School of Electrical, Electronic & Computer Engineering

Soft arbiters

Andrey Mokhov, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2009-149

August 2009

Contact:

Andrey.Mokhov@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grants EP/C512812/1 and EP/F016786/1

NCL-EECE-MSD-TR-2009-149

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

`http://async.org.uk/`

Soft arbiters

Andrey Mokhov, Alex Yakovlev
Microelectronics System Design Group, Newcastle University, UK

Abstract

The paper proposes a new type of arbiters which we call 'soft arbiters' as opposed to traditional ones that are built with the 'strict' arbitration scheme in mind: an m-of-n strict arbiter is bound to issue exactly m grants having received n requests. Instead, behaviour of an m-of-n soft arbiter is less restrictive: it is allowed to issue a different number of grants $k \neq m$ occasionally, as long as the average number of active grants converges to m over time; consequently m does not necessarily have to be an integer value.

Besides interesting theoretical properties and challenges the new type of arbiters presents, it has certain practical application. Due to the less restrictive behaviour soft arbiters can have smaller and faster implementation than their strict counterparts. Therefore it is beneficial to utilise them in those application areas which do not require the use of the strict arbitration scheme.

1 Introduction

Arbiters are basic controllers that manage concurrent access to system resources [4]. The notion of a resource in this context should not be narrowed down to that of a shared component; other system resources, such as energy, or processing/communication bandwidth should also be considered.

Often being on the critical path (with respect to performance) of a system arbiters have been subject to continuous optimisation attempts which led to development of different arbitration schemes and topologies: token ring [5], balanced trees [3], locking [1], flat [6], and other arbitration approaches have been proposed since 1970s. We look at this long-standing problem from a different angle. Instead of optimisation of general arbiters, we outline a class of systems for which the traditional definition of an arbiter can be relaxed thus leading to simpler circuit implementation.

Suppose there are n clients using a system resource and we want no more than m of them to access it simultaneously. This bound on the number of concurrent accesses to the resource may come from different reasons:

1. the system cannot physically serve more than m clients, e.g. it has only m processing units;
2. we want to reduce concurrency due to power management issues (exceeding m may lead to inefficient energy consumption or overheat);
3. efficiency of a Network-on-Chip reduces when the number of active connections exceeds m (oversaturation of the network, see [7] for analysis of this problem).

In the first case the bound has to be strict: granting access to $m + 1$ clients simultaneously leads to an unavoidable collision of two clients trying to access the same processing unit. However, cases (2) and (3) are more flexible by their nature: random, occasional events have little effect on the inertial, statistical characteristics of power consumption and network traffic; a system is likely to tolerate them. Therefore, in these cases the bound may be relaxed, and granting access to more than m clients may be allowed as long as the rate of such events is acceptable. It is undoubtedly necessary to provide formal criteria for a rate to be 'acceptable', – this issue will be addressed later.

The next section discusses modelling of strict and soft arbiters using STGs; Section 3 studies example of a soft arbiter implementation. Opportunities for the real-time control over 'softness' of an arbiter are studied in Section 4. A summary of the presented material can be found in Section 5.

2 Model

The classic specification of the strict m-of-n arbiter in the form of STG [2] is given in Figure 1. To access the resource, the i-th client sends a request to the arbiter by raising signal r_i. The clients are independent from each other, therefore their requests can arrive concurrently. In response, the arbiter issues a grant by raising signal g_i, provided that there is at least one token in the choice place p. Initially p contains m tokens, therefore it is guaranteed that at most m grants can be high at any time, no matter how many requests have been received by the arbiter. Upon receipt of the grant, a client can safely use the resource. Having finished, the client lowers its request r_i, and in response the arbiter lowers the grant g_i and returns the token to place p.

