Modular approach to multi-resource arbiter design

Stanislavs Golubcovs Delong Shang Fei Xia
Andrey Mokhov Alex Yakovlev

Newcastle University

ASYNC 2009

Summary

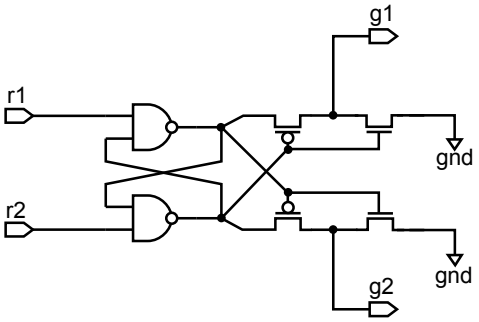
Newcastle University

Idea of arbitration

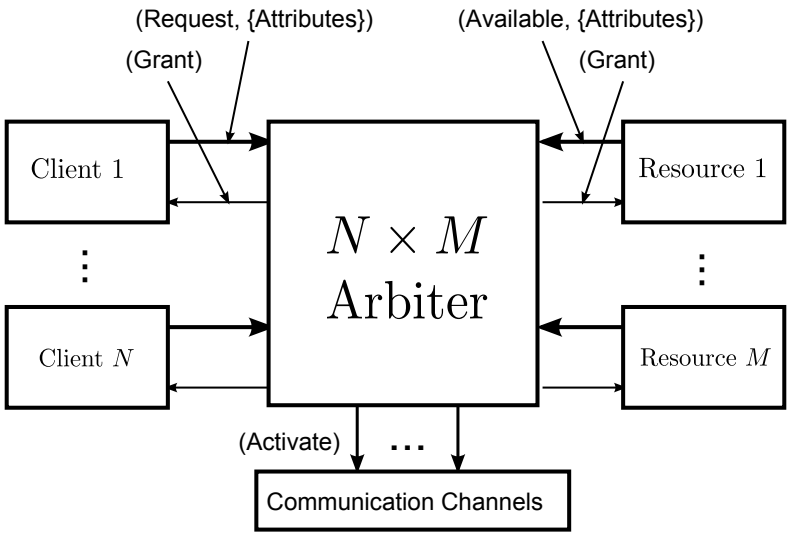
Definition

An arbiter is a circuit managing access of other independed circuits to a number of shared resources while preventing the conflicts

- The key component of any arbiter is the ME element
- It can have many clients and maybe many resources
- Resources can use explicit communication to inform about their availability (active resources)



General active resource arbiter structure



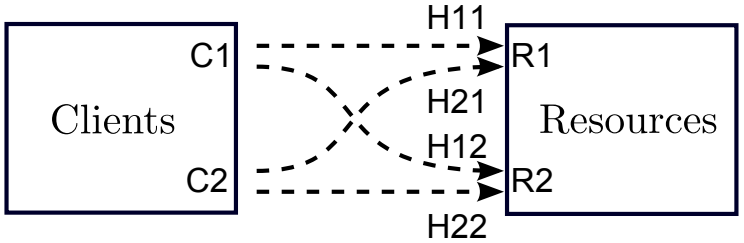
Features

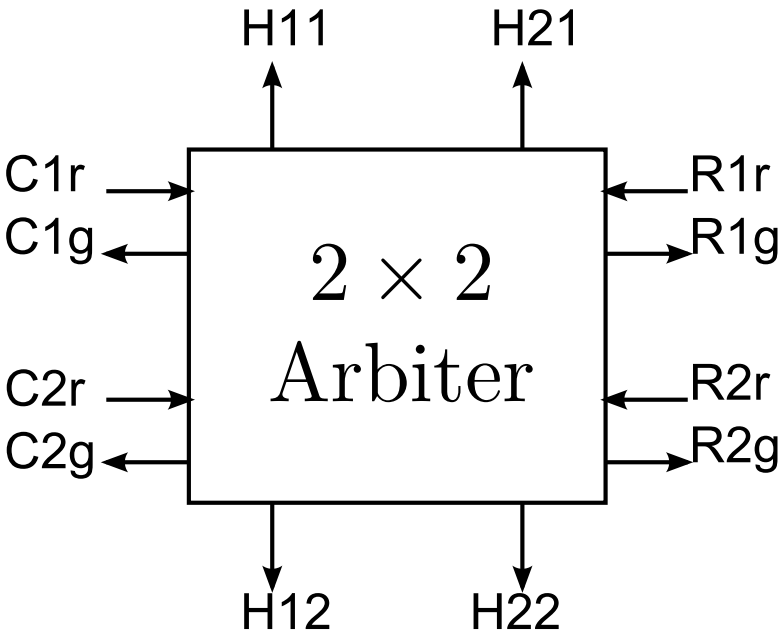
- Resources actively inform about their availability
- The arbiter activates channels connecting one client with one resource also called the handshakes
- Concurrent non-conflicting handshakes are allowed
- Structure is symmetric (resources are also the clients of their own kind)

