School of Electrical, Electronic & Computer Engineering

Multi-resource arbiter decomposition

S. Golubcovs, D. Shang, F. Xia, A. Mokhov, A. Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2009-143

February 2009

Contact:

Stanislavs.Golubcovs@ncl.ac.uk

Delong.Shang@ncl.ac.uk

Fei.Xia@ncl.ac.uk

Andrey.Mokhov@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant GR/E044662/1

NCL-EECE-MSD-TR-2009-143

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

http://async.org.uk/

Multi-resource arbiter decomposition

S. Golubcovs, D. Shang, F. Xia, A. Mokhov, A. Yakovlev

February 2009

Abstract

This paper describes variations of multi-resource arbiter decomposition into tiled layout. The tile is a fixed block of basic gates that has its own input and output connections providing certain functionality.

1 Introduction

1.1 Simple 2×2 arbiter

An arbiter with active resources assumes additional resource requests that may arrive any time and are intended to report when the resource is available. An example of simple arbiter with 2 clients and 2 resources is presented in Figure 1. Any of two clients may request a resource and any of the resources may offer their service. Once an arbiter finds a pair, it initiates one of the handshakes ($H11$, $H12$, $H21$, $H22$).

The arbiter is designed in a way, that a resource busy with one of the clients is not offered to the other one. It means, that handshake $H11$ (first client connected with the first resource) cannot be activated together with $H21$. Similarly, $H11$ is in conflict with $H12$, because a client can only be connected with one resource at a time. However, $H12$ and $H21$ or $H11$ and $H22$ are not in conflict because each participant has not more than one connection established. To achieve full advantage of the active resource arbiter, it needs to support such concurrent handshakes.

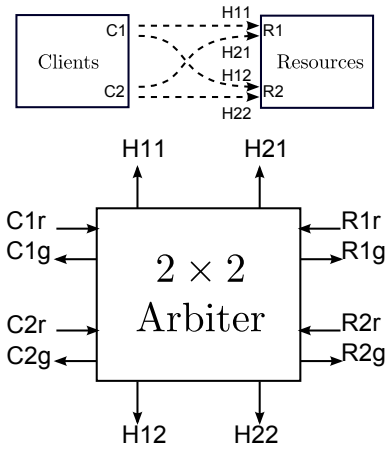


Figure 1: Simple 2×2 arbiter

1.2 The 2×2 arbiter implementation

Figure 2 shows one possible implementation of the 2×2 arbiter. It uses two mutual exclusion elements and several C-gates. As the signal request propagates through the MUTEX-es, it activates one of the internal handshakes ($h11$, $h12$, $h21$, $h22$). The internal handshakes activate main outgoing handshakes $H11$, $H12$, $H21$, $H22$ and disable their conflicting neighbours. For instance, $h11$ disables $h21$ and $h12$. Because of the ME elements, only one pair of requests can be initiated at a time. When all four requests arrive, the arbiter first matches one pair producing the first handshake and then masks the initial requests to allow the next non-conflicting pair to be matched. Since both ME elements may be freed at different times, there is a possibility that for a short interval there will be present old request from one side and the new request from the other side, which would always pair in one of the conflicting handshakes. This race condition between old and new grant signals on gc_i and gr_j (where i is the number of the requested column and j is the number of requested row) is eliminated by the disabling signal coming from one of the internal handshakes, which disables wrong pairings and ensures circuit speed independence.

In this report we are going to consider a number of ways to scale such design.