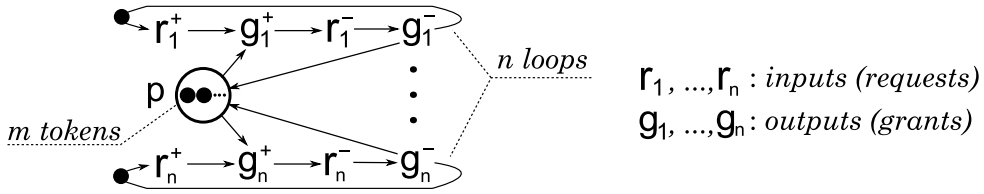


Figure 1: STG specification of a strict m-of-n arbiter

To model soft arbiters it is necessary to introduce the notion of time or probability into the STG specification as shown in Figure 2. During the normal operation of the arbiter (the strict mode), m tokens circulate between places p and q. A token from place b may be borrowed and introduced temporary into the arbitration loop via transition Δ which is timed, i.e. it has a certain delay Δ associated with its firing (it is the only timed transition in this STG; firing times of the other transitions are considered to be negligible). Introduction of the $(m+1)$-th token into the loop turns the arbiter into the soft mode and simultaneous issue of $m+1$ grants becomes possible. Absence of a token in place b leads to disconnection of the route between places q and p. This forces the arbitration loop to return the borrowed token as soon as possible. The next borrowing will be possible only after Δ time units.

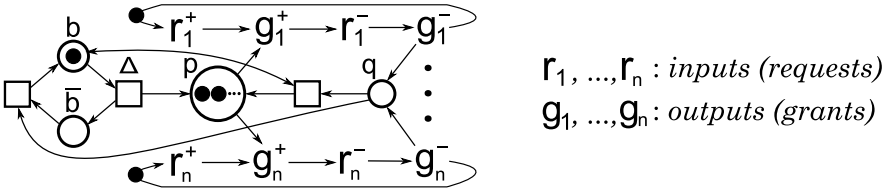


Figure 2: STG specification of a soft m-of-n arbiter

This particular specification has the following properties:

- The specified soft arbiter acts as an $[m;\ m+1]$-of-n soft arbiter, i.e. it issues m or $m+1$ grants to n clients. We say that m and $m+1$ are *lower* and *upper grant bounds* of the arbiter.
- Events of issue of $m+1$ grants simultaneously are at least Δ time units apart. We call this class of arbiters *timed soft arbiters*.

It turns out that timed soft arbiters are not easier for implementation than strict arbiters, because the 'softness' is imposed from the design point of view. As a result, one has to take implementation of a strict arbiter and introduce some form of timed token injection into it (yet another sort of restriction!), thus

defeating the whole purpose. A proper way of obtaining an efficient soft arbiter is to take an existing design of a strict arbiter and try to optimise it by *removing* certain restrictions, not by adding them. The next section shows an example of this approach.

3 Implementation example

Figure 3 shows implementation of the 1-of-3 flat arbiter [6]. As one can see it has a rather complicated construction with four layers: pairwise arbitration, reset filters, computation of the winner, and finally the layer of completion detection. The complexity comes from the fact that the *mutual-exclusion (ME) elements* [4] in the first layer (boxes with labels mAB, mAC, and mBC) can resolve in a contradictory way, and it is sometimes necessary to amend (some of) their decisions so that to detect a single winning request (see details in [6]). More than half of the circuit area and latency is spent on handling these rare occasions of contradictory ME elements resolution and thus guarantee the strict arbitration protocol.