Difficulties

- Requests may come at any time
- Sometimes more than one resolution exists

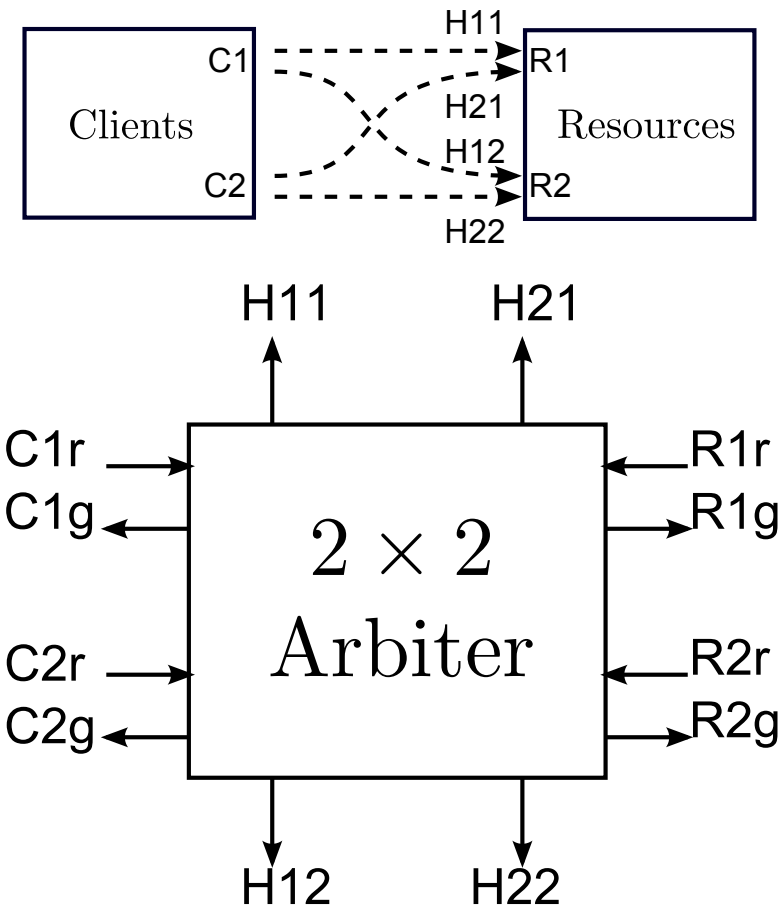
Task specification

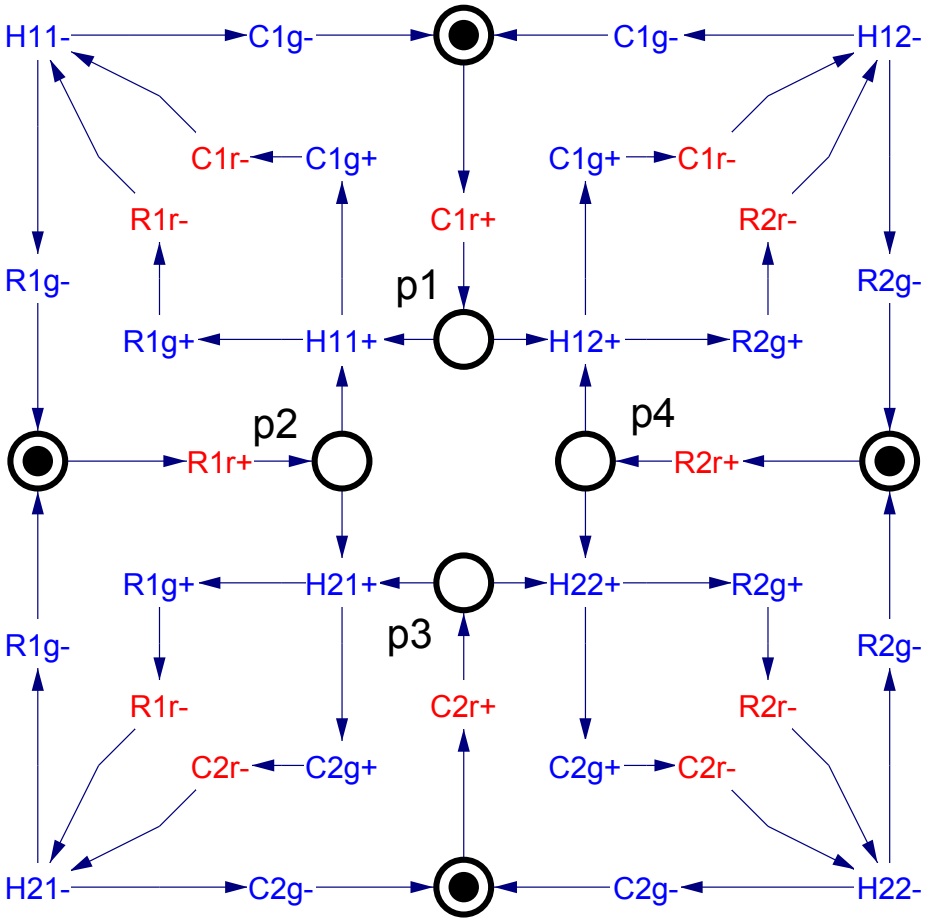




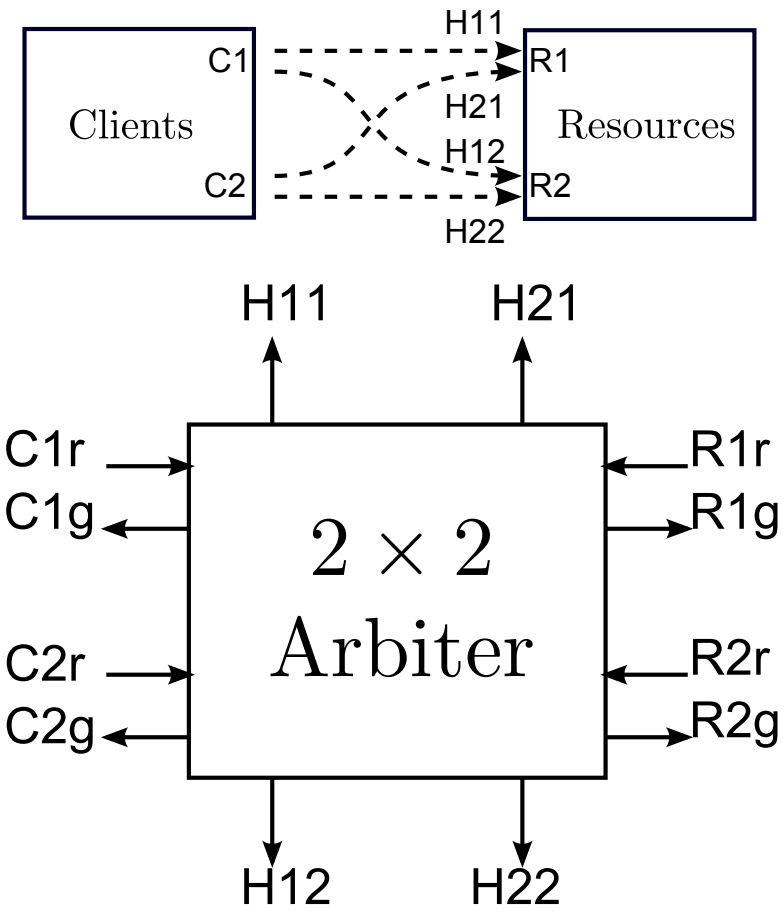
- Design the 2×2 multi-resource arbiter
- Find SI, asynchronous implementation
- Allow concurrent handshakes
- Prevent conflicting handshakes
- Minimise response latency

