Newcastle University e-prints

Date deposited: 31s^{t} March 2010

Version of file: Author final

Peer Review Status: Peer Reviewed

Citation for published item:

Mokhov A, Khomenko V, Yakovlev A. Flat Arbiters. *In: NINTH INTERNATIONAL CONFERENCE ON APPLICATION OF CONCURRENCY TO SYSTEM DESIGN, PROCEEDINGS*.2009,10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA:IEEE COMPUTER SOC

Further information on publisher website:

http://www.ieee.org

Publishers copyright statement:

The definitive version of this article, published by IEEE , 2009 and available at the DOI below

10.1109/TERAMIR.2009.5379644

The definitive version should always be used when citing.

Use Policy:

The full-text may be used and/or reproduced and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not for profit purposes provided that:

- A full bibliographic reference is made to the original source
- A link is made to the metadata record in Newcastle E-prints
- The full text is not changed in any way.

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Robinson Library, University of Newcastle upon Tyne, Newcastle upon Tyne. NE1 7RU. Tel. 0191 222 6000

Flat Arbiters

Andrey Mokhov[1] Victor Khomenko[2] Alex Yakovlev[1]
[1]School of Electrical, Electronic and Computer Engineering
[2]School of Computing Science
Newcastle University, UK
{Andrey.Mokhov, Victor.Khomenko, Alex.Yakovlev}@ncl.ac.uk

Abstract

A new way of constructing N-way arbiters is proposed. The main idea is to perform arbitrations between all pairs of requests, and then make decision on what grant to issue based on their outcomes. Crucially, all the mutual exclusion elements in such an arbiter work in parallel.

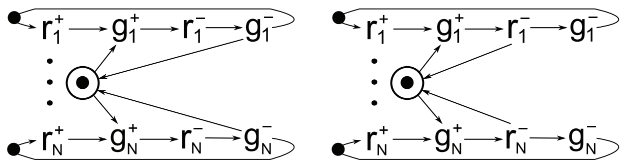
This 'flat' arbitration is prone to new threats such as formation of cycles (leading to deadlocks), but at the same time opens up new opportunities for designing arbitration structures with different decision policies due to the availability of the global order relation between requests. To facilitate resolution of such cycles and further developments in the context of flat arbitration, the paper presents new theoretical results, including a proof of correctness of a generic structure for the N-way arbiter decision logic. In particular, in some situations a request that lost some pairwise arbitrations has to be granted to avoid a deadlock.

Keywords: *Arbiters, Speed-independent circuits, Asynchronous circuits, Signal Transition Graph (STG).*

1. Introduction

Arbiters [11] are basic blocks guarding access to shared resources and, as such, they play a very important role in circuit design. Hence their efficient and correct implementation is essential.

The specification of an N-way arbiter, in the form of a Signal Transition Graph (STGs are explained in Sect. 2) is shown in Fig. 1(left). Suppose there are N clients using a shared resource in a mutually exclusive way. Before accessing the resource, the i^{th} client sends a request to the arbiter (by raising signal r_i). Such requests can be sent concurrently by different clients. In response, the arbiter issues a grant (by raising signal g_i). At most one of $g_1, \ldots, g_N$ can be high at any time, no matter how many concurrent requests have been received by the arbiter. Upon receipt of the grant, a client can safely use the resource, with the guarantee that no interference is possible from other clients. Having finished using the resource, the client lowers its request r_i, and in response the arbiter lowers g_i. At this point the arbiter can issue a grant to another client.



inputs: $r_1, \ldots, r_N$ **outputs:** $g_1, \ldots, g_N$

Figure 1. A specification of an N-way arbiter: the traditional and early protocols.

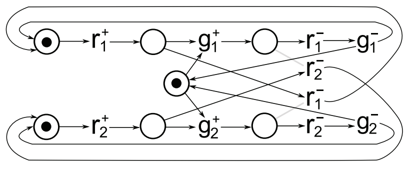
An alternative *early* protocol (see the notion of *output-delay-insensitisation* [7]) is shown in Fig. 1(right); the difference here is that once the i^{th} client lowers the request r_i, the arbiter is allowed to immediately issue a grant g_j ($j \neq i$) to another client, in parallel with lowering the grant g_i. Hence, g_i and g_j can be simultaneously high, but this is harmless since the i^{th} client has already declared (by lowering r_i) that it had finished using the resource, and, according to this early protocol, it will not send another request (i.e. raise r_i again) until the arbiter lowers g_i. In fact, any QDI circuit (see Sect. 2) that assumes the traditional protocol will work correctly with the early protocol as well: Indeed, the delay in producing g_i can be much longer than that in producing g_j (e.g. due to a longer wire — which is allowed by the definition of QDI); thus, the arbiter executing the traditional protocol may reset g_i and quickly set g_j, before the change in g_i has been noticed by other parts of the circuit (note the similarity with the early protocol). Hence, a glitch is possible unless the circuit permits also the early protocol.

N-way arbiters are usually constructed using basic 2-way *mutual exclusion (ME) elements.* The behaviour of an ME element is given by Fig. 2. Note that in particular it implements a 2-way arbiter, but it allows for some additional behaviour of the environment, e.g. the trace

$$r_1^+ \; r_2^+ \; g_1^+ \; r_2^-$$

can be performed by the ME element but not by the 2-way arbiter in Fig. 1(left) for $N = 2$. This turns out to be important for the purposes of this paper.

It is a well-known fact that one cannot construct even a 2-way arbiter using only digital logic gates [18]. Indeed,



inputs: r_1, r_2 **outputs:** g_1, g_2

Figure 2. A specification of an ME element.

when the two requests arrive almost simultaneously, the ME element, like Buridan's ass, has to make an arbitrary choice between them. It enters a *metastable* state, in which it can stay indefinitely. To prevent the ME element from outputting the near-threshold values during the metastability resolution process, an analog filter is employed [17].

Though the time for resolving metastability is exponentially distributed, and so most arbitrations are fast, the fact that there is no upper bound on this time means that circuits that require the arbitration decision within bounded time (e.g. by the start of the next clock cycle) occasionally fail. One of traditional ways of designing N-way arbiters is to combine the basic ME elements in a balanced tree-like fashion. This design is simple and results in a small circuit; however, its disadvantage is that several ($\log N$) arbitrations happen sequentially. This can significantly increase the latency of the arbiter, especially in balanced circuits where the requests usually arrive almost simultaneously, and so several sequential ME elements, one after another, can spend long time in their metastable states.[1]

In this paper we propose an alternative way of constructing N-way arbiters. The main idea is to perform concurrent arbitrations between all pairs of requests, and then make the decision on what grant to issue based on their outcomes, see Fig. 3. Crucially, all the ME elements in such an arbiter work in parallel (hence the name 'flat arbiter'), and the subsequent decision logic has bounded latency. Of course, this comes at the price of increasing the number of ME elements from $N - 1$ (in tree arbiter) to $N(N - 1)/2$ (in flat arbiter) and somewhat complicated decision circuitry.

The idea of flat arbitration is fundamentally different from the traditional tree-based, ring-based and other decompositions. The flat structure produces the nearest approximation to the actual order of arrival of requests, which can be represented by an arbitration matrix storing the results of pairwise arbitrations. This information can be used in its full entirety, i.e. globally, to compute various kinds of decisions about grants. In particular, out of necessity to avoid deadlocks, in certain situations flat arbiters have to grant a request that has not won every arbitration it is involved in. Also, as requests can arrive at any time and no assumptions are made on the response time of ME elements, before granting a request the decision logic must witness that it has either won or lost each arbitration in which it is involved.

[1]Some sophisticated tree arbiter designs address this problem by early propagation of requests, see [8].