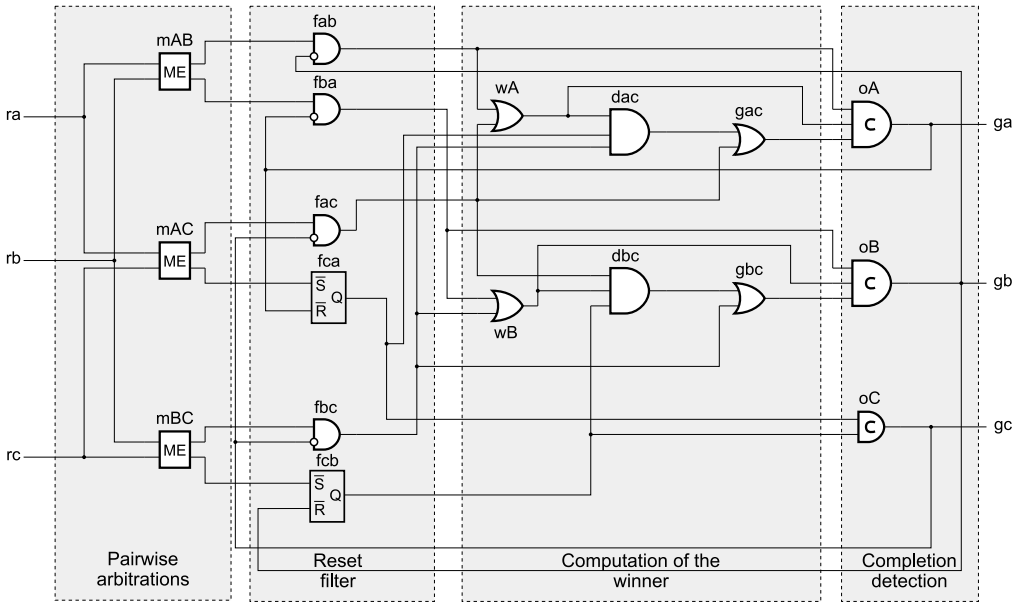
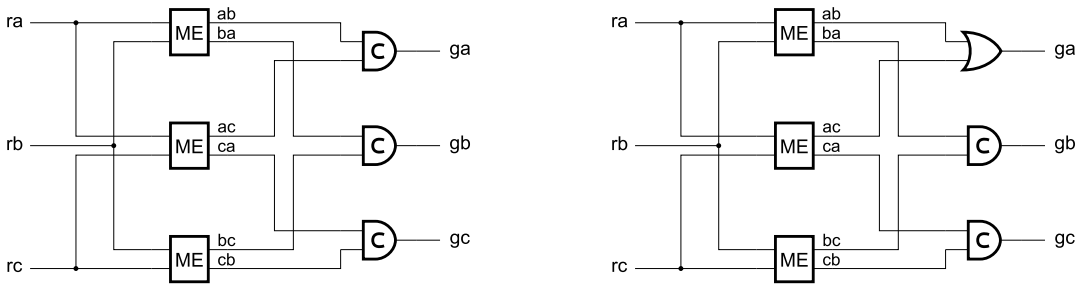


Figure 3: Implementation of the 1-of-3 strict arbiter

Figure 4(a) shows the modified version of the above circuit after removal of the two middle layers. The circuit is speed-independent and conforms to the strict arbitration protocol but it has deadlocks which occur if ME elements resolve contradictorily, e.g. $ab = bc = ca = 1$.



(a) Implementation with deadlocks

(b) Soft implementation

Figure 4: Simplified implementations of the 1-of-3 flat arbiter

The simplest way to eliminate the deadlocks is to substitute one of the C-elements with an OR-gate as shown in Figure 4(b). As a consequence, the circuit issues either one or two grants depending on the order of request arrival, i.e. it is [1; 2]-of-3 soft arbiter. Strictly speaking this circuit is also

not speed-independent, because grant ga can be generated in the OR-causal manner before both ME elements connected to request ra have been resolved; the one which is still unresolved may be disabled prematurely by event $ra-$. In practice it can be safely assumed that resolution of an ME element takes less time than access of a client to a resource. If the timing assumption is not acceptable, it is possible to fix this problem by adding the appropriate completion detection circuitry; we prefer circuit in Figure 4(b) here as it is easier for visual comprehension.

Let us study behaviour of the obtained [1; 2]-of-3 soft arbiter. Table 1 shows which grants are issued for a given order of requests. The last two rows correspond to those rare cases, when the result of pairwise arbitration is contradictory, and thus the arbiter cannot principally detect the order of requests arrival; in these cases it issues grant ga. In the other six cases the arbiter always gives the grant to the first request, plus to request ra if it came second. Thus, for a single burst of incoming requests, the probability of the arbiter to issue two grants is $\frac{1}{3}$ (assuming the probability of a contradictory decision to be zero). On the other hand, if there is a constant flow of incoming requests $abcabcabc...$ than the arbiter will be giving one and two grants alternatingly, effectively behaving as an 1.5-of-3 soft arbiter (the series of grants will be $ga - gb - \{ga,\ gc\} - gb - \{ga,\ gc\} - ...$ etc).

