A Low Latency Arbitration Circuit

Alexandre Yakovlev
Department of Computing Science
University of Newcastle upon Tyne, NE1 7RU
England

Alexei Petrov
Laboratory of Automation Technology
Helsinki University of Technology
Otakaari 1, Espoo-02150
Finland

Luciano Lavagno
Dipartimento di Elettronica
Politecnico di Torino
C. Duca degli Abruzzi 24
10129 Torino
Italy

Abstract

Indexing terms: Asynchronous circuits, Arbiters

We present an asynchronous circuit for an arbiter cell that can be used to construct cascaded multi-way 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.

1 Background and Previous Work

Arbitration circuits are commonplace in digital systems with a single resource allocated to different user processes. The typical examples are systems with shared busses, multi-port 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 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 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 multi-way arbiter that arbitrates between two users. Multi-way 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. Figure 1.(a) shows one such cell, a 2-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 Figure 1.(b), illustrates the regular way in which a cascaded multi-way arbiter can be composed from the basic cells. Figure 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.

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. An example of a 2-way arbiter with an analogue ME element[1] was published in [1]. Other examples, together with a rigorous proof of the impossibility of a correct implementation of an arbiter with logical gates, were presented in [2]. 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).

Recently, an elegant solution for the first problem has been suggested by Josephs [2], Yantchev and Nedelchev [3]. Their circuit, although not openly published yet, reportedly uses an OR-functionality to produce the request on R from the arrival of R1 or R2, without waiting for the

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

[2] M.B. Josephs and J. Yantchev, Low Latency Asynchronous Arbiter, patented through Isis Innovation Limited, communication via e-mail on 23 March 1993.

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 has been made in [4]. The published circuit (Figure 11.(a) in [4]) is however not strictly speed-independent because the delays at the outputs of two gates (denoted by *ha3* and *ha4* in [4]) 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]). Another advantage of this circuit is that it has been obtained by using an automatic synthesis approach. First, the behaviour 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.

2 Proposed Solution

2.1 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 [6]. 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 behaviour of a simple 2-input synchroniser (C-element of Muller [8]) is shown in Figure 2.(a). 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 in the 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 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 Figure 2.(a) is shown in Figure 2.(b). 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. The state graph, generated by an STG, which is used as a behavioural specification of a synthesized circuit, 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 transition labelled 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 non-input 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.

2.2 Arbiter Cell Model and Implementation

The behaviour of the arbiter cell solving both the above mentioned problems is defined in the STG shown in Figure 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 problem. The second problem is solved by beginning the resetting of G1 and 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 [9, 10]. The circuit, shown in Figure 4, 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 ([8]). 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 is given in brackets. This circuit consists of an ME element[3], 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.

The performance of the circuit can be evaluated by determining the length of the cycle between two successive settings of, say, R1.

1. With the aid of the STG one can easily trace from the circuit that between R1+ and G1'- our cell introduces the delay of 6τ (each AND-OR-NOT or invertor is regarded as one gate unit delay τ), provided that the delay of the ME is less than the delay of 4τ plus the delay introduced by the remaining cascaded part between R and G'. Thus, for an N-level arbiter, it is realistic to estimate the overall delay between R1+ and G1'- in the first cascade as $(N-1)(6\tau + t_{wire}) + 2\tau + max(3\tau, t_{ME})$, where t_{wire} and t_{ME} are the average delays introduced by the interconnections between cascades and by the ME element (in the arbitration phase), respectively. In simpler terms, we can estimate such a delay as $N(6\tau + t_{wire})$.

2. The releasing phase, between R1- and G1'+, is independent of the arbiter size and is estimated only as $5\tau + t_{wire}$, i.e. the delay in the path through two latches, for R and G1, and one invertor.

Assuming that N is sufficiently large (as, e.g., in linear or ring interconnections), so that the latter component is negligible compared to the first one, the latency of this arbiter is twice as low as the latency of a solution that does not release G1' (or G2') and G' in parallel, in which both delay components are proportional to N.

[3]Its "behavior" is defined by the signals R1, R2, A1 and A2 in the STG and an example of implementation can be found in [1].

Moreover, the above mentioned suggestion by Josephs *et al.* further reduces the latency of our solution with respect to the solution from [1], Figure 9, in which R+ must wait for the ME element to resolve the conflict between R1+ and R2+.

A more precise estimation 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 proposed by [1].

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., Figure 5.(7) in [11]) 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.

