284–287.

[10] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Develop.*, vol. 10, pp. 278–291, July 1966.

[11] J. Savir, "Skewed-load transition test: Part I, calculus," in *Proc. Int. Test Conf.*, Sept. 1992, pp. 705–713.

[12] J. Savir and R. Berry, "At-speed test is not necessarily an AC test," in *Proc. Int. Test Conf.*, Oct. 1991, pp. 722–728.

[13] J. Savir and W. H. McAnney, "Random pattern testability of delay faults," *IEEE Trans. Comput.*, vol. 37, pp. 291–300, March 1988.

[14] G. L. Smith, "Model for delay faults based upon paths," in *Proc. Int. Test Conf.*, Nov. 1985, pp. 342–349.

A Low Latency Asynchronous Arbitration Circuit

Alexandre Yakovlev, Alexei Petrov, and Luciano Lavagno

***Abstract*—We present an asynchronous circuit for an arbiter cell that can be used to construct cascaded multiway arbitration circuits. The circuit is completely speed-independent. It has a short response delay at the input request-grant handshake link due to both a) the propagation of requests in parallel with starting arbitration and b) the concurrent resetting of request-grant handshakes in different cascades of a request-grant propagation chain.**

***Index Terms*—Arbiters, asynchronous circuits, conflict resolution, modular control logic, resource allocation, signal transition graphs**

I. Introduction

Arbitration circuits are commonplace in digital systems with a single resource allocated to different user processes. The typical examples are systems with shared busses, multiport memories, packet routers to name but a few. An asynchronous arbiter is defined as a circuit that dynamically allocates a single shared resource to the user components in a system which is free from common clock. Each user, when it requires the resource, issues an asynchronous request and waits until the arbiter produces a grant. The user then uses the resource and after finishing its action releases its request. This results in a subsequent release of the grant, after which the user can issue another request and so on.

The arbiter, when it receives a number of active requests from different users, generates after some finite delay a grant to exactly one of them and leaves other requests pending until the granted user

Manuscript received July 7, 1993; revised December 17, 1993 and February 15, 1994. The work of A. Yakovlev was supported by an SERC Grant GR/J52327. The work of A. Petrov was supported by the Royal Society of the UK. The work of L. Lavagno was supported by MURST under 40% project "VLSI Architecture".

A. Yakovlev is with the Department of Computing Science, University of Newcastle upon Tyne NE1 7RU, England.

A. Petrov is with the Department of Computing Science, St. Petersburg Electrical Engineering University, St. Petersburg 197022, Russia.

L. Lavagno is with the Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy.

IEEE Log Number 9403171.

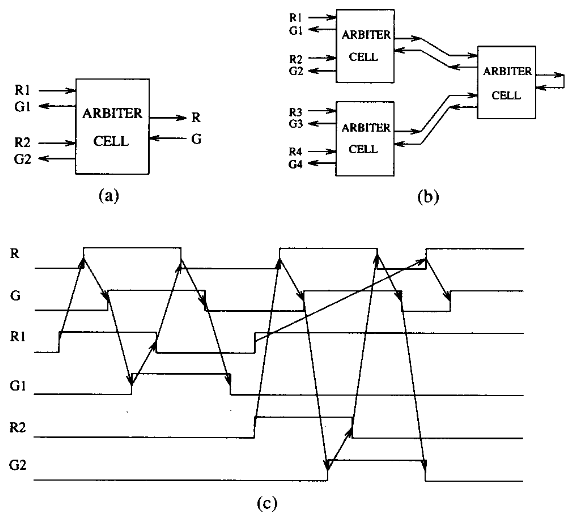

Fig. 1. A two-input arbiter cell.

has released the request. The arbiter then releases the grant and, if there are pending requests, produces another active grant, again on a mutually exclusive basis.