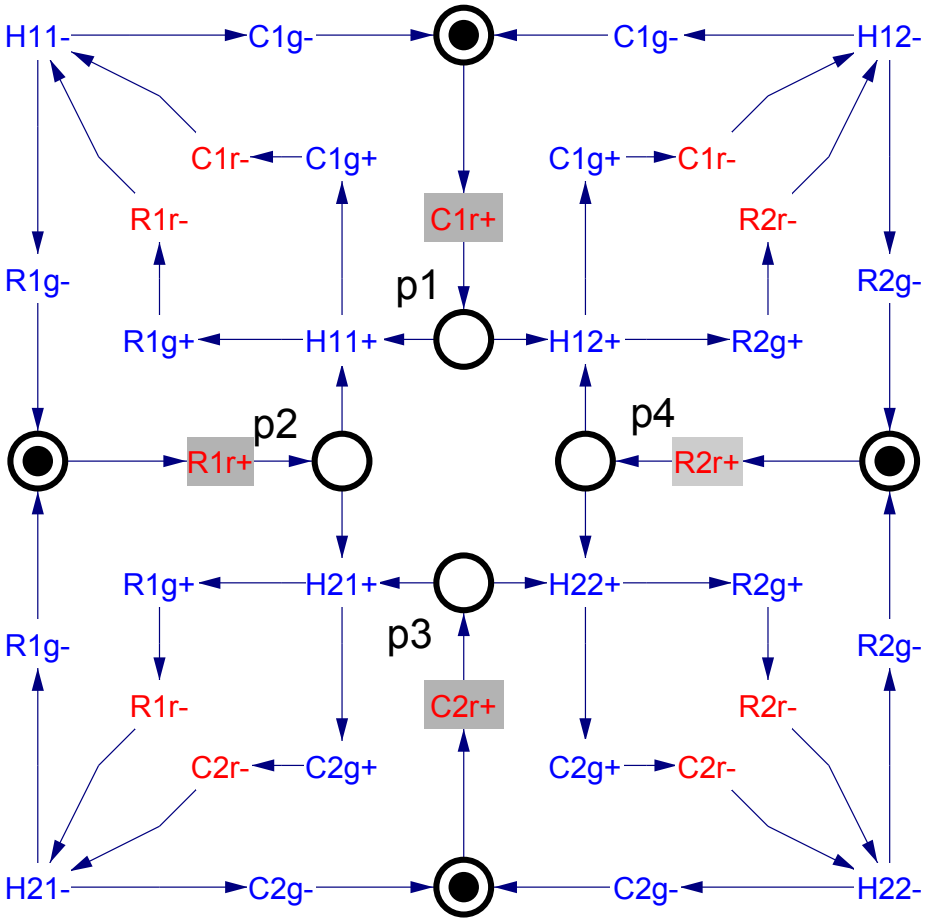
Task specification



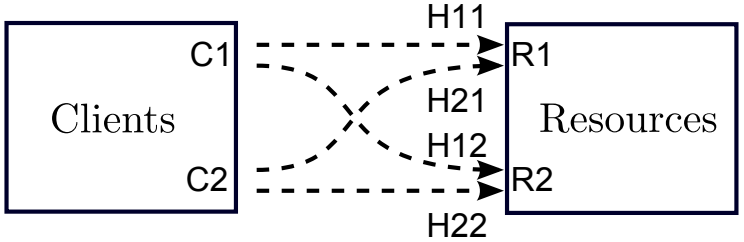


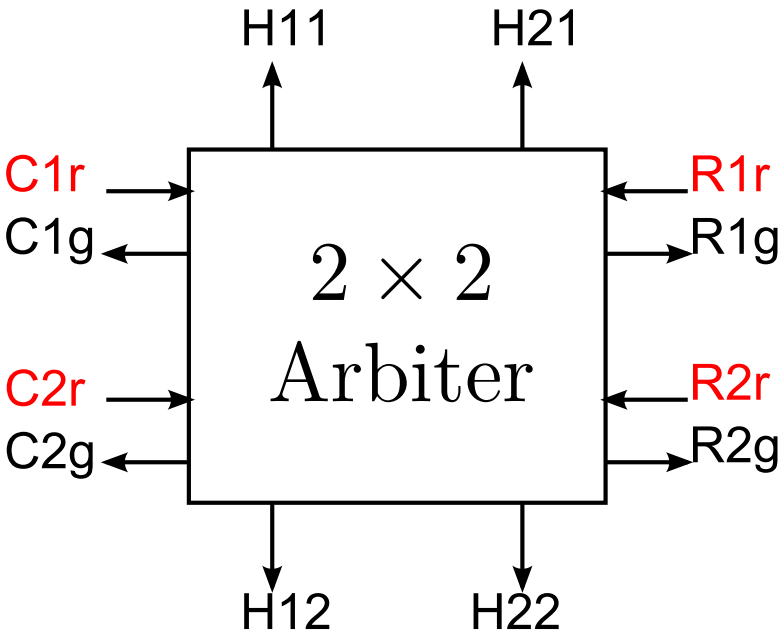
Task specification

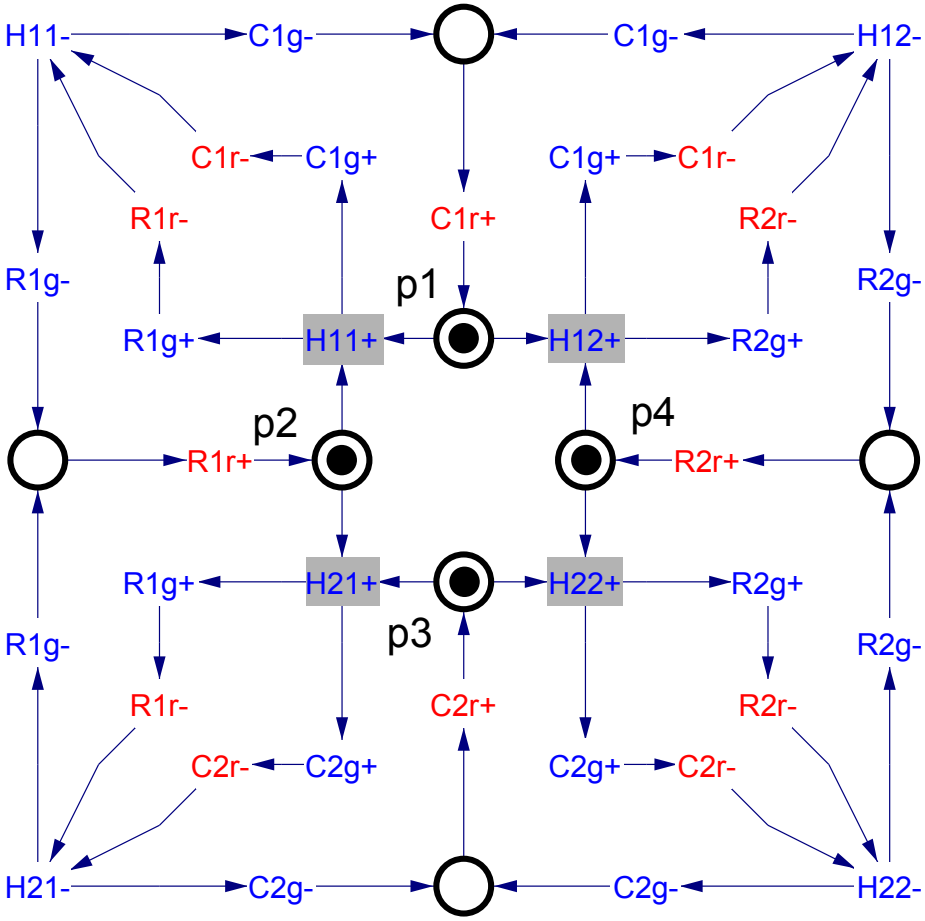