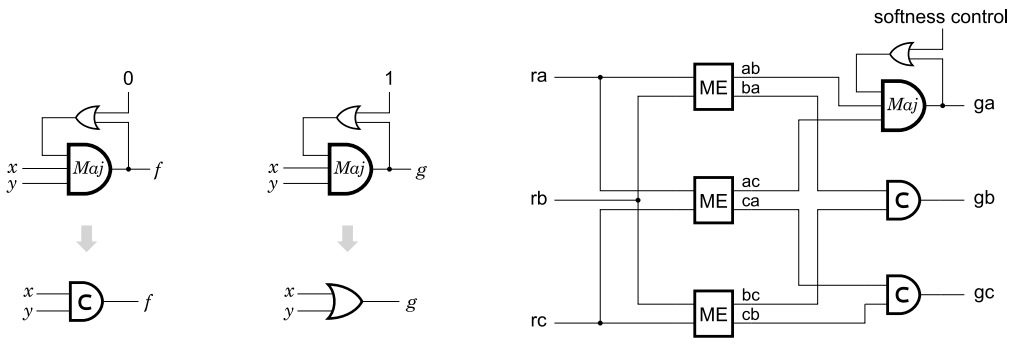
Request order	ab/ba	ac/ca	bc/cb	Issued grant(s)
abc	ab	ac	bc	ga
acb	ab	ac	cb	ga
bac	ba	ac	bc	$ga,\ gb$
bca	ba	ca	cb	gb
cab	ab	ca	cb	$ga,\ gc$
cba	ba	ca	cb	gc
impossible to detect	ab	ca	bc	ga
impossible to detect	ba	ac	cb	ga

Table 1: Analysis of arbiter in Figure 4(b) with respect to request orders

The flat arbitration scheme [6] gives opportunity to build soft arbiters with the amount of 'softness' controlled in real-time as explained in the next section.

4 Real-time control

Looking at the circuits in Figure 4 one can see that the only difference between them is the function of the gate generating grant ga: C-element (a) and OR-gate (b). It is possible to use a *majority gate* $Maj(x,\ y,\ z) = xy+xz+yz$ to implement both types of behaviour: function $f = Maj(x,\ y,\ f)$ is equivalent to the function of a C-element $f = C(x,\ y)$, while $g = Maj(x,\ y,\ 1)$ is equivalent to $g = OR(x,\ y)$ as shown in Figures 5(a, b).



(a) C-element (b) OR-gate (c) Soft arbiter with real-time softness control

Figure 5: Real-time 'softness' control using majority function

Figure 5(c) shows implementation of the arbiter with real-time softness control using this idea. To guarantee robust behaviour of the arbiter the control input must be changed only when request ra is inactive. It is possible to 'soften' other outputs as well, thereby obtaining a variety of arbiters with different bounds on softness. The most controllable one (with three control bits) behaves like a valve which can restrict the flow of requests to a specified degree (including the scenario when it is 'fully open' or transparent to all the requests).

5 Conclusions

The paper introduced the concept of soft arbitration and outlined its application area. Implementation of [1; 2]-of-3 soft arbiter is given and discussed. Real-time control of the arbitration 'softness' is possible on the basis of flat arbitration scheme, which is demonstrated with the example of variable 3-way soft arbiter.

Future work is focused on development of a robust n-way arbiter with controllable softness for practical values of n, as well as investigation of different speed-independent implementations and realistic timing assumptions that can be used to optimise them.

Acknowledgement

This work was supported by EPSRC grants EP/C512812/1 and EP/F016786/1.

References

[1] A. Bystrov, D. J. Kinniment, and A. Yakovlev. Priority arbiters. In *Proc. of the 6th International Symposium on Asynchronous Circuits and Systems (ASYNC'2000)*, page 128. IEEE Computer Society, 2000.

[2] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. *Logic synthesis of asynchronous controllers and interfaces.* Advanced Microelectronics. Springer-Verlag, 2002.

[3] Mark B. Josephs and Jelio T. Yantchev. CMOS design of the tree arbiter element. *IEEE Transactions VLSI Syst.*, 4(4):472–476, 1996.

[4] David J. Kinniment. *Synchronization and Arbitration in Digital Systems.* John Wiley and Sons, 2008.

[5] A J Martin. The design of a self-timed circuit for distributed mutual exclusion. In *Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration*, 1985.

[6] Andrey Mokhov, Victor Khomenko, and Alex Yakovlev. Flat arbiters. In *Proc. of 9th International Conference on Applicatioon of Concurrency to System Design (ACSD'09)*, 2009.

[7] Shufan Yang, Steve Furber, Yebin Shi, and Luis A. Plana. An admission control system for QoS provision on a best-effort GALS interconnect. In *Proc. of 8th International Conference on Applicatioon of Concurrency to System Design (ACSD'08)*, pages 200–207, 2008.