We consider a standard basic cell of a multiway arbiter that arbitrates between two users. Multiway arbitration is organised by building a cascade of such cells to form a tree or a linear structure. Each cell thus propagates the request in the direction from the lower level to the higher level of the structure, while the grants are generated in the opposite direction. Fig. 1(a) shows one such cell, a two-way arbiter, with its three request-grant handshake links $(R1, G1)$, $(R2, G2)$ and (R, G), where $(R1, G1)$ and $(R2, G2)$ stand for the links with lower level cascades, generating competing requests at $R1$ and $R2$, and the (R, G) pair is the link with the higher level cascade. An example of a 4-way arbiter, shown in Fig. 1(b), illustrates the regular way in which a cascaded multiway arbiter can be composed from the basic cells. Fig. 1(c) illustrates the handshaking protocol between the links. After a first request by $R1$ is granted and the resource is released, two simultaneous requests are made by $R1$ and $R2$ and granted in turn.

II. Background

Asynchronous arbiters of the above type have been studied in literature extensively. The early attempts to build an arbitration circuit from logical gates were unsuccessful due to the metastability and oscillation anomalies occurring in a basic mutual exclusion (ME) element that the arbiter has to employ [9], [8]. An example of a two-way arbiter with an analogue ME element[1] was published in [10]. Other examples, together with a rigorous proof of the impossibility of a correct implementation of an arbiter with logical gates, were presented in [12]. Despite functioning correctly these arbiters suffer from two problems:

1) they wait for the arbitration to be resolved by the local ME before propagating the request (signal R) to the high-level cascade, and
2) they wait for the complete release of the grant (signal G) from the higher level cascade before releasing its own grant signal ($G1$ or $G2$).

[1] i.e., where mutual exclusion was achieved at the transistor rather than gate interconnection level.

Both these problems were first tackled by Pearce, Field, and Little in [8] but the proposed solution may malfunction if the mutual exclusion resolution takes longer than the delay with which the grant arrives from the higher cascade. Their circuit is thus not speed-independent because its operation strictly relies on gate delay ratios.

Recently, an elegant solution for the first problem has been suggested by Josephs *et al.* [3], [14] with an analogue ME element. Their circuit uses an OR-functionality to produce the request on R from the arrival of $R1$ or $R2$, without waiting for the result of mutual exclusion from the ME. However, this solution still does not completely reduce latency in the request-grant propagation chain by leaving the second problem unresolved.

On the other hand, an attempt to solve only the second problem using a similar ME element has been made in [2]. The published circuit (Fig. 11(a) in [2]) is however not strictly speed-independent because the delays at the outputs of two gates (denoted by *ha*3 and *ha*4 in [2]) must be severely bounded. If the latter are assumed to be arbitrary, the circuit may malfunction.

The circuit which is presented in the following section solves both of these problems, and works well with respect to any delays associated with the outputs of the gates. Furthermore, it is insensitive to the delays in the wires between the cells. Some internal wires must however have bounded delays, but this is quite a standard restriction for most practically useful asynchronous circuits (see, e.g., the idea of isochronic forks in [5]). The requirement of speed-independence and delay-insensitivity with respect to the wires between cells is quite important if the arbiter is constructed in a distributed system, where the system's components contain only one or several arbitration cells (e.g., [5]). Another advantage of this circuit is that it has been obtained by using an automatic synthesis approach. First, the behavior of the arbiter, as solving the above problems, is precisely defined in a formally based notation, which has an explicit notion of causality and concurrency. Second, a software tool has been used in order to produce a correct speed-independent circuit.

III. Proposed Solution

A. Signal Transition Graphs

In order to precisely describe the required functionality of the arbiter cell, we use a graph-based formalism, called Signal Transition Graph (STG). Avoiding formal definitions and properties of STG's, as the purpose of using this model here is purely illustrative, we refer the reader to [13]. Informally, an STG is a Petri net [7] whose transitions are labelled with the changes of binary signals of the described circuit. An example of the STG which describes the behavior of a simple two-input synchroniser (C-element of Muller [6]) is shown in Fig. 2(a). Fig. 2(b) represents a timing diagram-like description of the same behavior.

Here X and Y are input signals, and Z is the output of the circuit. The circuit changes its current output value (say, 0), when both its inputs transition from the previous value (0) to the new value (1). A similar synchronisation takes place when the circuit's output is originally at logical 1.