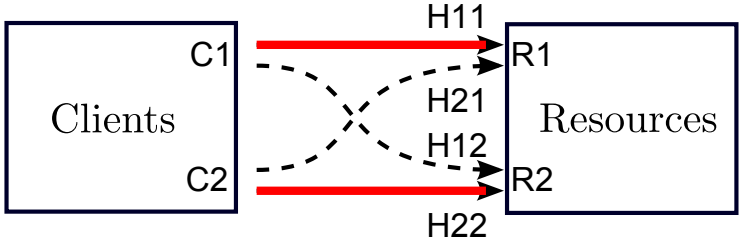
Task specification

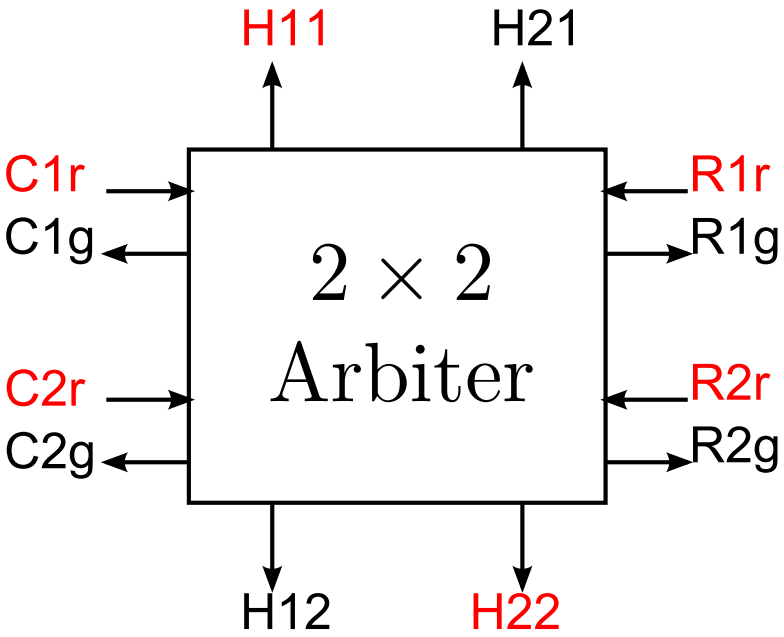


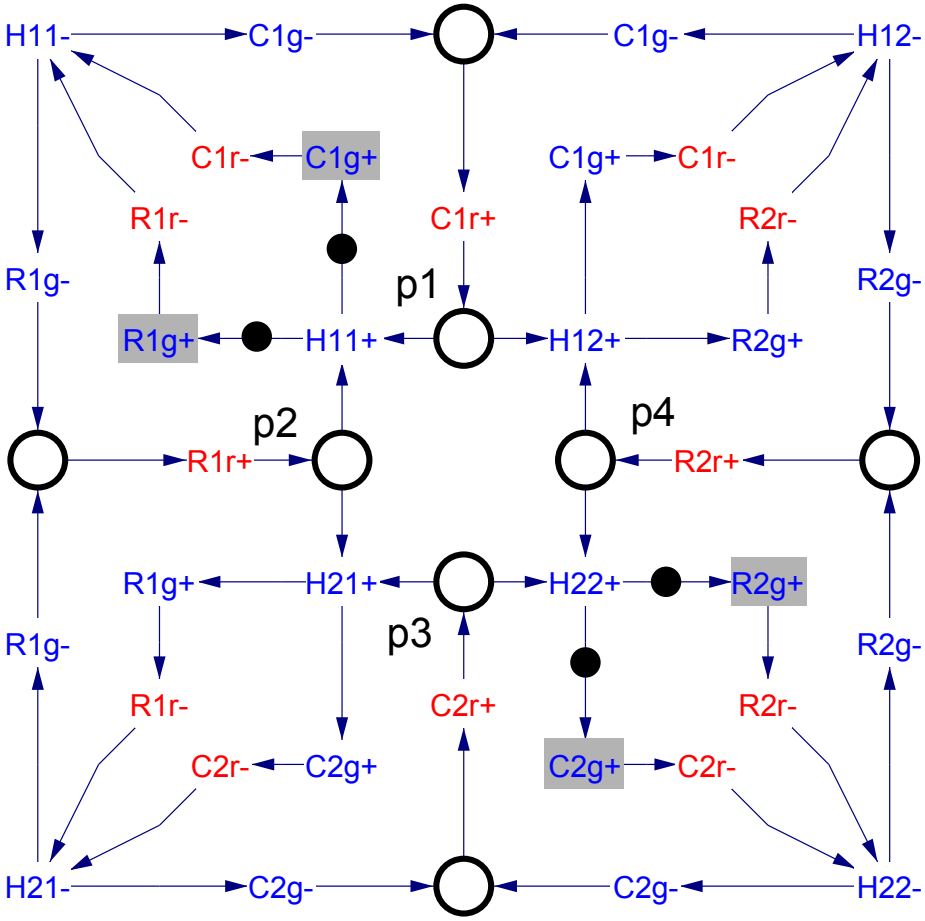




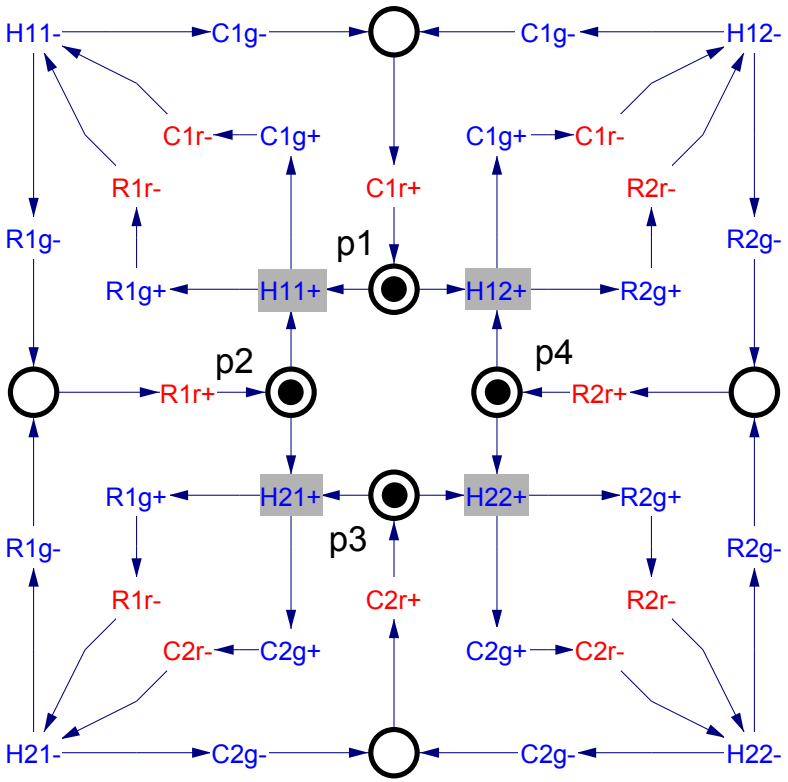
Task specification

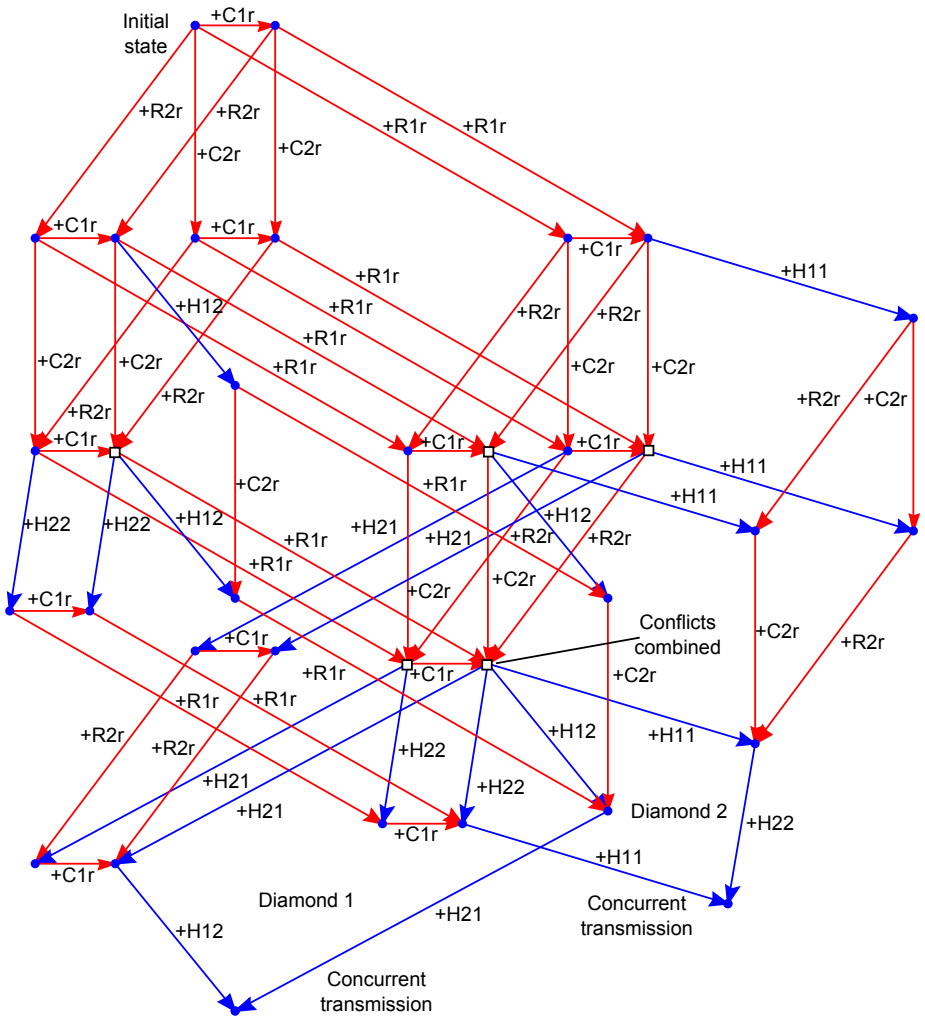