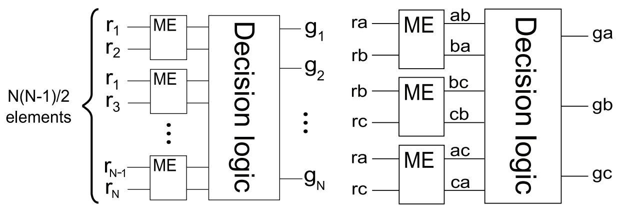


Figure 3. The top-level view of the flat arbiter and its specialisation for $N = 3$.

Tree and ring based structures have only partial information about some arbitration relations, which of course has the advantages of reducing the interconnect complexity and leading to simpler circuit structures. On the other hand, the availability of the global information in flat arbitration can be advantageous in producing different sorts of decision (e.g. up to $m < N$ requests can be granted). The overall complexity of the logic and how it can be made less sensitive to delays in gates also depends on the discipline. Two particularly distinct disciplines are:

- Generate one grant (winner) from those arrived so far, which is basically the traditional arbiter protocol.
- Generate a table of competition (i.e. a total order) or a single winner from the full set of N requests arriving in parallel — here the protocol is such that as soon as the result is computed, all requests must withdraw.

2. Basic Notions

This paper is concerned with asynchronous arbiters, as opposed to synchronous ones, where all the requests arrive at a clock edge and a deterministic decision can be made (i.e. there is no need to resolve metastability). In asynchronous arbiters, the inputs arrive asynchronously, and clock cannot be used as a frame of reference. Note that asynchronous arbiters can be employed by either synchronous or asynchronous circuits (e.g. in Globally Asynchronous Locally Synchronous (GALS) circuits they are often used in the interfaces between the synchronous and asynchronous parts of the circuit). Hence below we briefly explain asynchronous circuits and Signal Transition Graphs — a Petri net based formalism which is widely used for specifying asynchronous circuits. We then introduce some terminology used in designing flat arbiters.

Asynchronous Circuits (ACs) are circuits without clocks. ACs have been getting much attention in the last years, as they often have lower power consumption and electro-magnetic emission, no problems with clock skew and related subtle issues, and are fundamentally more tolerant of voltage, temperature and manufacturing process variations. The International Technology Roadmap for Semiconductors report on Design [6] predicts that 22% of the designs will be driven by handshake clocking (i.e. asynchronous) in 2013, and this percentage will rise up to 40% in 2020.

Though the listed advantages look rather attractive in the view of the current microelectronics design challenges, correct and efficient ACs are notoriously difficult to design — there are examples of published ACs which subsequently turned out to be incorrect [4].

In this paper we focus on an important subclass of ACs, called *speed-independent (SI)* circuits; this model follows the classical approach of Muller [14] and regards each gate as an atomic evaluator of a Boolean function, with a delay element associated with its output. In the SI framework this delay is unbounded, i.e. the circuit must work correctly regardless of its gate delays, and the wires are assumed to have negligible delays (or, alternatively, wire forks are assumed to be isochronic — in such a case the circuit is often referred to as *quasi-delay-insensitive (QDI)* [12]).

Signal Transition Graphs (STGs) are a particular type of labelled Petri net developed specifically for modelling asynchronous digital circuits [2, 3, 16]. The idea is to associate a set of Boolean variables, referred to as *signals*, with a Petri net to represent the state of the actual digital signals (i.e. wires) within a circuit. The Petri net's transitions are then labelled to represent changes in the state of these signals; a transition label either has the form a^+ to indicate a signal a goes from 0 to 1, or a^- to indicate the signal goes from 1 to 0. (In general, several transitions can have the same label.) Thus, the underlying Petri net specifies the causal relationship between signals and is intended to capture the behaviour of a circuit. For an STG to correctly represent a circuit one has to ensure that the labels a^+ and a^- are correctly alternated between for each signal.

An STG can be represented graphically simply as a labelled Petri net. However, a short-hand notation is often used, in which transitions are simply represented by their labels, and places with one incoming and one outgoing arc are contracted (see e.g. Fig. 1). Moreover, we use a single grey line without arrowheads to represent *read arcs*, i.e. pairs of arcs (p, t) and (t, p) with the same end nodes and opposite directions. Such arcs are used to test for the presence of a token in a place without consuming it.

The signals of an STG are partitioned into *input*, *output* and *internal* signals; the output and internal signals are collectively referred to as *local* signals. The inputs are controlled by the environment of the STG, and the outputs are controlled by the circuit itself and are observable by the environment. Internal signals represent some auxiliary entities needed to produce outputs; like outputs, they are controlled by the circuit, but are not observable by the environment.

The behaviour of an STG is based on its underlying Petri net's behaviour; in particular, the concepts of enabling and firing of transitions are the same. Intuitively, an STG represents a contract between the circuit and its environment, and is interpreted in the following way. If an input signal transition is enabled, then the environment is allowed (but is not obliged) to send this input, and the environment is not allowed to send inputs which are not enabled. If a local transition is enabled, then the circuit is obliged eventually to produce this signal (or it is eventually disabled by another transition, in which case the *output-persistency,* discussed later, is violated), and vice versa, it is not allowed to produce outputs which are not enabled. That is, an STG specifies the behaviour of a circuit in the sense that the circuit must provide *all and only* the specified outputs, and that it must allow *at least* the specified inputs (in fact, it could optionally allow more inputs, which means that it could work in a more demanding environment).

For an STG to be directly implementable by an SI digital circuit the following properties should be satisfied [3]:

Boundedness The underlying Petri net of the STG should be bounded, which is equivalent to requiring that it has a finite number of reachable markings.

Consistency In each execution, the transitions representing the rising and falling edges of each signal must be correctly alternated between, always starting from the same edge.

Complete State Coding (CSC) If the STG has two reachable states in which the values of all the signals coincide but the set of enabled local signals are different, then these two states are said to be in *Complete State Coding (CSC)* conflict. The STG satisfies the *CSC property* if no two of its reachable states are in CSC conflict.

An STG not satisfying the CSC property cannot be directly implemented as an SI circuit. Intuitively, during its execution the circuit can 'see' only the values of its signals, but not the marking of the STG. Hence, if two semantically different reachable states with the same values of all the signals exist, the circuit cannot distinguish between them, and so cannot know what to do next.

Output-persistency (OP) If some local signal becomes enabled, it cannot be disabled by firing some other transition, i.e. there should be no choices involving local transitions. The rationale for this is that once a signal becomes enabled, its voltage starts to change, e.g. to rise from 0 to 1. If the signal is disabled during this process, the voltage is suddenly pulled down, resulting in a glitch. This glitch can be interpreted in different ways by the logic gates 'listening' this signal, depending on whether or not the voltage has crossed the threshold between 0 and 1. Hence the behaviour of the circuit becomes non-deterministic and non-digital.

Visually, if OP is violated then there are two transitions with different labels in the STG with at least one of them marked by a local signal, which share some pre-places and can be enabled simultaneously (unless both transitions are connected to these shared pre-places by read arcs).

Note that a choice involving only inputs is not a violation of OP, and simply models a choice in the environment. Since this choice does not have to be implemented by the circuit, SI circuits can be synthesised for such STGs (provided that all the other conditions necessary for SI are met).

A choice involving only local transitions (such choices are fundamental in arbiters, e.g. consider the choice between the transitions g_i^+ in Fig. 1) can still be implemented in a speed-independent way (in spite of the violation of OP) using ME elements. Note that there is no contradiction here, as ME elements are not fully digital and contain some analog circuitry to properly handle the metastable behaviour associated with such a choice. In current practice, however, ME elements are 'factored out' to the environment, so that the remaining part of the circuit can be specified by an OP STG that can be synthesised as a fully digital circuit. In particular, for the flat arbiter circuit in Fig. 3, the ME elements form a part of the environment for the 'Decision logic' sub-circuit. That is, even though the outputs of these ME elements are internal signals from the point of view of the whole circuit, they are inputs from the point of view of the 'Decision logic' sub-circuit; therefore, the corresponding choices in the STG modelling this sub-circuit will be between inputs, and thus not constitute OP violations.