The rules of action of the STG are the same as those of Petri nets. Every time, all input places (circles in the graph) of some transition (a bar in the graph) contain at least one token (bold dot in the circle) each, the transition is assumed to be enabled and can fire. The firing of an enabled transition results in a change of the net marking (the current assignment of tokens to the places). Exactly one token is removed from each input place of the transition and exactly one token is added to each output place of the transition. The firing action is

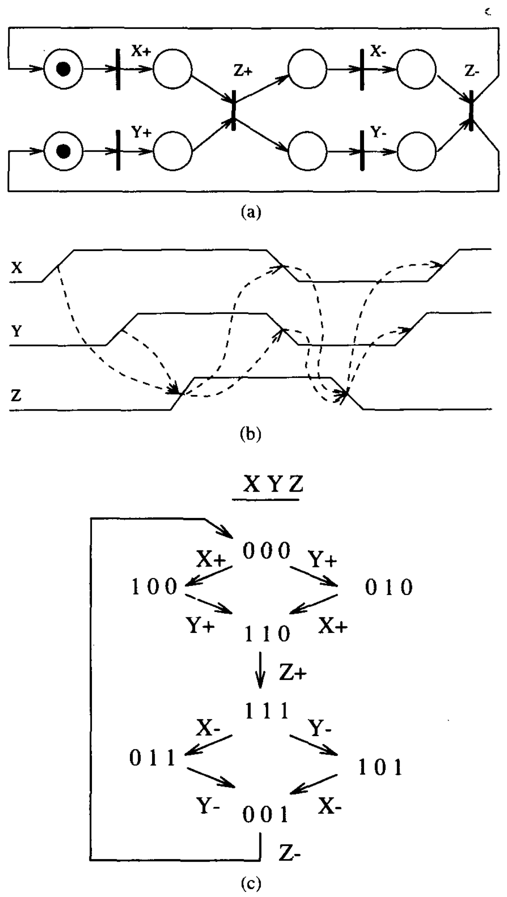

Fig. 2. A timing diagram and signal transition graph example.

assumed to be atomic, i.e., it occurs as a single, indivisible event taking no time. An unbounded finite non-negative amount of time can elapse between successive firings.

The operation of the STG can be traced in the *state graph* which is a graph of reachable markings obtained through the potential firings of the enabled transitions (two or more transitions enabled in the same marking can produce alternative sequences of signal changes). The state graph for the STG in Fig. 2(a) is shown in Fig. 2(c). The vertices in this graph are the markings labelled with a vector of values of the signals, whose changes label the transitions in the original STG.

For example, the initial marking of the STG (dotted places) corresponds to the top state of the state graph. In that marking, all signals have just had a falling transition, so the state can be labeled by the vector 000 (with signal ordering XYZ). Two transitions, X^+ and Y^+, are concurrently enabled in that marking. Assuming X^+ fires first, its successor place becomes marked, thus reaching the state labeled 100. Then Y^+ fires (Z^+ cannot fire yet, because only one predecessor place is marked), thus reaching the state labeled 110. Now Z^+ can fire, because both its predecessors are marked, and so on. The complete state graph is constructed by exhaustive firing of the STG transitions, resolving concurrency in all possible ways (e.g., by firing Y^+ before X^+ as well).

The state graph generated by an STG specification can be used to derive the logical implementation of the circuit in the form of the functions of logical gates. This however requires some conditions to be satisfied both at the STG and the state graph levels:

- the transitions labeled with the same signal interleave in their signs ("+" and "−") for any firing sequence, and
- all the states that are labeled with the same vector have the same set of enabled non-input signal transitions.

The former condition guarantees that a vector label consistent with the firing can be assigned to each state. The latter condition ensures that each noninput signal can be implemented with a *combinational* logical circuit computing its *implied value*. The implied value of a signal in a marking is the same as the value in the marking label if no transition for that signal is enabled in the marking; the complement otherwise.