Acknowledgements Thanks to 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] C. L. Seitz. Ideas about arbiters. *Lambda*, 1(1, First Quarter):10–14, 1980.

[2] V.I. Varshavsky, editor. *Self-Timed Control of Concurrent Processes.* Kluwer AP, Dordrecht, 1990.

[3] J. Yantchev and I. Nedelchev. Implementation of a packet switching device as a delay-insensitive circuit. In *Proceedings of the Symposium of Integrated Circuits*, 1993.

[4] H.J. Genrich and R.M. Shapiro. Formal verification of an arbiter cascade. In *Proceedings of 13th Int. Conferenece on Application and Theory of Petri Nets*, 1992. Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[5] A.J. Martin. Synthesis of asynchronous vlsi circuits. In J. Staunstrup (ed.), editor, *Formal Methods for VLSI Design*, chapter 6. North Holland, Amsterdam, 1990. IFIP WG 10.5 Lecture Notes.

[6] A. V. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified signal transition graph model for asynchronous control circuit synthesis. In *Proceedings of Int. Conf. on CAD*, November 1992.

[7] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, April 1989.

[8] R. E. Miller. *Switching theory*, volume 2, chapter 10, pages 192–244. Wiley and Sons, 1965.

[9] 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. Technical Report UCB/ERL M92/41, U.C. Berkeley, May 1992.

[10] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky. *Concurrent Hardware. The Theory and Practice of Self-Timed Design.* John Wiley and Sons Ltd., February 1993. To appear.

[11] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* The MIT Press, Cambridge, Mass., 1988. An ACM Distinguished Dissertation 1988.