STGs are supported by a range of tools, such as PETRIFY [3] and MPSAT [9,10], which are able to analyse their behaviour as well as synthesise circuits from them.

Arbitration matrices and digraphs Suppose an observer monitors the outputs of the ME elements in Fig. 3(left) and records the snapshots in a Boolean $N \times N$ *arbitration matrix* $\mathcal{A}$, where $\mathcal{A}_{ij}=1$ iff r_i won the arbitration with r_j, and $\mathcal{A}_{ij}=0$ otherwise ($\mathcal{A}_{ii}=0$ for all i). Note that it is possible that $\mathcal{A}_{ij}=\mathcal{A}_{ji}=0$ for some $i \neq j$ — this would indicate that the arbitration between r_i and r_j has not completed yet or neither request has arrived; however, the situation $\mathcal{A}_{ij}=\mathcal{A}_{ji}=1$ is impossible.

We will denote by $\widehat{\mathcal{A}}$ the symmetric closure of a matrix $\mathcal{A}$, defined as $\mathcal{A}+\mathcal{A}^\top$, where $\top$ is the matrix transposition operator. Observe that in order to prove some property for all elements of $\widehat{\mathcal{A}}$ it is sufficient to consider the elements $\widehat{\mathcal{A}}_{ij}$ such that $i \le j$. Observe that $\widehat{\mathcal{A}}_{ij}=1$ indicates that at least one of r_i and r_j has arrived, and the arbitration between r_i and r_j has been completed.

Since the inputs of the ME elements are not observed, it is impossible to tell if a request r_k has arrived unless it has at least one win (note that r_k having lost some arbitration does not mean that it has arrived, as the signals that have not arrived always lose to those which have). Hence we say that a request is *observable in* $\mathcal{A}$ if it has at least one win.

An arbitration matrix $\mathcal{A}$ is called *stable w.r.t. a request* r_i if all the arbitrations in which r_i participates have completed, i.e. $\widehat{\mathcal{A}}_{ij}=1$ for all $j \neq i$. Otherwise $\mathcal{A}$ is called *unstable w.r.t.* r_i. Note that $\mathcal{A}$ can be stable w.r.t. r_i even if r_i has not arrived — in such a case r_i has lost the arbitrations with all the other requests. $\mathcal{A}$ is called *stable* if it is stable w.r.t. all the requests observable in $\mathcal{A}$, and *unstable* otherwise. Intuitively, when a burst of requests arrives, the ME

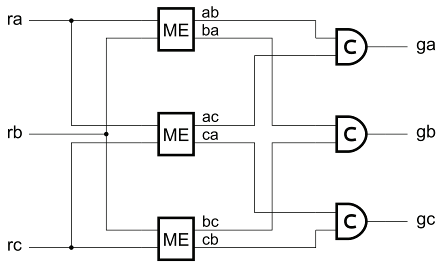


Figure 4. The basic early 3-way flat arbiter.

elements arbitrate between them, and some of the entries of the arbitration matrix are changed from 0 to 1 when these arbitrations complete, hence the term 'unstable'. Once all these arbitrations are complete, the matrix does not change (unless new requests arrive), hence the term 'stable'.

Note that if $\mathcal{A} \neq \mathbf{0}$ is stable then the decision circuitry of the flat arbiter should eventually generate a grant, even if no new requests arrive (if it does not, we call this situation a deadlock, even though formally the circuit can progress by sending new requests). On the other hand, it is not obliged (but is allowed) to generate a grant for an unstable matrix, or can opt to wait for it to become stable. This is important for our correctness proof: the states in which the arbitration matrix is unstable cannot be deadlocked, while stable matrices have some structure allowing the proof to proceed.

$\mathcal{A}$ can be interpreted as the adjacency matrix of some digraph, which we call an *arbitration digraph.* Its vertices correspond to the requests (we will denote the vertex corresponding to r_i also by r_i), and an arc (r_i, r_j) indicates that r_i has won the arbitration with r_j. Note that due to the properties of $\mathcal{A}$ the corresponding arbitration digraph has no self-loops and 2-cycles, but can have other cycles.

In what follows, we use the terms arbitration matrix and arbitration digraph interchangeably, e.g. we could say that an arbitration matrix 'has a cycle' or is 'acyclic', meaning that the corresponding digraph has the respective property.

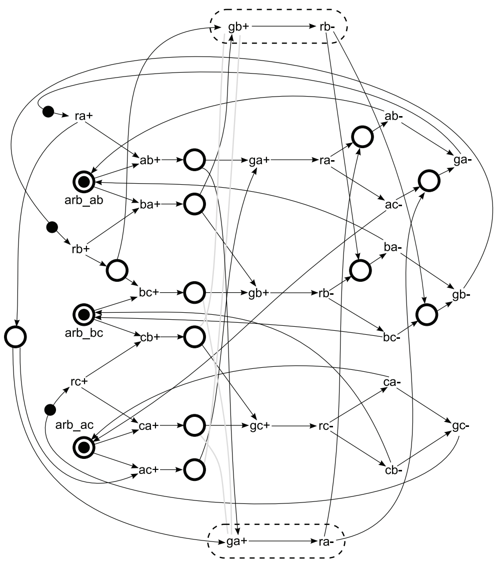
3. Basic flat arbiter

The basic N-way early flat arbiter can be constructed as follows. Given an arbitration matrix $\mathcal{A}$, the grant signals g_k, $k = 1 \dots N$, are generated by N C-elements:

$$g_k = C(\mathcal{A}_{k1}, \mathcal{A}_{k2}, \dots, \mathcal{A}_{k(k-1)}, \mathcal{A}_{k(k+1)}, \dots, \mathcal{A}_{kN}),$$

where $C(x, y, \dots, z)$ represents the output of a C-element with inputs $(x, y, \dots, z)$. The intuition here is that a grant g_k can be issued only if the corresponding request r_k won the arbitrations with all the other requests (i.e. $\mathcal{A}_{kj}=1$ for all $j \neq k$), and g_k can be reset as soon as the r_k is reset and this fact has been acknowledged by the arbiters by resetting the corresponding outputs (i.e. $\mathcal{A}_{kj}=0$ for all $j \neq k$).

The top-level view of an early 3-way flat arbiter is shown in Fig. 3(right), and the result of application of this construction for $N = 3$ is shown in Fig. 4. There are three



inputs: $ra, rb, rc, ab, ba, bc, cb, ac, ca$; **outputs:** ga, gb, gc

Figure 5. STG specifications of an early 3-way flat arbiter with additional transitions eliminating the deadlocks.

clients, a, b and c, and $3(3-1)/2 = 3$ ME elements, resolving pairwise competitions between three concurrent requests, ra, rb and rc. Depending on the outcomes of these competitions, the ME elements raise one of their outputs, e.g. suppose ra wins both competitions it is participating in, which results in signals ab and ac becoming high. They enable the grant ga to be issued in response to ra, and the client a can use the resource. Having finished, this client lowers the request, which causes the ME elements resolving the competitions in which ra participated to lower concurrently their outputs. At this point the arbiter can reset the grant it issued; moreover, concurrently with this, another request (which had previously lost to ra) can now win the competition.

This basic implementation of flat arbiters is relatively simple and scales well with N. Unfortunately, this basic implementation can deadlock if the arbitration matrix $\mathcal{A}$ has a cycle, as will be demonstrated in the next section.

4. 3-way flat arbiter

Before considering the general case, we present a detailed flat arbiter design for the case $N = 3$; this will help us to illustrate some of the problems pertinent to flat arbiters.