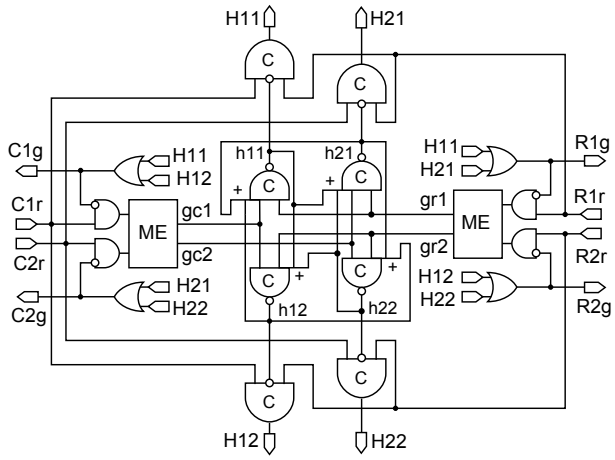
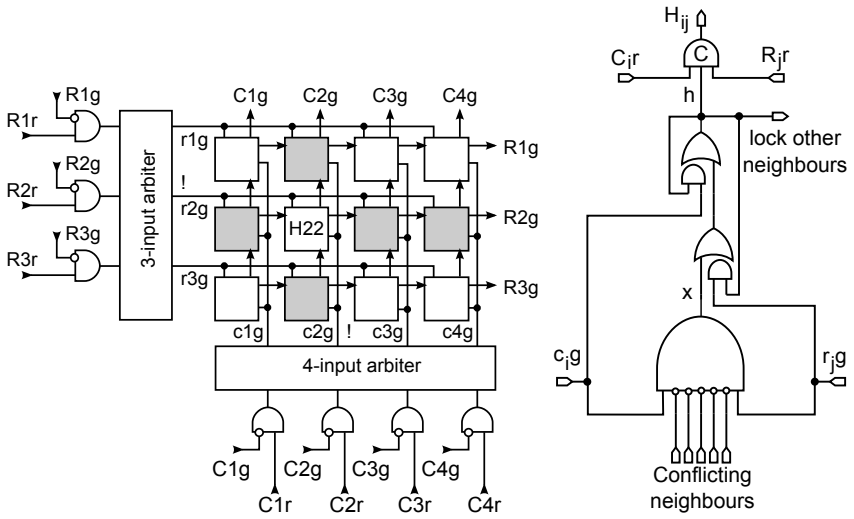


Figure 2: 2×2implementation

2 Extending up to $N \times M$ arbiter

In general problem statement, the arbiter may need to support N clients and M resources. We may lay it out as a rectangular grid of tiles implementing the functionality of C-elements in the original design in Figure 2. An example of such layout for 4 clients and 3 resources is given in Figure 3a. The internal resource grants $r1g$, $r2g$, $r3g$ and client grants $c1g$, $c2g$, $c3g$, $c4g$ form three rows and four columns of the grid. Similarly to the 2×2 arbiter all conflicting neighbours need to be disabled before the initial requests are masked to let a new requests propagate into it. In particular, the handshake row needs to be disabled before the new client request and the handshake column before the new resource request. As it can be seen from the figure, there are $N + M - 2$ conflicting neighbours for each tile.

If we use asymmetric C-elements to solve the general problem, the set phase of the gates would have inputs from all conflicting neighbours. The speed-independent decompositions of



(a) Tiles forming the circuit (b) Implementation of a tile

Figure 3: Schematics for the implementation of 4×3 arbiter.

the C-elements would still have gates with growing number of inputs (Fig. 3b). In practise, such circuits are not scalable.

One of way to solve this problem is by constructing signal x by smaller gates and introducing delay on the signal of H_{ij}. That would provide us with a scalable implementation in tiles with the timing assumption, that the signal locking the other neighbours is managing on time, before the outgoing handshake masks the x for all conflicting neighbours. In the following part of the paper, we will discuss the alternative SI implementations for the tiles so that the complexity of the gates used is reduced or not growing with the number of inputs.

2.1 Ring-based approach

We can try to model the behaviour of a tile, bearing in mind additional requirement that the outgoing handshake signal happens only after all the neighbouring tiles have been disabled. The block signal can propagate in both (column and row) directions using the ring architecture. When the outgoing block signal returns as the input on the opposite side of the tile, it signals that all the neighbouring tiles have been disabled. Together that would form a torus network of tiles blocking their neighbours. The behaviour of such tile is depicted in Figure 4

To initiate the handshake ($h+$), the tile needs to receive the grant signals $cg+$ and $rg+$ coming from N- and M-way arbiters. When both $cg+$ and $rg+$ arrive, tile knows it was chosen for the handshake and initiates its internal variable $x+$. After that, before the handshake signal is activated, the tile blocks its conflicting neighbours by $bro+$ and $bro+$. The signals bro and bco are outgoing block signals for the row and the column. Their appearance is followed by the incoming block signals bri and bci when the block (or unblock) signal propagates through all conflicting tiles. When it happens, the tile is at state when it can issue the internal handshake $h+$. By using a sequence of transitions, the handshake is delivered to one of the outgoing handshakes $H??+$. After sending out the grant signals, it results in mask hiding the initial requests causing signals $cg-$ and $rg-$ and the arbiters propagating new internal grants. The tile that produced $h+$