For example, the implied value of output Z in the initial state is 0, because no transition of Z is enabled in that state. Its implied value in the state labeled 110 is 1, the complement of the value of Z in the label, because transition Z^+ is enabled. Intuitively, the implied value is the value each output signal "tends to" in each state, according to the enabled transitions. The truth table describing the implied value Z' of signal Z is as follows:

X	Y	Z	Z'
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
1	1	1	1
0	1	1	1
1	0	1	1
0	0	1	0

From it we can derive, using standard truth table minimization techniques, a Boolean expression describing a gate-level implementation of a C element:

$$Z' = XY + XZ + YZ$$

B. Arbiter Cell Model and Implementation

The behavior of the arbiter cell solving both the above mentioned problems is defined by the STG shown in Fig. 3. Places with a single predecessor and successor, e.g., between $R1+$ and $A1+$, are omitted. This STG clearly demonstrates that after the arrival of either $R1$ or $R2$ the handshake R/G is set in parallel with resolving mutual exclusion at the outputs $A1$ and $A2$. This solves the first of the two problems outlined in Section II. The second problem is solved by beginning the resetting of $G1$ or $G2$ immediately after the R output has been reset, thus making the release of the R/G handshake in parallel with the release of the grant and a potential new setting of the request signal in the link ($R1, G1$) (or ($R2, G2$)). The latter makes possible the propagation of the new request "setting wave" through the cascades directly after the "releasing wave". Note that it is possible that two (or more) users see a Grant signal high at the same time, due to this "pipelining" of the Grant reset phases. Nevertheless, the overall protocol ensures correct mutual exclusion from the shared resource, because only one of these users is also *requesting* the resource, while all others have finished using it and are waiting for the handshake closing.

This STG was implemented and verified by automatic asynchronous circuit synthesis tools [11], [4]. The circuit, shown in Fig. 5, is speed-independent with respect to delays in the gates and the wires between the arbiter cells. i.e., it operates correctly without hazards no matter what the values of those delays are ([6]). Note that correctness is not guaranteed independent of the delays of the wires *inside* each cell. The initial state of each signal in the circuit

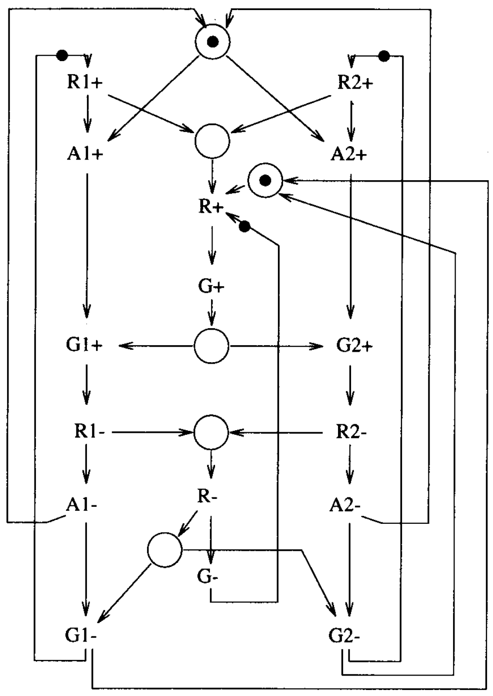

Fig. 3. The arbiter signal transition graph specification.

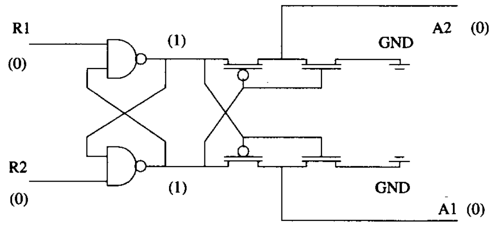

Fig. 4. A mutual exclusion element (from [10]).

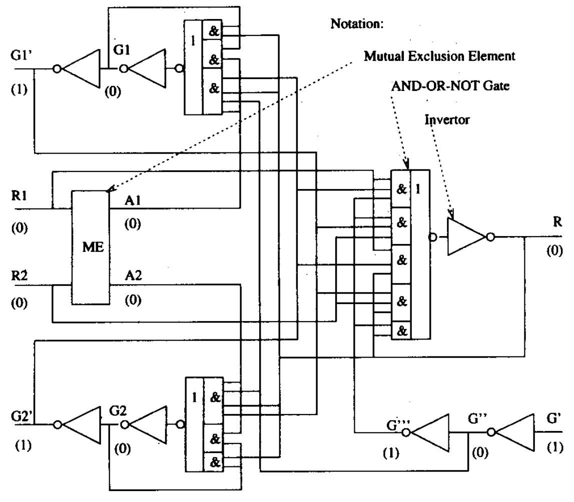

Fig. 5. The arbiter circuit implementation.