The STG corresponding to the basic implementation of an early 3-way flat arbiter is shown in Fig. 5 (ignore the elements in the dashed boxes). This STG is relatively straightforward but, unfortunately, it can deadlock. Formal verification using the MPSAT tool reveals that there are exactly two deadlocked states which are reachable by the following traces:

$$ra^+, rb^+, rc^+, ab^+, bc^+, ca^+$$
$$ra^+, rb^+, rc^+, ba^+, cb^+, ac^+.$$

Intuitively, these two deadlocks correspond to the cycles in the arbitration matrix: in the first case, b loses to a, c loses to b, but a loses to c, and in the second case a loses to b, b loses to c, but c loses to a. The corresponding arbitration matrices are

$$\mathcal{A} = \begin{pmatrix} 0&1&0\\ 0&0&1\\ 1&0&0 \end{pmatrix} \quad \text{and} \quad \mathcal{A} = \begin{pmatrix} 0&0&1\\ 1&0&0\\ 0&1&0 \end{pmatrix} .$$

In these two cases there is no clear winner, and the STG deadlocks. (In general, the deadlocks precisely correspond to the non-trivial strongly connected subgraphs of the arbitration digraph.)

One can observe that in any of the deadlocked states it is possible to grant the victory to an arbitrary client; e.g. we chose to grant victory to a in the first case, and to b in the second case.[2] Thus, the STG is augmented by additional transitions shown in Fig. 5 in the dashed boxes.

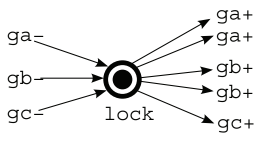
The behaviour of this augmented STG coincides with the behaviour of the original STG as long as the latter does not enter one of its deadlocked states. However, once one of the two such states is entered, the augmented STG, instead of deadlocking, enables one of the added transitions, ga^+ or gb^+, depending on which of the two states has been entered. (At the level of the state graph, deadlocked states have no outgoing arcs; hence augmenting the STG with additional transitions as shown in Fig. 5 provides an exit arc from each of the two deadlock states.)

Notice that the environment of the ME elements is more general than that of a 2-way arbiter, i.e. its extended behaviour shown in Fig. 2 is essential. Indeed, in the STG shown in Fig. 5 the trace

$$\boldsymbol{ra}^+\ rb^+\ \boldsymbol{rc}^+\ ab^+\ bc^+\ \boldsymbol{ca}^+\ ga^+\ \boldsymbol{ra}^-$$

can occur; however, its projection to the interface of the ME element with the inputs ra and rc (the highlighted transitions) is not allowed by a 2-way arbiter, but permitted by the specification of the ME element.

Fig. 5 provides a specification of an early arbiter. If necessary, it can be converted into a specification of the traditional arbiter by augmenting the STG with a new initially marked place $lock$, the arcs from it to all the transitions labelled ga^+, gb^+ and gc^+, and the arcs from all the transitions labelled ga^-, gb^- and gc^- to it, as illustrated in the picture below (note that the STG has two transitions labelled ga^+ and two transitions labelled gb^+). This new place does not allow to issue a grant until the previous grant has been lowered.



[2]We also explored the case when the victory is granted to a in both cases. This resulted in a circuit of the same size, but much less balanced. The other possible cases are easily seen to be symmetric to these two.

In spite of being in the preset of several output transitions, $lock$ does not cause violations of output-persistency, as the grant transitions cannot be enabled simultaneously (i.e. this is a *controlled choice*).

The STG in Fig. 5 as well as the version augmented with the $lock$ place satisfy all the properties listed in Sect. 2 and so can be synthesised as SI circuits. Note that though the request signals ra, rb and rc are declared as inputs, they are not actually needed for the final implementations.[3] In fact, the corresponding transitions can be contracted, yielding a weakly bisimilar STG (the bisimilarity follows from the fact that these transitions are not in the choice relation with any other transition, as can be easily seen from the STG). This transformation can be formally justified using the theory of *STG-bisimulation* developed in [19].

Various architectures are used to implement SI circuits [3]. Below we give the implementations in the following three architectures, which are probably the most well-known in SI design: the *complex-gate (CG)*, *generalised C-element (gC)*, and *standard-C (stdC)*.

CG implementation In the complex-gate implementation every local signal in the circuit is implemented as a single atomic gate, i.e. every gate is assumed to be an atomic evaluator of a Boolean function, with a delay element associated with its output [14]. Such an implementation can be described by the next-state functions of all the local signals. Note that if such a function depends on the implemented signal itself (i.e. the next-state value of the signal depends on its value in the current state) then a latch can often be used to implement this signal.

The CG implementation of the early flat arbiter is

$$\begin{aligned}
[ga] &= ab \cdot (\overline{gb} \cdot \overline{gc} \cdot bc \cdot ca + ac) + ga \cdot (ab + ac)\\
[gb] &= ba \cdot (\overline{ga} \cdot \overline{gc} \cdot ac \cdot cb + bc) + gb \cdot (ba + bc)\\
[gc] &= gc \cdot (ca + cb) + ca \cdot cb\ ,
\end{aligned}$$

and the implementation for the traditional protocol is

$$\begin{aligned}
[ga] &= \overline{gb} \cdot \overline{gc} \cdot ab \cdot (bc \cdot ca + ac) + ga \cdot (ab + ac)\\
[gb] &= \overline{ga} \cdot \overline{gc} \cdot ba \cdot (ac \cdot cb + bc) + gb \cdot (ba + bc)\\
[gc] &= \overline{ga} \cdot \overline{gb} \cdot ca \cdot cb + gc \cdot (ca + cb)
\end{aligned}$$

(these implementations can be automatically obtained from the corresponding STGs using either PETRIFY or MPSAT).

Note that the next-state function for gc in the early flat arbiter can be mapped to a single C-element. Unfortunately, the complex-gate implementations of the other two output signals are rather complicated, and it would be difficult to satisfy the atomicity assumption for such gates in practice. However, these complex gates can be decomposed in an SI way using PETRIFY, as shown in Fig. 6.

gC implementation In gC implementations [13] each signal is implemented using a pseudo-static latch called *generalised C element (gC-element)* which is assumed to

[3]Fig. 3 does not allow the decision circuitry to use the request signals; moreover, even if it did, using such signals turns out to have no effect on the size of the circuit.

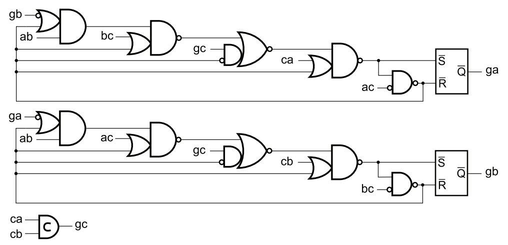


Figure 6. The decomposed CG implementation of an early 3-way flat arbiter.

be atomic. A gC implementation is specified by the *set* and *reset* functions for each local signal. In the states where the set function evaluates to 1, the next-state function for the signal evaluates to 1; in the states where the reset function evaluates to 1, the next-state function for the signal evaluates to 0; and in the states where both set and reset functions evaluate to 0 the signal should keep its current value (it is an error if in some reachable state both set and reset functions evaluate to 1 — this can lead to a short circuit).

The set and reset functions for the gC implementation of the early flat arbiter are

$$\begin{aligned}
[ga\uparrow] &= ab\cdot(\overline{gb}\cdot\overline{gc}\cdot bc\cdot ca+ac) & [ga\downarrow] &= \overline{ab}\cdot\overline{ac}\\
[gb\uparrow] &= ba\cdot(\overline{ga}\cdot\overline{gc}\cdot ac\cdot cb+bc) & [gb\downarrow] &= \overline{ba}\cdot\overline{bc}\\
[gc\uparrow] &= ca\cdot cb & [gc\downarrow] &= \overline{ca}\cdot\overline{cb},
\end{aligned}$$

and the functions for the flat arbiter implementing the traditional protocol are

$$\begin{aligned}
[ga\uparrow] &= \overline{gb}\cdot\overline{gc}\cdot ab\cdot(bc\cdot ca+ac) & [ga\downarrow] &= \overline{ab}\cdot\overline{ac}\\
[gb\uparrow] &= \overline{ga}\cdot\overline{gc}\cdot ba\cdot(ac\cdot cb+bc) & [gb\downarrow] &= \overline{ba}\cdot\overline{bc}\\
[gc\uparrow] &= \overline{ga}\cdot\overline{gb}\cdot ca\cdot cb & [gc\downarrow] &= \overline{ca}\cdot\overline{cb}
\end{aligned}$$

(these functions can be automatically obtained from the corresponding STGs using either PETRIFY or MPSAT).