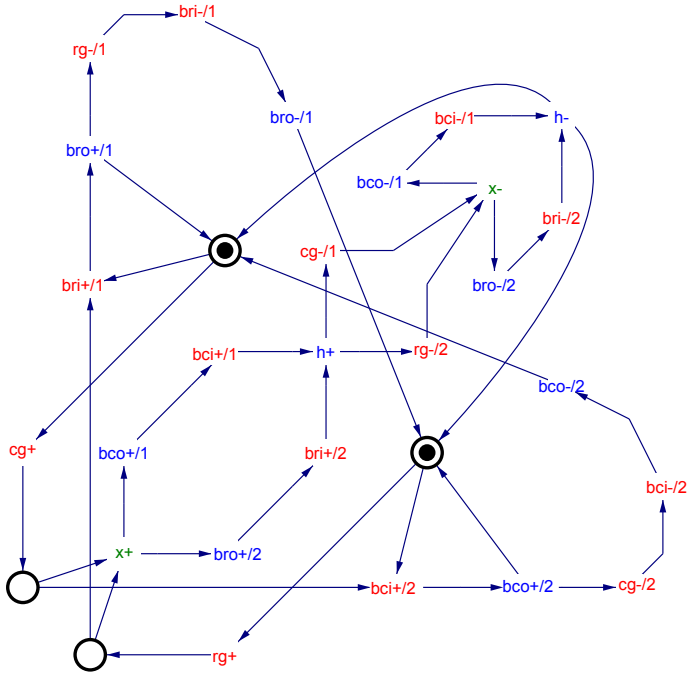


Figure 4: Ring-based tile STG

before now needs to remove its internal handshake while making sure the row and the column neighbours are freed only after the signals $cg-$ and $rg-$ have arrived. After it happens, we may release the handshake by $h-$.

Since both $cg+$ and $rg+$ signals are going to propagate to each tile in certain row and column, there will be tiles receiving only $cg+$ or $rg+$. The safe-net modelling rules require that the tokens produced in $x+$ pre-set are properly propagated. For that purpose we can use block inputs $bri+/1$ and $bci+/2$ because blocking signal according to the protocol always follows when $cg+$ and $rg+$ pair is found. The signal sequences $cg+ \rightarrow bci+ \rightarrow bco+ \rightarrow cg- \rightarrow bci- \rightarrow bco-$ and $rg+ \rightarrow bri+ \rightarrow bro+ \rightarrow rg- \rightarrow bri- \rightarrow bro-$ will form the communication protocol of the disabled tiles. In is important that the disabled tile never reacts with $h+$ and that is ensured in the given STG.

We can find the implementation of the arbiter using *Petrify* tool [1] directly from the modelled STG diagram (Fig. 4). It is presented in Figure 5. Let's consider, how it works. Initially, output of the gate 3 is high and all other outputs are low. When high input arrives on rg and cg, it affects gate 6 and consequently gate 4 (which is the internal x signal in the STG). The circuit is at state to block the conflicting neighbours. The signal on gate 4 ignites gates 2 and 5 producing outputs on bco and bro and disabling gate 6 by gate 5. Finally, as we would expect it from the environment, bro and bco would eventually produce inputs on bri and bci which would be the a sufficient condition to produce the output on h. The reset phase on signals rg and cg would first disable gate 3 (the output of. gate 4 is active). Which would disable the C-element and the output signals on bco and bro unblocking the row and the column. As soon as both the column and the row are unblocked, the output on h will go low.