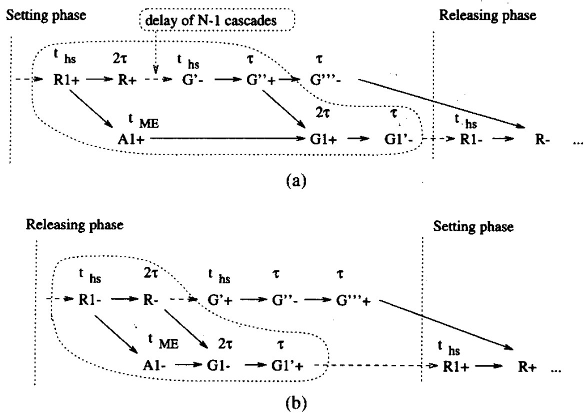

Fig. 6. Event charts for performance evaluation.

is given in brackets. This circuit consists of an ME element[2], three latches built on AND-OR-NOT gates, and a number of auxiliary invertors. The circuit implementation uses the *complement* of G, $G1$ and $G2$ (denoted by G', $G1'$ and $G2'$, respectively) in the handshakes between cells in order to ensure the absence of hazards.

C. Performance Evaluation and Comparison with Previous Work

In order to compare the performance of this circuit with the previously known arbiter cells, we should consider its action within the context of the cascaded arbiter. Let us assume that the given circuit is used as a cell in the Nth cascade of the N-level arbiter.

The performance of the circuit can then be evaluated by determining the length of the cycle between two successive settings of, say, $R1$. For this we shall use the partial orders on the sets of switching events that happen between $R1+$ and $G1'-$ (this interval being called the setting phase) and between $R1-$ and $G1'+$ (the releasing phase) shown in Fig. 6(a) and (b), respectively. These event charts correspond to the case when the request arriving on $R1$ gets the grant.

[2]Its "behavior" is defined by the signals $R1$, $R2$, $A1$ and $A2$ in the STG and an example CMOS implementation can be found in Fig. 4 (derived from [10]).

Similar event charts could be constructed for the case when $R2$ is successful. The solid arrows in these partial order representations correspond to the direct (causal) precedence relation between signal transitions. For example, the rising edge on the input $R1$ causes the switching of the AND-OR-NOT gate (its upper AND term becomes equal to 1) and the subsequent invertor, thus making output R set to 1. The delay between $R1+$ and $R+$ can be estimated as the sum of delays of the AND-OR-NOT gate and the invertor. On the other hand $R1-$ causes the 0–1 transition on $A1$ to occur after some delay, which can be called the delay of the ME element. All other solid arrows can be interpreted in similar way. The dashed arrows correspond to the switching actions on the request-grant handshakes. For example, the arrow between $R+$ and $G'-$ shows that the rising transition on R will eventually cause the arrival of the falling transition on G'. The associated delay is attributed to the setting phase delay of the remaining, $N-1$, cascades of the arbiter plus the delay of the interconnection between the given, Nth, and the next, $(N-1)$th, cascades.

With each signal transition in the setting and releasing phases we associate a delay, which is shown above the actual transition, according to the corresponding delay in the circuit. Thus, it is assumed that each AND-OR-NOT or invertor introduces one gate unit delay τ

(note that this assumption is introduced for *performance evaluation* only; the correctness of the circuit behavior is independent of the gate delays). With the transitions of $A1$ we associate $t_{\mathtt{ME}}$, which is the average delay introduced by the ME in the setting (arbitration resolution) phase. In the releasing phase, where there are no potential metastability delays, the delay introduced by the ME can be safely assumed equal to that of two gates, i.e., 2τ. Every time a switching process passes through a handshake, we add a delay t_{hs} associated with the transition of the input signal of the corresponding handshake pair. Thus, the transitions on $R1$ and G' are marked with t_{hs}. We therefore assume for simplicity that t_{hs} is the average cumulative delay of the handshake interconnections, including also, if necessary, the delays of the line transceivers.