The above gC implementation of the early flat arbiter is shown in Fig. 7, where the set and reset functions are mapped to the pull-down and pull-up networks of transistors, respectively (note that the output is inverted). When both networks are off, the 'keeper' composed of the two inverters (with the feedback inverter being weak so that it can be overpowered by the pull-up and pull-down networks) is used to preserve the current output value of the signal.

Note that the transistor stacks in this implementation are quite tall, e.g. the pull-down networks for ga and gb have five transistors in sequence. However, this is alleviated by putting the transistors controlled by signals $\overline{gb}$ and $\overline{gc}$ (resp. $\overline{ga}$ and $\overline{gc}$) in the implementation of ga (resp. gb) close to the ground. According to MPSAT, these signals never trigger the change in the values of the corresponding set functions, and so they always arrive earlier than the other signals controlling these gC elements. Hence the corresponding transistors fire early, and in practice the actual latency of the pull-down network is at most three transistor delays.

stdC implementation In stdC implementations [1] each signal is implemented using a C-latch (with one inverted

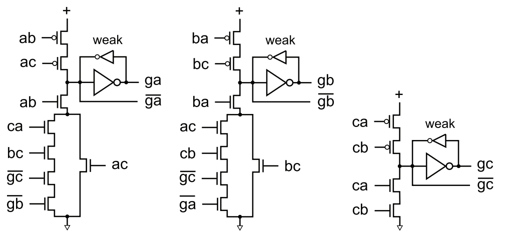


Figure 7. The gC implementation of an early 3-way flat arbiter.

input) controlled by set and reset signals, which we here assume are implemented as complex-gates. This architecture is superficially similar to the previous one, but one should bear in mind that a gC element is assumed to be atomic, while in the stdC implementation the gates controlling a C-latch have delays. Hence a naïve transformation of a gC implementation into an stdC one can result in a hazardous circuit. To avoid hazards, the derived set and reset functions must satisfy the *Monotonic Cover* condition [1, 3].

The stdC implementations of the flat arbiter for the early and traditional protocols differs from the corresponding gC implementations only in the set functions of ga and gb:

$$\begin{array}{ll} [ga\uparrow] = ab \cdot (\overline{ga} \cdot \overline{gb} \cdot \overline{gc} \cdot bc \cdot ca + ac) & \text{(early)} \\ [gb\uparrow] = ba \cdot (\overline{ga} \cdot \overline{gb} \cdot \overline{gc} \cdot ac \cdot cb + bc) & \text{(early)} \\ [ga\uparrow] = \overline{gb} \cdot \overline{gc} \cdot ab \cdot (\overline{ga} \cdot bc \cdot ca + ac) & \text{(traditional)} \\ [gb\uparrow] = \overline{ga} \cdot \overline{gc} \cdot ba \cdot (\overline{gb} \cdot ac \cdot cb + bc) & \text{(traditional)} \end{array}$$

(these functions can be automatically obtained from the corresponding STGs using either PETRIFY or MPSAT).

Similarly to gC architecture, stdC architecture allows for some signals to be implemented as complex gates, in particular signal gc in the early arbiter can still be implemented by a C-element.

One can see that the obtained monotonic functions turn out to be just slightly more complicated compared with the gC implementation. However, the advantage of this stdC implementation is that the atomicity assumption has to be ensured for smaller gates.

5. General case

The previous section presented a solution for the 3-way case, where there were only two possible arbitration matrices containing a cycle, and the presented solution explicitly detected and resolved these two deadlock situations. Unfortunately, the number of such deadlock situations grows exponentially with N: there are already 40 different stable arbitration matrices containing a cycle for $N = 4$. Therefore, the explicit resolution of every individual deadlock is infeasible in general case. This section presents a construction for speed-independent N-way early flat arbiters whose size is polynomial in N.

The idea behind the presented construction is, given an arbitration matrix $\mathcal{A}$ (which can have cycles), to compute an acyclic arbitration matrix $\mathcal{B}$, which is obtained from $\mathcal{A}$ by negating certain entries, i.e. changing the outcomes of certain arbitrations. Below we define $\mathcal{B}$ formally.

A request r_k is called *dominated* (denoted dom_k) if $\mathcal{A}_{k'k} = 1$ for some request $r_{k'}$ such that $k' < k$ (in other words, r_k has lost an arbitration with some lexicographically preceding request $r_{k'}$). Similarly, request r_k is called *non-dominated* (denoted $ndom_k$) if $\mathcal{A}_{kk'} = 1$ for all requests $r_{k'}$ such that $k' < k$ (in other words, r_k has won all the arbitrations with the lexicographically preceding requests). Formally,

$$dom_k = \bigvee_{k'<k} \mathcal{A}_{k'k} \quad \text{and} \quad ndom_k = \bigwedge_{k'<k} \mathcal{A}_{kk'} .$$

Note that it is possible that $dom_k = ndom_k = 0$ (this would indicate that some of the arbitrations in which r_k participates have not completed yet), but one can easily show that the situation $dom_k = ndom_k = 1$ is impossible.

Proposition 1 (Properties of dom and $ndom$)**.**

1. *If $\mathcal{A}$ is stable w.r.t. r_k then $dom_k + ndom_k = 1$.*
2. *If $\mathcal{A}$ is stable and at least one of the requests $r_1, \ldots, r_k$ is observable in $\mathcal{A}$ then $dom_k + ndom_k = 1$.*

Proof. For the sake of contradiction, suppose that $dom_k = 0$ and $ndom_k = 0$. By the former equality, $\mathcal{A}_{k'k} = 0$ for all $k' < k$ (*).

To prove the first claim, note that due to stability of $\mathcal{A}$ w.r.t. r_k, $\widehat{\mathcal{A}}_{k'k} = 1$ for all $k' \neq k$. Together with (*) this means that $\mathcal{A}_{kk'} = 1$ for all $k' < k$, and so $ndom_k = \bigwedge_{k'<k} \mathcal{A}_{kk'} = 1$, a contradiction. Thus the claim holds.

To prove the second claim, we argue that r_k is observable in $\mathcal{A}$ (**). At least one of the requests $r_1, \ldots, r_k$ is observable in $\mathcal{A}$, say, $r_{k''}$, $k'' \leq k$. If $k'' = k$ then (**) trivially holds. Otherwise $k'' < k$, and by (*) $\mathcal{A}_{k''k} = 0$. Since $\mathcal{A}$ is stable and $r_{k''}$ is observable in $\mathcal{A}$, $\mathcal{A}_{kk''} = 1$, which means that (**) holds.

Due to (**) and the stability of $\mathcal{A}$, $\mathcal{A}$ is stable w.r.t. r_k, and so the second claim follows from the first one. □

$\mathcal{B}$ is then defined by assuming that a dominated request r_k surrenders all its arbitration wins (if any) over the lexicographically preceding requests $r_{k'}$, $k' < k$:

$$\mathcal{B}_{ij} = \begin{cases} \mathcal{A}_{ij} + \mathcal{A}_{ji} \cdot dom_j & \text{if } i \leq j \\ \mathcal{A}_{ij} \cdot ndom_i & \text{if } i > j. \end{cases} \qquad (1)$$

Intuitively, if $i \leq j$ (i.e. r_i lexicographically precedes r_j) then $\mathcal{B}_{ij} = 1$ iff r_i won the arbitration with r_j (i.e. $\mathcal{A}_{ij} = 1$) or r_i lost the arbitration with r_j (i.e. $\mathcal{A}_{ji} = 1$) but r_j is dominated; if $i > j$ (i.e. r_i lexicographically succeeds r_j) then $\mathcal{B}_{ij} = 1$ iff r_i won the arbitration with r_j and is non-dominated.