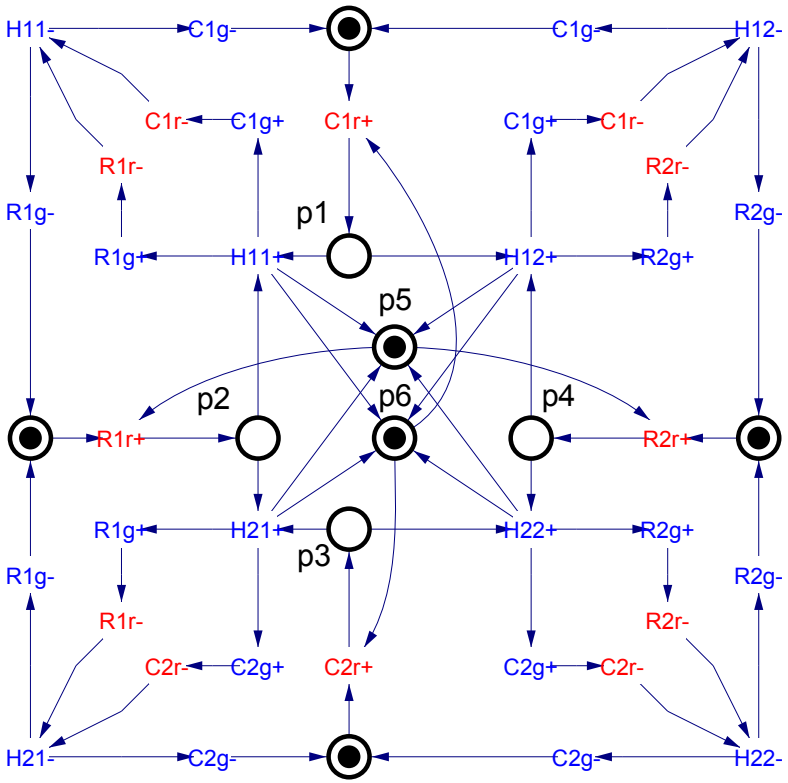


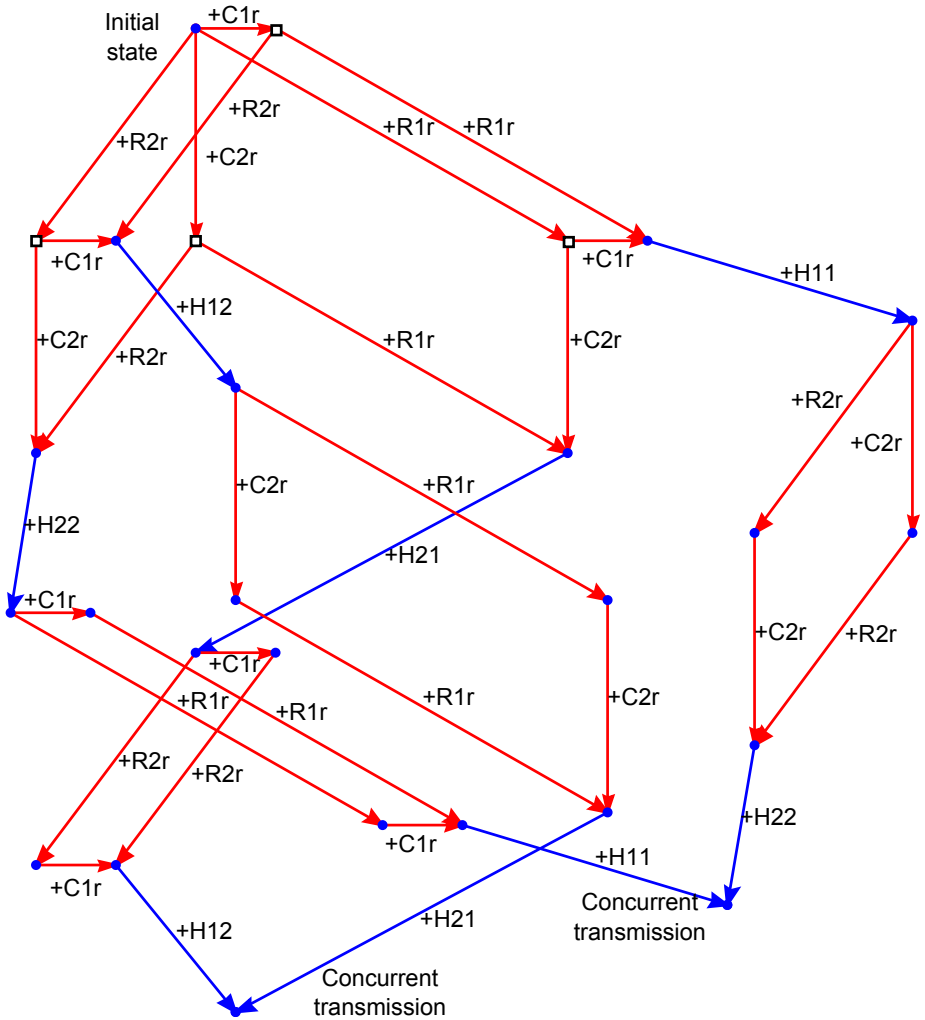
Resolving conflict



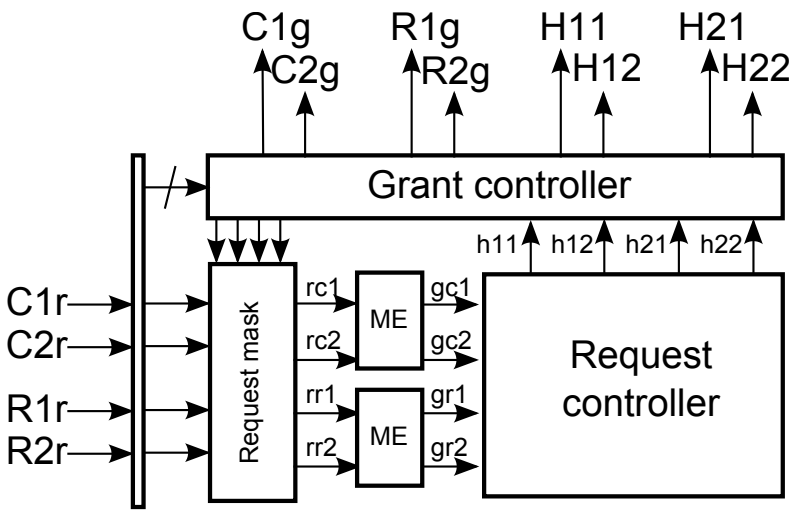


Resolving conflict



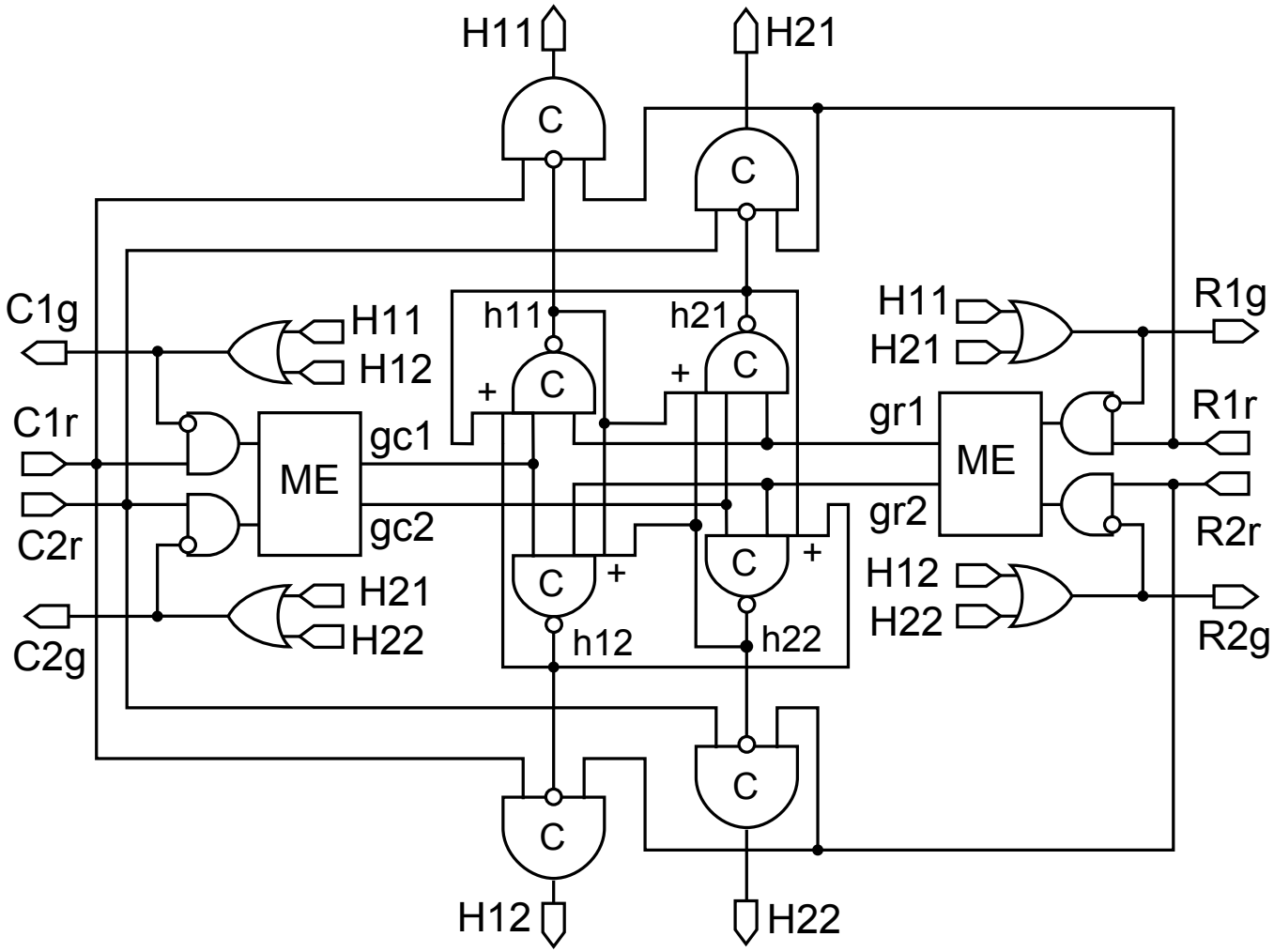