1) With the aid of Fig. 6(a) one can easily derive the delay of the setting phase:

$$D_N^{\mathrm{set}} = \max((3\tau + 2t_{\mathrm{hs}} + D_{N-1}^{\mathrm{set}}), t_{\mathtt{ME}}) + 3\tau,$$

where D_{N-1}^{set} denotes the setting phase delay introduced by the subsequent $N-1$ cascades (we assume that $N > 1$). The last (top level) cascade can be optimised to become a single ME element (we do not need any extra logic), whose signals $A1$ and $A2$ can be used directly as $G1$ and $G2$. Furthermore, as one may easily notice, the invertor between G' and G'' in the cascade next to the top level can be deleted. The grant signals between these two cascades are transmitted in direct form (without inversion).

If we assume that the average value of $t_{\mathtt{ME}}$ is such that $3\tau + 2t_{\mathrm{hs}} + D_{N-1}^{\mathrm{set}} > t_{\mathtt{ME}}$, then bearing in mind that D_{N-1}^{set} can be recursively expressed in the same form, we can easily write down the setting phase delay of the cascaded circuit:

$$D_N^{\mathrm{set}} = (N-1)(6\tau + 2t_{\mathrm{hs}}) + t_{\mathtt{ME}} - \tau.$$

The last two terms in this expression cater for the aforementioned effect of the top level cascade.

2) The chart shown in Fig. 6(b) helps to estimate the delay of the releasing phase, the one between $R1-$ and $G1'+$. Since the partial order segment between these two transitions does not involve the action with the subsequent cascades, this delay is entirely independent of the arbiter size and can be estimated only as:

$$D_N^{\mathrm{rel}} = 5\tau + t_{\mathrm{hs}},$$

i.e. the delay in the path through two latches, for R and $G1$, and one invertor. Recall that in the releasing phase $t_{\mathtt{ME}}$ is assumed to be equal to 2τ.

Now if we assume that N is sufficiently large (as, e.g., in linear or ring interconnections), the D_N^{rel} component is negligible compared to the D_N^{set}. Furthermore, the last two terms in the D_N^{set} can also be neglected. The overall average latency of the N-cascade arbiter built using our circuit can thus be estimated as $(N-1)(6\tau + 2t_{\mathrm{hs}})$.

For comparison, we can estimate the latency for the previously known examples of speed-independent arbiters with the same functional interface. For the one in [10], Fig. 9, we would have $2(N-1)(t_{\mathrm{cl}} + t_{\mathtt{ME}}/2 + 2t_{\mathrm{hs}})$ (here t_{cl} stands for the cumulative delay of the single cell logic), because both delay components (setting and releasing phase) are proportional to N and the ME element's delay is always included into the action cycle in the setting phase. The solution suggested in [3] reduces this latency by excluding the $t_{\mathtt{ME}}/2$ term from it, because the transition $R+$ does not need to wait for the ME element to resolve the conflict between $R1+$ and $R2+$. Thus in the latter case the latency would be estimated as $2(N-1)(t_{\mathrm{cl}} + 2t_{\mathrm{hs}})$.

Now, setting this expression against the one obtained for our arbiter, it is easy to derive that the condition under which our cascaded arbiter operates faster than the one proposed in [3] is that $3\tau < t_{\mathrm{cl}} + t_{\mathrm{hs}}$. This inequality seems to hold because, although the actual cell circuit is represented in [3] at the CMOS transistor level, it is possible to restore it in the gate level form, which has four gate delays acting in series in each phase (thus, $t_{\mathrm{cl}} = 4\tau$). If we assume also that the arbiter was implemented in a distributed way, where the handshakes introduce a considerable delay (transceiver delays are usually much greater than ordinary gate delays), then the effect of releasing the cells in parallel would be crucial.

A more precise estimation and comparison can of course be made only when the implementation technology is taken into account. However, despite some delay overhead paid for organising the parallel release of $G1'$ (or $G2'$) and G' in the cascades, this possible extra delay per cascade has conceivably a smaller effect than the doubling of the delay required by the original solution in [10].