Remark 2. *One might be tempted to simplify the definition of $\mathcal{B}_{ij}$ to $\mathcal{A}_{ij} + dom_j$ in the case $i \leq j$. Unfortunately, this will invalidate some essential properties needed for the flat arbiter construction. In particular, Th. 8(4) below, which is required for output-persistency, depends on Prop. 7, whose proof fails for this simplified notion of $\mathcal{B}$. For example, in a 3-way flat arbiter the situation $ab = bc = 1$ would lead* *to domination of rc by rb, and ga can be issued before the arbitration between ra and rc completes. Then the environment can reset ra, disabling the output of the ME element arbitrating between ra and rc, which results in a glitch.*

We now investigate some properties of $\mathcal{B}$. When doing so, we will often use the following observation:

$$\widehat{\mathcal{B}}_{ij} = \mathcal{A}_{ij} + \mathcal{A}_{ji} \cdot (dom_j + ndom_j) \quad \text{if } i \leq j. \qquad (2)$$

The first property shows that $\mathcal{B}$ is an arbitration matrix.

Proposition 3 (Consistency)**.** *$\mathcal{B}_{ij} = 0$ or $\mathcal{B}_{ji} = 0$ for any i, j (in particular, $\mathcal{B}_{ii} = 0$ for all i).*

Proof. W.l.o.g., suppose $i < j$ and, for the sake of contradiction, suppose that $\mathcal{B}_{ij} = \mathcal{B}_{ji} = 1$. Then $\mathcal{A}_{ij} + \mathcal{A}_{ji} \cdot dom_j = 1$ and $\mathcal{A}_{ji} \cdot ndom_j = 1$ by (1), and thus $\mathcal{A}_{ji} = 1$ by the latter equality and $\mathcal{A}_{ij} = 0$ due to the consistency of $\mathcal{A}$. Therefore $dom_j = 1$ by the former equality, which leads to a contradiction, since $ndom_j = 1$ by the latter equality, and the situation $dom_j = ndom_j = 1$ cannot happen. □

Now we show that $\mathcal{B}$ can be obtained from a stable $\mathcal{A}$ by changing the outcomes of some of the pairwise arbitrations.

Proposition 4 (Relationship to $\mathcal{A}$)**.**

1. *if $\mathcal{A}_{ij} = \mathcal{A}_{ji} = 0$ then $\mathcal{B}_{ij} = \mathcal{B}_{ji} = 0$.*
2. *if $\mathcal{A}$ is stable and $\widehat{\mathcal{A}}_{ij} = 1$ then $\widehat{\mathcal{B}}_{ij} = 1$.*

Proof. The former claim trivially follows from (1). To prove the latter claim, w.l.o.g., we assume that $i < j$. If $\mathcal{A}_{ij} = 1$ then the claim obviously holds due to (2). Otherwise $\mathcal{A}_{ji} = 1$, and so r_j is observable in $\mathcal{A}$ and $\widehat{\mathcal{B}}_{ij} = dom_j + ndom_j$ by (2). Thus the property holds by Prop. 1(2). □

Corollary 5 (Relationship to $\mathcal{A}$)**.** *If $\mathcal{A}$ is stable then:*

1. *$\widehat{\mathcal{A}} = \widehat{\mathcal{B}}$ (in particular, $\mathcal{A} = \mathbf{0}$ iff $\mathcal{B} = \mathbf{0}$).*
2. *$\|\mathcal{A}\| = \|\mathcal{B}\|$, where $\|\cdot\|$ denotes the number of 1s in a matrix.*

Proposition 6 (Acyclicity)**.** *$\mathcal{B}$ is acyclic.*

Proof. For the sake of contradiction, suppose that the arbitration digraph corresponding to $\mathcal{B}$ has a cycle $r_{i_1} \to r_{i_2} \to \ldots \to r_{i_m} \to r_{i_1}$. Let k be the largest index among $i_1, \ldots, i_m$, and $r_{k'}$ and $r_{k''}$ be the vertices immediately preceding and immediately succeeding r_k in this cycle: $\ldots r_{k'} \to r_k \to r_{k''} \ldots$

Since $\mathcal{B}_{kk''} = 1$, $\mathcal{A}_{kk''} \cdot ndom_k = 1$ by (1). Therefore, $ndom_k = 1$, and so $dom_k = 0$, as dom_k and $ndom_k$ cannot be 1 simultaneously. Since $\mathcal{B}_{k'k} = 1$, $\mathcal{A}_{k'k} + \mathcal{A}_{kk'} \cdot dom_k = 1$ by (1), which, due to $dom_k = 0$, implies that $\mathcal{A}_{k'k} = 1$. But the latter implies that $dom_k = 1$ according to the definition of dom, a contradiction. □

Proposition 7 (Stability)**.** *If $\mathcal{B}$ is stable w.r.t. r_i then $\mathcal{A}$ is stable w.r.t. r_i.*

Proof. We need to show that $\widehat{\mathcal{A}}_{ij} = 1$ for all $j \neq i$. W.l.o.g., we assume that $i < j$. Then

$$\widehat{\mathcal{A}}_{ij} = \mathcal{A}_{ij} + \mathcal{A}_{ji} \geq \mathcal{A}_{ij} + \mathcal{A}_{ji} \cdot (dom_j + ndom_j) = \widehat{\mathcal{B}}_{ij} = 1,$$

where the last two equalities follow from (2) and the fact that $\mathcal{B}$ is stable w.r.t. r_i, respectively. □

The following result is crucial for the flat arbiter design. Basically, it tells that whenever $\mathcal{A} \neq \mathbf{0}$ is stable, $\mathcal{B}$ has a unique *winner,* i.e. a request r_i such that $\mathcal{B}_{ij} = 1$ for all $j \neq i$, and this winner is among the requests that have actually arrived, i.e. it is observable in $\mathcal{A}$. Note that such a winner exists even if $\mathcal{B}$ is unstable. The last part of the proposition states that if r_i is a winner then all the arbitrations in which it participates are completed. This ensures that once the grant g_i is issued, the subsequent lowering of r_i by the environment will not disable the outputs of any of the ME elements handling these arbitrations, which would result in a glitch (see also Rem. 2). Due to this result, one can essentially re-use the basic solution from Sect. 3 in the decision logic for N-way flat arbiters, provided that matrix $\mathcal{A}$ is substituted with matrix $\mathcal{B}$.

Theorem 8 (Existence of a winner)**.**

1. *$\mathcal{B}$ has at most one winner.*
2. *If $\mathcal{A} \neq \mathbf{0}$ is stable then $\mathcal{B}$ has a winner.*
3. *If $\mathcal{B}$ has a winner then it is observable in $\mathcal{A}$.*
4. *If $\mathcal{B}$ has a winner then $\mathcal{A}$ is stable w.r.t. it.*

Proof.

Claim 1 For the sake of contradiction, suppose there are two different winners, r_i and r_j. Then $\mathcal{B}_{ij} = \mathcal{B}_{ji} = 1$ by the definition of a winner, which contradicts the consistency of $\mathcal{B}$ (see Prop. 3).

Claim 2 Let r_k be the lexicographically smallest request such that all its lexicographical successors are dominated. (Such a request always exists, in particular $k = N$ if no request is dominated.) Then r_k is not dominated (i.e. $dom_k = 0$), since otherwise r_{k-1} would be a lexicographically smaller request satisfying the condition (observe that r_1 can never be dominated).