Overall structure

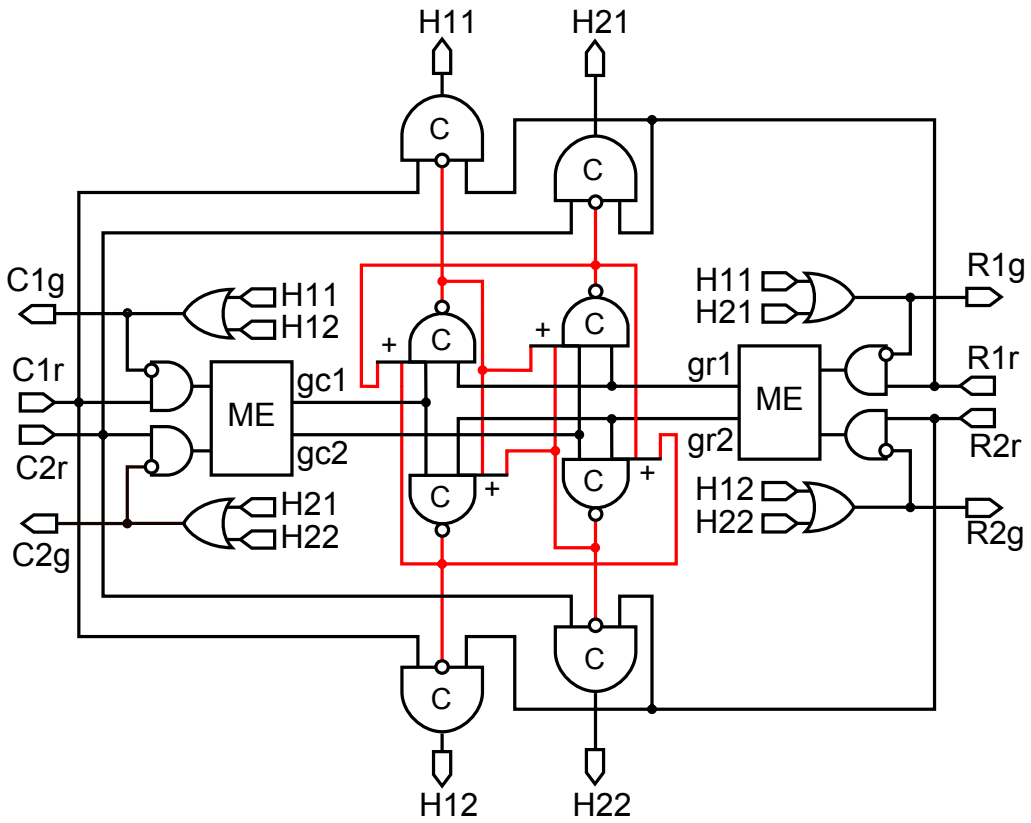


- ME elements ensure not more than one channel can be selected by the request controller
- The request controller selects one channel to be activated (inside critical section)
- Grant controller activates the channel, returns grant signals (outside critical section)
- Request mask hides initial requests as grant signals arrive to free the MUTEX-es and the Request controller for further requests