It should be pointed out that the actual circuit implementation of the N-level arbiter may improve on the switching times. For example, it is possible to delete some explicit invertors in the Grant chain and use opposite polarities of R and G in odd/even cascades.

Some authors (e.g., Fig. 5(7) in [1]) use inverted inputs (often depicted by bubbles on the diagrams) to gates, which effectively conceal extra delays and potential hazards in the tacitly assumed invertors. Our circuit, free from any such concealment, may look overly conservative in this sense, but this does not seem to cost us much in both area and performance.

IV. Conclusion

This paper has presented a circuit implementing a cascaded asynchronous arbitration protocol that improves in several ways over existing solutions. First of all, it allows to *pipeline* two phases of the arbitration cycle among levels: 1) the resolution of mutual exclusion, and 2) the release of the grant. In both cases, local actions are conducted in parallel with higher levels of the cascade, thus ensuring a globally reduced latency. Moreover, the circuit has been automatically synthesized from a high-level timing diagram-like specification, and its correctness has been automatically verified. This computer-assisted design methodology allows a substantially faster exploration of the design space, by relieving the designer from the most time-consuming tasks. For example, in this case it allowed trading off pipelining versus circuit complexity.

Acknowledgment

Thanks to M. Josephs of Oxford University for sending us his circuit and M. Kishinevsky of the Technical University of Denmark for his help with the Forcage software used for verifying our circuit. He has also pointed out ways of improving the circuit if a particular implementation technology is known.

References

[1] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* Cambridge, MA: MIT, 1988 (an *ACM Distinguished Dissertation*).
[2] H. J. Genrich and R. M. Shapiro, "Formal verification of an arbiter cascade," in *Proc. 13th Int. Conf. Applicat. and Theory of Petri Nets, Lecture Notes in Computer Sci.*. Berlin: Springer-Verlag, 1992.
[3] M. B. Josephs and J. Yantchev, "Low latency asynchronous arbiter," Patent application 9308161.0., Oxford Univ. Computing Lab., 1993.
[4] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky, *Concurrent Hardware. The Theory and Practice of Self-Timed Design.* London: Wiley, 1993.
[5] A. J. Martin, "Synthesis of asynchronous VLSI circuits," in *Formal Methods for VLSI Design*, J. Staunstrup, Ed. Amsterdam: North Holland, 1990, ch. 6.

[6] R. E. Miller, *Switching theory.* New York: Wiley, 1965, vol. 2, ch. 10, pp. 192–244.
[7] T. Murata, "Petri nets: properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[8] R. C. Pearce, J. A. Field, and W. D. Little, "Asynchronous arbiter module," *IEEE Trans. Comp.*, vol. C-24, pp. 931–932, Sept. 1975.
[9] W. W. Plummer, "Asynchronous arbiters," *IEEE Trans. Comp.*, vol. C-21, pp. 37–42, Jan. 1975.
[10] C. L. Seitz, "Ideas about arbiters," *Lambda*, vol 1, pp. 10–14, First Quarter 1980.
[11] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," *Tech. Rep.* No. UCB/ERL M92/41, Univ. Calif., Berkeley, May 1992.
[12] V. I. Varshavsky, Ed., *Self-Timed Control of Concurrent Processes.* Dordrecht: Kluwer AP, 1990.
[13] A. V. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, "A unified signal transition graph model for asynchronous control circuit synthesis," in *Proc. Int. Conf. on CAD*, Nov. 1992, pp. 104–111.
[14] J. Yantchev and I. Nedelchev, "Implementation of a packet switching device as a delay-insensitive circuit," in *Proc. Int. Symp. on Integrated Circ.*, Seattle, WA 1993.