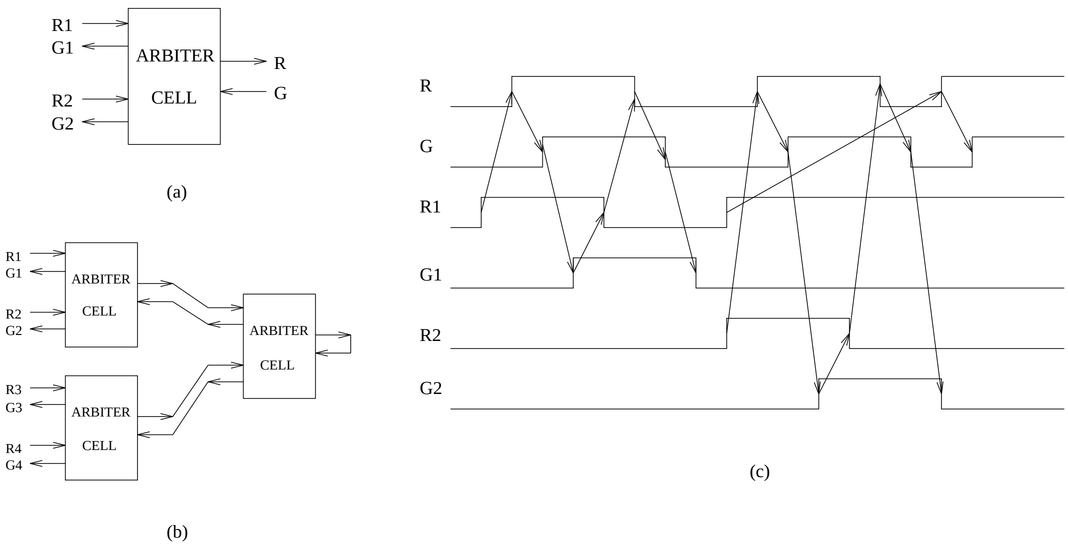

Figure 1: A two-input arbiter cell

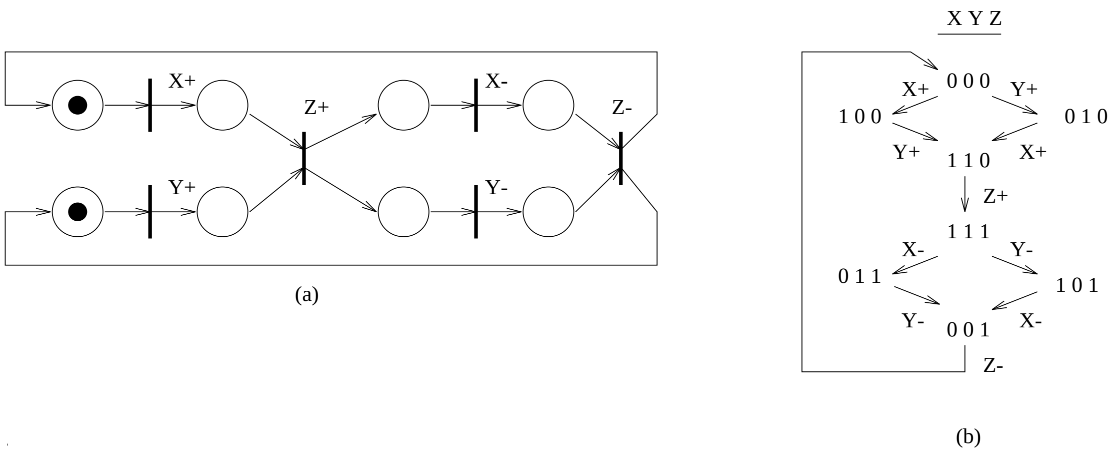

Figure 2: A Signal Transition Graph example

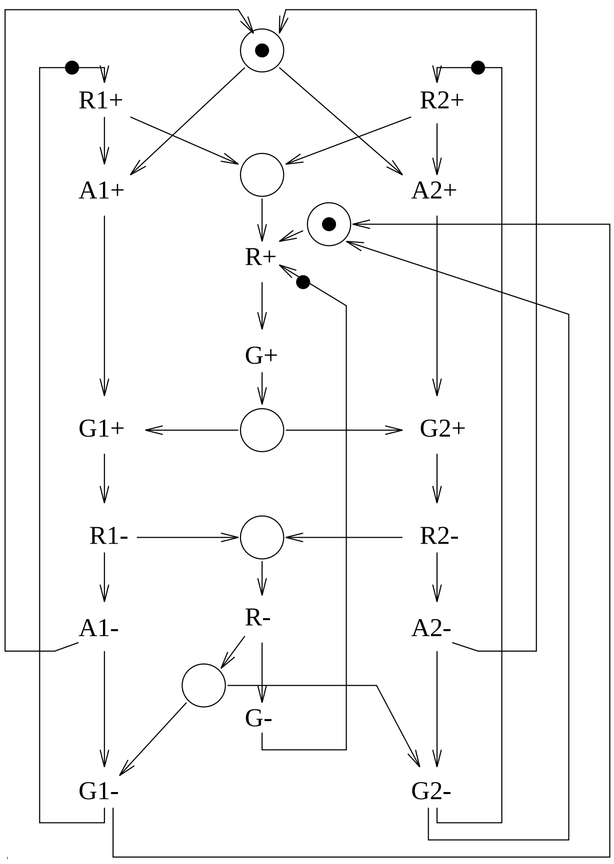

Figure 3: The arbiter Signal Transition Graph specification

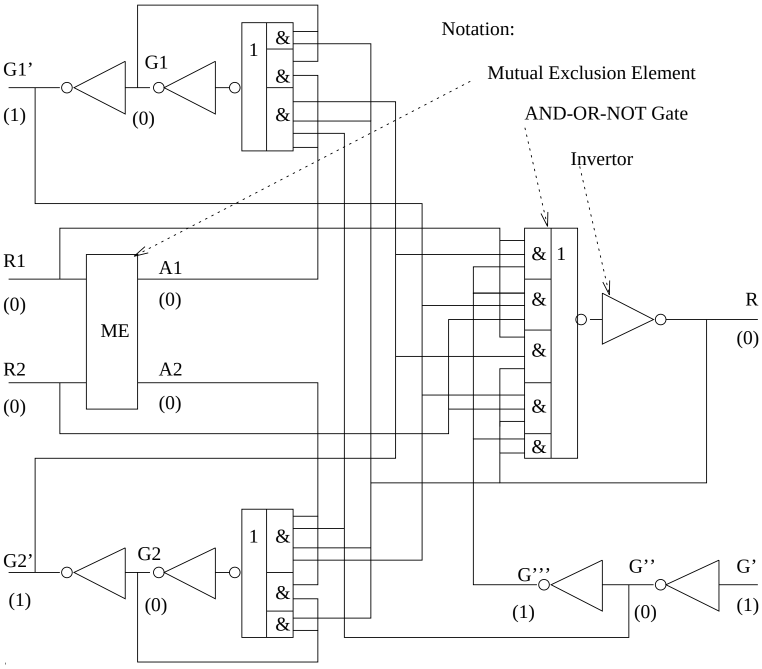

Figure 4: The arbiter circuit implementation