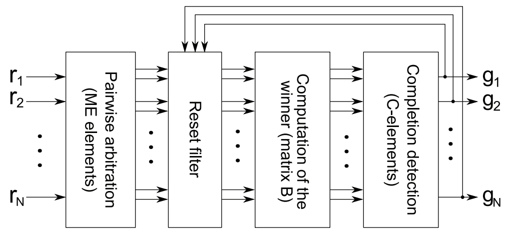
We now argue that r_k is observable in $\mathcal{A}$ (*). Let r_i be the lexicographically smallest request observable in $\mathcal{A}$. Note that such a request exists due to $\mathcal{A} \neq \mathbf{0}$, and $i \leq k$ since all the lexicographical successors of r_k are dominated. If $i = k$ then (*) trivially holds, and in the case $i < k$, $\widehat{\mathcal{A}}_{ik} = 1$ due to the stability of $\mathcal{A}$, and $\mathcal{A}_{ik} = 0$ since otherwise r_k would be dominated. Hence $\mathcal{A}_{ki} = 1$, i.e. (*) holds.

We now show that $\mathcal{B}_{kk'} = 1$ for all $k' \neq k$, i.e. r_k is a winner. Due to the stability of $\mathcal{A}$ and (*), $\widehat{\mathcal{A}}_{k'k} = 1$.

$k' < k$ $\mathcal{A}_{k'k} = 0$ since otherwise r_k would be dominated. Hence $\mathcal{A}_{kk'} = 1$, and so $\mathcal{B}_{kk'} = \mathcal{A}_{kk'} \cdot ndom_k = ndom_k$. Due to (*), Prop. 1(2) can be applied, yielding $dom_k + ndom_k = 1$. Due to $dom_k = 0$, $ndom_k = 1$, and so $\mathcal{B}_{kk'} = 1$.

$k' > k$ $\mathcal{B}_{kk'} = \mathcal{A}_{kk'} + \mathcal{A}_{k'k} \cdot dom_{k'} = \mathcal{A}_{kk'} + \mathcal{A}_{k'k} = \widehat{\mathcal{A}}_{k'k} = 1$, since $dom_{k'} = 1$ due to the choice of k.

Claim 3 Let r_k be the winner. If $k \neq 1$ then $\mathcal{B}_{k1} = \mathcal{A}_{k1} \cdot ndom_k = 1$, and so $\mathcal{A}_{k1} = 1$, i.e. r_k is observable in $\mathcal{A}$. If $k = 1$ then $\mathcal{B}_{12} = \mathcal{A}_{12} + \mathcal{A}_{21} \cdot dom_1 = 1$, and due to $dom_1 = 0$ by the definition of dom, $\mathcal{A}_{12} = 1$, i.e. r_k is observable in $\mathcal{A}$.



r_1
r_2
r_N
Pairwise arbitration (ME elements)
Reset filter
Computation of the winner (matrix B)
Completion detection (C-elements)
g_1
g_2
g_N

Figure 8. Top-level view of N-way flat arbiter.

Claim 4 $\mathcal{B}$ is stable w.r.t. the winner, so the result follows from Prop. 7. □

The general top-level view of an N-way flat arbiter built using the presented idea is shown in Fig. 8. The reset filter is transparent for the incoming arbitration results ($\mathcal{A}_{jk} = 1$ or $\mathcal{A}_{kj} = 1$) only if the decision logic associated with the corresponding grants g_j and g_k has been completely reset after a previously issued grant (this fact is acknowledged by both grants being low).

Fig. 9(left) shows an STG specification of the filter element which propagates the arbitration result $\mathcal{A}_{ij}$ (i.e. the output of the ME element responsible for $\mathcal{A}_{ij}$ entry of arbitration matrix $\mathcal{A}$) to the decision circuitry (signal $\mathcal{F}_{ij}$), depending on the states of the corresponding grants g_i and g_j. Fig. 9(right) shows a decomposed implementation derived by PETRIFY from this STG. Note that the inverters can be shared by the filter elements, i.e. there are N inverters altogether (one per grant signal). Alternatively, one can use the gC implementation of the filter element:

$$[\mathcal{F}_{ij}\uparrow] = \mathcal{A}_{ij} \cdot \overline{g_i} \cdot \overline{g_j} \qquad [\mathcal{F}_{ij}\downarrow] = \overline{\mathcal{A}_{ij}}.$$

The filter elements can be simplified depending on their position. We call an arc $\mathcal{A}_{ij}$ of the arbitration digraph *irreversible* iff either $i < j$ or $i = 2$ and $j = 1$; intuitively, such arcs cannot be reversed in $\mathcal{B}$. The other arcs are called *reversible.* For an irreversible arc $\mathcal{A}_{ij}$ the filter can be simplified to

$$[\mathcal{F}_{ij}\uparrow] = \mathcal{A}_{ij} \cdot \overline{g_j} \qquad [\mathcal{F}_{ij}\downarrow] = \overline{\mathcal{A}_{ij}} + g_j,$$

i.e. to an AND2 gate with an input inverter. This is due to the fact that $\mathcal{A}_{ij}^-$ cannot be preceded by g_j^+. Hence $\mathcal{F}_{ij}$ cannot be prematurely reset by g_j^+, since the irreversibility of the arc guarantees that r_j is dominated and g_j cannot be issued before $\mathcal{A}_{ij}^-$ (g_i will always be issued first).

If $\mathcal{A}_{ij}$ is reversible, the filter can be simplified to

$$[\mathcal{F}_{ij}\uparrow] = \mathcal{A}_{ij} \cdot \overline{g_j} \qquad [\mathcal{F}_{ij}\downarrow] = \overline{\mathcal{A}_{ij}},$$

(which can be implemented by an RS latch). This implementation also does not need to monitor g_i, but for a different reason.[4] If g_i is high when $\mathcal{A}_{ij}$ is low then $\mathcal{A}_{ji}$ must be high, and thus $\mathcal{A}_{ij}$ cannot rise. Hence if $\mathcal{A}_{ij}$ rises then g_i must be low, because g_i could not be issued without $\mathcal{A}_{ij}$ or $\mathcal{A}_{ji}$ being high due to Th. 8(4).

[4]The previous argument is no longer applicable: g_j^+ can happen before $\mathcal{A}_{ij}^-$, because the arc can be reversed in $\mathcal{B}$ and g_j can be issued before g_i.

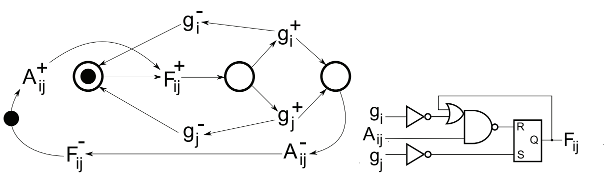


Figure 9. An STG specification and a complex gate implementation of the reset logic element for signal $\mathcal{A}_{jk}$.