Power-Delay Characteristics of CMOS Adders

Chetana Nagendra, Robert Michael Owens, and Mary Jane Irwin

***Abstract*—An approach to designing CMOS adders for both high speed and low power is presented by analyzing the performance of three types of adders-linear time adders, log$N$ time adders and constant time adders. The representative adders used are a ripple carry adder, a blocked carry lookahead adder and several signed-digit adders, respectively. Some of the tradeoffs that are possible during the logic design of an adder to improve its power-delay product are identified. An effective way of improving the speed of a circuit is by transistor sizing which unfortunately increases power dissipation to a large extent. It is shown that by sizing transistors judiciously it is possible to gain significant speed improvements at the cost of only a slight increase in power and hence a better power-delay product. Perflex, an in-house performance driven layout generator, is used to systematically generate sized layouts.**

***Index Terms*— Power-delay product, static CMOS adders, transistor sizing.**

I. Introduction

The three most widely accepted metrics for measuring the quality of a circuit are area, delay and power. Minimizing area and delay has always been considered important, but reducing power consumption has been gaining prominence recently [8], [7], [6], [5]. This can be attributed to the increasing popularity of mobile communication systems. Since dramatic improvements in battery technology are not foreseen, low-power designs are crucial not only to lighten the overall weight, but also to reduce the time between recharges. On the other hand, with advances in CMOS technology and reduction of the feature size, area is no longer as scarce a resource as it once used to be. Portability imposes a strict limitation on power dissipation while still demanding high computational speeds as required by real-time tasks. Hence we use the **power-delay product** as the metric of performance in this paper and downplay the importance of the area occupied by the circuit.

Manuscript received December 7, 1993; revised February 24, 1994 and May 10, 1994. This work was supported in part by NSF Grant MIP-9102500.

The authors are with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802 USA.

IEEE Log Number 9403170.

Chandrakasan et. al. describe the *four* degrees of freedom available in the design of low-power circuits and systems, namely technology, circuit design styles, architecture and algorithms [8]. One popular technology family is static CMOS since it has large noise margins and consumes the least power in its class [9]. The dynamic power consumption of a CMOS gate is given by $P = p_f C_L V_{\rm dd}^2 f$, where p_f is the activity factor, C_L is the load capacitance, $V_{\rm dd}$ is the supply voltage and f is the clock frequency. In this paper, we assume that the supply voltage is fixed at 5 V and clock all adders at the same frequency. We attempt to lower the load capacitance and the activity factor of the circuit by playing with the circuit design styles (transistor sizing) and addition algorithms, respectively. We compare the power consumed and the delay of adders using three different addition algorithms, namely, ripple carry addition, blocked carry lookahead addition and signed-digit addition. The choice of the addition algorithm significantly affects p_f, the activity factor of a circuit [8].

Reference [6] compares the power consumption of some CMOS adders. But they do not size transistors whereas practical circuits usually employ transistor sizing to improve speed. Further, estimating the power consumption of signed-digit adders has not been done before.

The remainder of this paper is organized as follows. Section II presents a brief description of the three adder topologies. An overview of the layout generation of the adder circuits using *Perflex*, a performance driven module generator, is given in Section III. In Section IV, the experiments conducted to estimate the energy and the delay of the adders are described along with the performance numbers and we conclude in Section V.

II. Adder Topologies

We have chosen adders with a wide spectrum of timing and complexity which makes it interesting to compare their performance in terms of power and power-delay product. The adders range from the simple but slow (linear time) ripple carry adder to the fairly complex but extremely fast (constant time) signed-digit adders. The third type of adder is the $O(\log N)$ time blocked carry lookahead adder. The N-bit operands in the ripple carry and blocked carry lookahead adders are represented in 2's complement format and the output is generated in the same format. The operands in the signed-digit adder are, of course, signed-digit.

A. Ripple Carry Adder

The basic unit of a ripple carry adder (RCA) is a Full Adder (FA) which computes a sum bit and a carry bit: $s_i = x_i \oplus y_i \oplus c_i$, $c_{i+1} = x_i y_i + x_i c_i + y_i c_i$. That is, it adds the two operand bits with the incoming carry bit to produce a sum bit and an outgoing carry bit. Since, in the worst case, the carry can propagate from the least significant bit position to the most significant bit position, the addition time of an N-bit RCA is $O(N)$.

B. Blocked Carry Lookahead Adder

We use the structure of the "ELM" blocked carry lookahead adder (BCLA) [14]. Compared to the Brent and Kung adder [4], the ELM adder occupies the same area $(O(N \log N))$, but uses fewer