The second scenario for the signal propagation is when the tile wasn't chosen for the handshake; however, it occurred to be one of the conflicting neighbours. This is the case, when only rg or cg arrives. It has no effect on gate 6, so the circuit does not react in producing either bco or

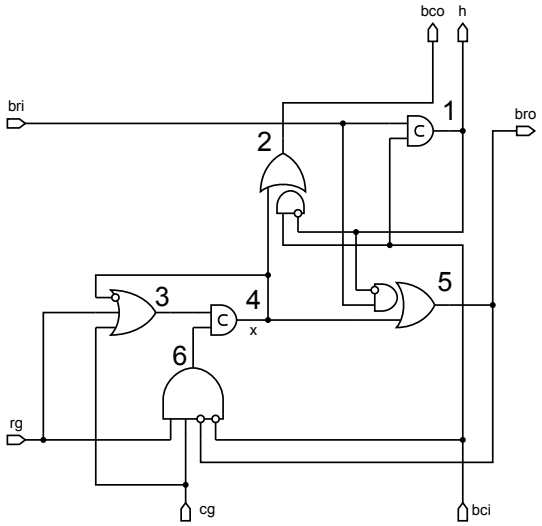
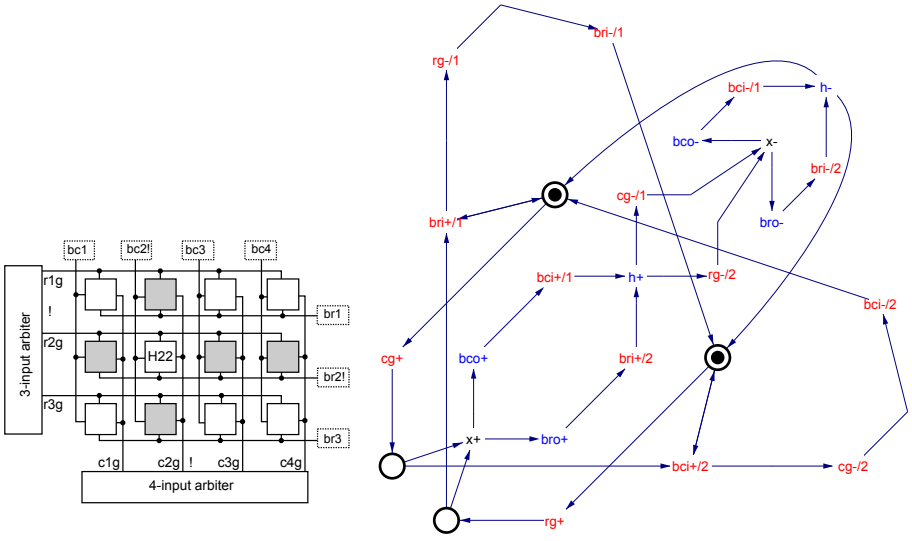


Figure 5: Tile implementing ring-based approach

bro. Nevertheless, the circuit does propagate both incoming block signals. Additionally, either *bri* or *bci* disable the activation of gate 6 preventing circuit reaction because of the race condition between old and new values on *rg* and *cg*.

2.2 Column/row block approach

The alternative way to distribute the block signal is by creating additional block tiles associated with each column and row (Fig. 6a). Similarly to Fig. 4, the activation (and deactivation) of the internal handshake *h* follows the change on the block signals *bci* and *bri*. Now, however, if the tile isn't selected by both client and resource, it is not propagating any of the incoming block signals to the output (Fig. 6b).



(a) Schematic diagram (b) STG of the tile

Figure 6: Arbiter with blocking tiles

One possible implementation of such a tile is shown in Fig. 7. The signals *bci* and *bri* are obtained by OR-ing all *bco* from the handshake column and all *bro* from the handshake row.

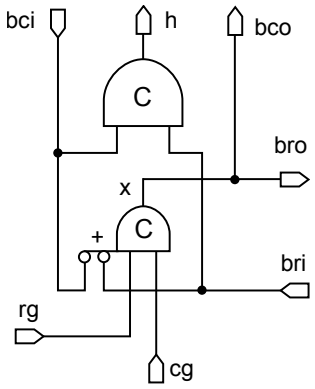


Figure 7: Tile implementation for the C/R block approach

3 Conclusions

This paper considers a problem of creating the $N \times M$ arbiter with active resources. It is shown that the initial design is not SI extendable as each additional client or resource is bound to increase the fan-in. Two approaches are considered. The first is based on tiles being interconnected in a ring propagating the block signal. The second and simpler approach is designed to block the whole column/row and is expected to be faster than that ring-based solution.

References

[1] Petrify: `http://www.lsi.upc.es/~jordic/petrify/petrify.html`.