The proposed generic approach was applied to synthesis of 3-way and 4-way flat arbiters. The decomposed implementation of the 3-way flat arbiter is shown in Fig. 10. In spite of having been derived using a general method, this solution is just slightly bigger than the decomposed implementation derived by PETRIFY using STG synthesis (see Fig. 6). Note that all the gates have only 2 or 3 inputs, and thus the circuit is implementable in most gate libraries. (We also verified that each of the 3-input C-elements can be decomposed into two 2-input ones.) A gC implementation of a 4-way flat arbiter is given below. (We omitted the filters; the signals in the right hand side of these equations are the filter outputs, cf. Fig. 8).

$$\begin{aligned}
[ga\uparrow] &= \mathcal{F}_{ab}\cdot(\mathcal{F}_{ac}+\mathcal{F}_{ca}\cdot\mathcal{F}_{bc})\cdot(\mathcal{F}_{ad}+\mathcal{F}_{da}\cdot(\mathcal{F}_{bd}+\mathcal{F}_{cd}))\\
[ga\downarrow] &= \overline{\mathcal{F}_{ab}}\cdot\overline{\mathcal{F}_{ac}}\cdot\overline{\mathcal{F}_{ad}}\\
[gb\uparrow] &= \mathcal{F}_{ba}\cdot(\mathcal{F}_{bc}+\mathcal{F}_{cb}\cdot\mathcal{F}_{ac})\cdot(\mathcal{F}_{bd}+\mathcal{F}_{db}\cdot(\mathcal{F}_{ad}+\mathcal{F}_{cd}))\\
[gb\downarrow] &= \overline{\mathcal{F}_{ba}}\cdot\overline{\mathcal{F}_{bc}}\cdot\overline{\mathcal{F}_{bd}}\\
[gc\uparrow] &= \mathcal{F}_{ca}\cdot\mathcal{F}_{cb}\cdot(\mathcal{F}_{cd}+\mathcal{F}_{dc}\cdot(\mathcal{F}_{ad}+\mathcal{F}_{bd}))\\
[gc\downarrow] &= \overline{\mathcal{F}_{ca}}\cdot\overline{\mathcal{F}_{cb}}\cdot\overline{\mathcal{F}_{cd}}\\
[gd\uparrow] &= \mathcal{F}_{da}\cdot\mathcal{F}_{db}\cdot\mathcal{F}_{dc}\\
[gd\downarrow] &= \overline{\mathcal{F}_{da}}\cdot\overline{\mathcal{F}_{db}}\cdot\overline{\mathcal{F}_{dc}}
\end{aligned}$$

Note that the stacks in this gC implementation are at most 5 transistors tall, so this solution is comparable with that in Fig. 7 in this respect. In general, these stacks will be $1 + 2(N - 2) = 2N - 3$ transistors tall in the set networks and $N - 1$ transistors tall in the reset network for the case of N-way flat arbiter.

Using the WORKCRAFT framework [15] and the MPSAT tool [9] we have formally verified that these implementations satisfy the following properties:

Output-persistency In particular, the outputs of the ME elements cannot be disabled, cf. Th. 8(4).

Computational non-interference Whenever the environment sends an input, the circuit is ready to receive it, and vice versa, whenever the circuit produces an output, the environment is ready to receive it, see [5].

Deadlock freeness The definition of a deadlock in an arbiter is slightly different from the standard deadlock, as in the situation when only some requests have arrived, the arbiter is obliged to eventually issue a grant, *even when the remaining requests never arrive.* Hence, if some state (except the initial one) does not enable any transitions besides

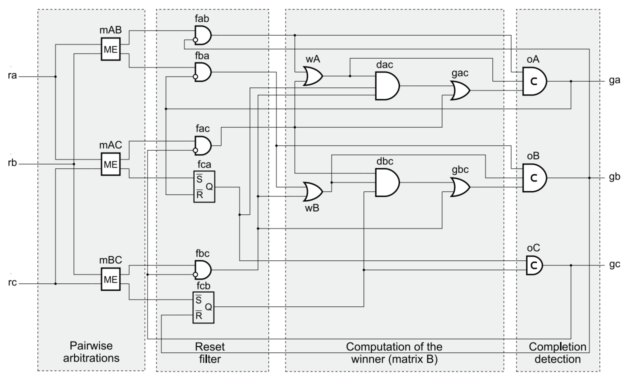


Figure 10. A decomposed implementation of a 3-way flat arbiter.

the rising requests then it is classified as a deadlock. Another way of putting it is that in the STG the rising request transitions are not *weakly fair,* i.e. they may remain enabled forever, without firing. (See [9] for more detail.)

Mutual exclusion Since these arbiters implement the early protocol, one cannot just require that at most one grant is high at any time; in fact, the STG in Fig. 1(right) has reachable states in which all the grants are high. Instead, one should check that at any time there is at most one client for which both the request and the grant are high. (See [9] for more detail.)

6. Conclusions

In this paper we exploit the use of global information about pairwise arbitrations (effectively, a full snapshot of the order relation), which opens up opportunities for developing new arbitration structures, as opposed to standard arbiters built around local order information as in trees or rings. In particular, we have proposed a new way of constructing N-way arbiters. The advantage of this construction is that all the ME elements work in parallel (as opposed to sequential arbitration in the traditional arbiter types), which helps to decrease the latency of the arbiter. Moreover, arbiters with other decision policies (e.g. when up to $m < N$ requests can be granted) can be designed using this framework. An interesting peculiarity of flat arbiters is that they use ME elements in a non-standard way (see Fig. 2 and the related explanation in Sect. 1).

We give detailed and practical designs for a 3-way arbiter, and also show a theoretical construction for general N-way arbiters. In doing so, we proposed a general solution to the problem of cycles in the arbitration matrix. Of course, such a construction quickly becomes impractical due to the growth in size and latency of the decision logic. However, N-way arbiters with small values of N are most important in practice. If an N-way arbiter for a large N is needed, one can combine the traditional and flat arbiters, e.g. by building a tree of 3- or 4-way flat arbiters.

Acknowledgements This research was supported by the EPSRC grants EP/D053064/1 (SEDATE), EP/C512812/1 (NEGUS) and EP/E044662/1 (STEP), and Royal Academy of Engineering / EPSRC post-doctoral research fellowship EP/C53400X/1 (DAVAC).

References

[1] P. Beerel, C. Myers, and T.-Y. Meng. Covering conditions and algorithms for the synthesis of speed-independent circuits. *IEEE Trans. on CAD*, 1998.

[2] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, Lab. for Comp. Sci., MIT, 1987.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002.

[4] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT Press, 1988.

[5] J. Ebergen. Arbiters: an exercise in specifying and decomposing asynchronously communicating components. *Sci. of Comp. Prog.*, 18:223–245, 1992.

[6] International Technology Roadmap for Semiconductors: Design, 2007. URL: `http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf`.

[7] M. Josephs. Gate-level modelling and verification of asynchronous circuits using CSPM and FDR. In *Proc. ASYNC'07*, pages 83–94. IEEE Comp. Soc. Press, 2007.

[8] M. Josephs and J. Yantchev. CMOS design of the tree arbiter element. *IEEE Trans. on VLSI*, 4(4):472–476, 1996.

[9] V. Khomenko. A usable reachability analyser. Technical Report CS-TR-1140, School of Comp. Sci., Newcastle Univ., 2009.

[10] V. Khomenko, M. Koutny, and A. Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental SAT. *Fund. Inf.*, 70(1–2):49–73, 2006.

[11] D. Kinniment. *Synchronization and Arbitration in Digital Systems*. John Wiley & Sons Ltd, 2007.

[12] A. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Proc. 6^{th} MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[13] A. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[14] D. Muller and W. Bartky. A theory of asynchronous circuits. In *Proc. Int. Symp. of the Theory of Switching*, pages 204–243, 1959.

[15] I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov, and A. Yakovlev. Automated verification of asynchronous circuits using circuit Petri nets. In *Proc. ASYNC'08*, pages 161–170. IEEE Comp. Soc. Press, 2008.

[16] L. Rosenblum and A. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proc. Int. Workshop on Timed Petri Nets*, pages 199–206. IEEE Comp. Soc. Press, 1985.

[17] C. Seitz. Ideas about arbiters. *Lambda*, 1:10–14, 1980.

[18] V. Varshavsky, editor. *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*. Kluwer Academic Publishers, 1990. Translated from Russian, published by Nauka, Moscow, 1986.

[19] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella, A. Yakovlev, and G. Rozenberg, editors, *Concurrency and Hardware Design*, volume 2549 of *LNCS*, pages 152–190. Springer-Verlag, 2002.