Implementation

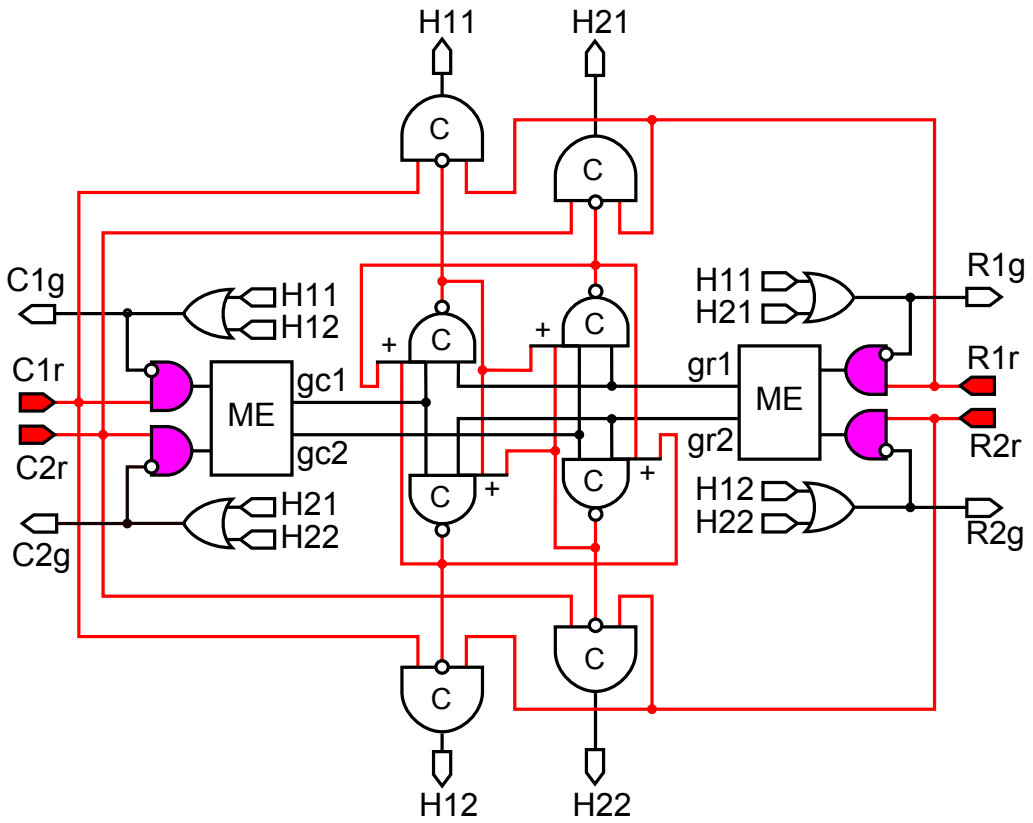


Arbiter in action



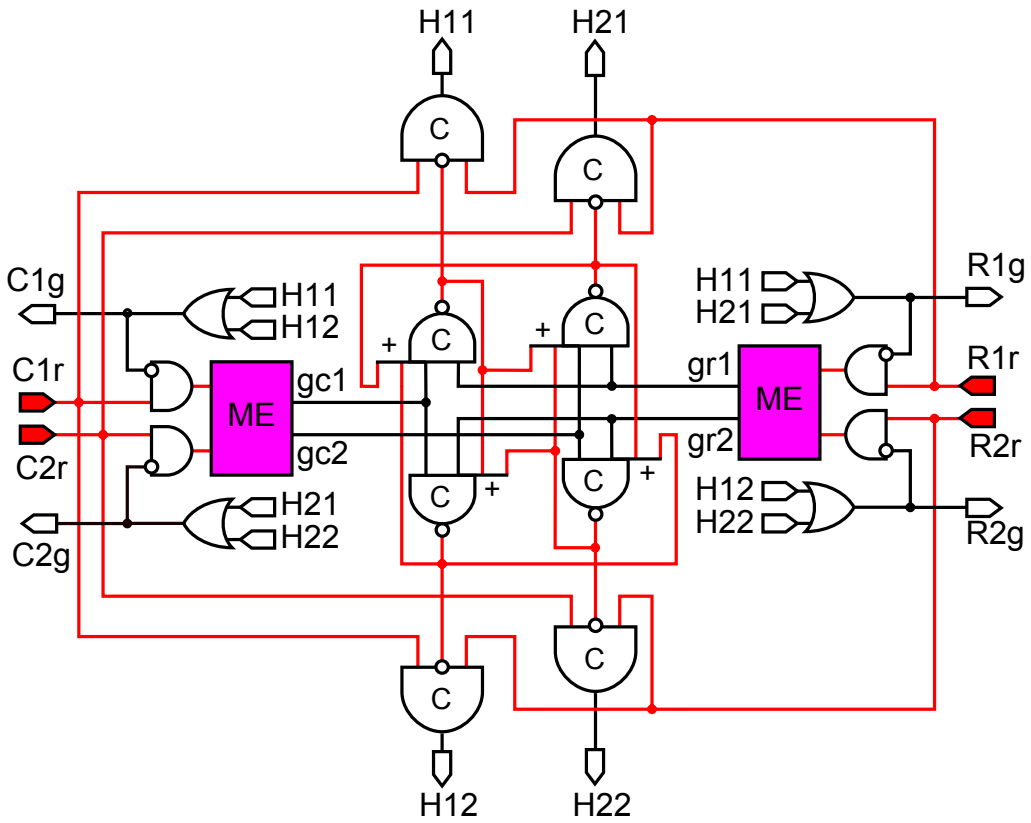
The initial state

Arbiter in action



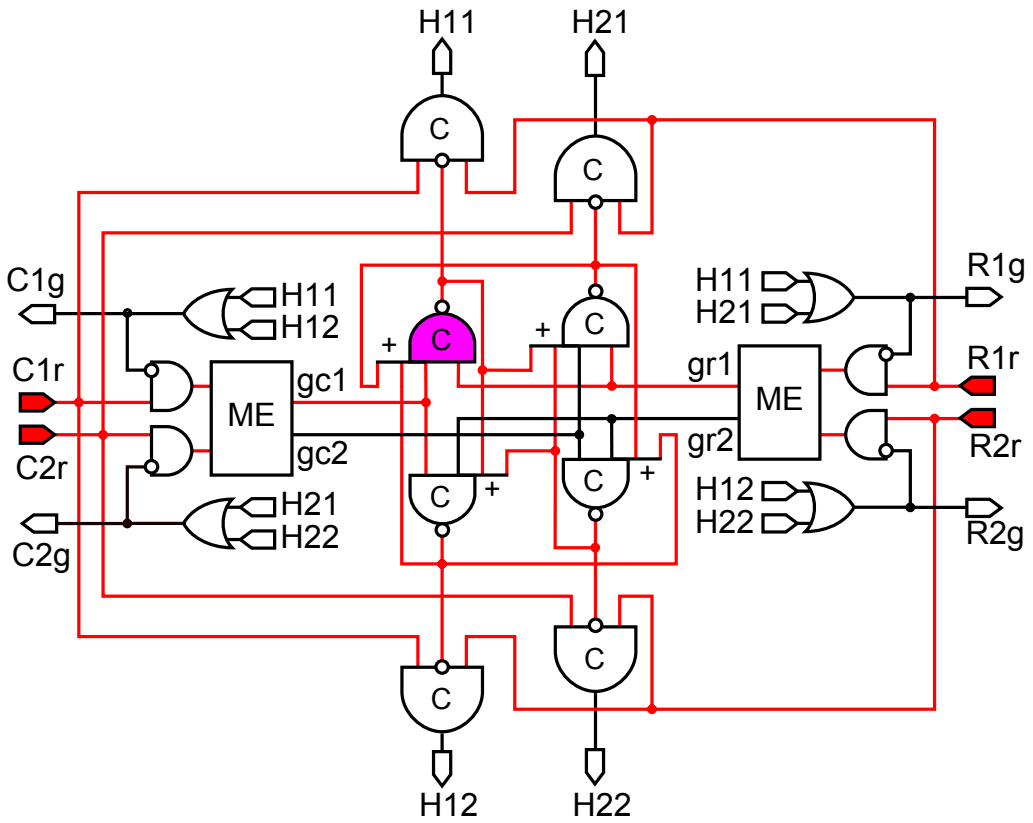
All 4 request signals are coming at the same time. The mask is propagating the requests

Arbiter in action



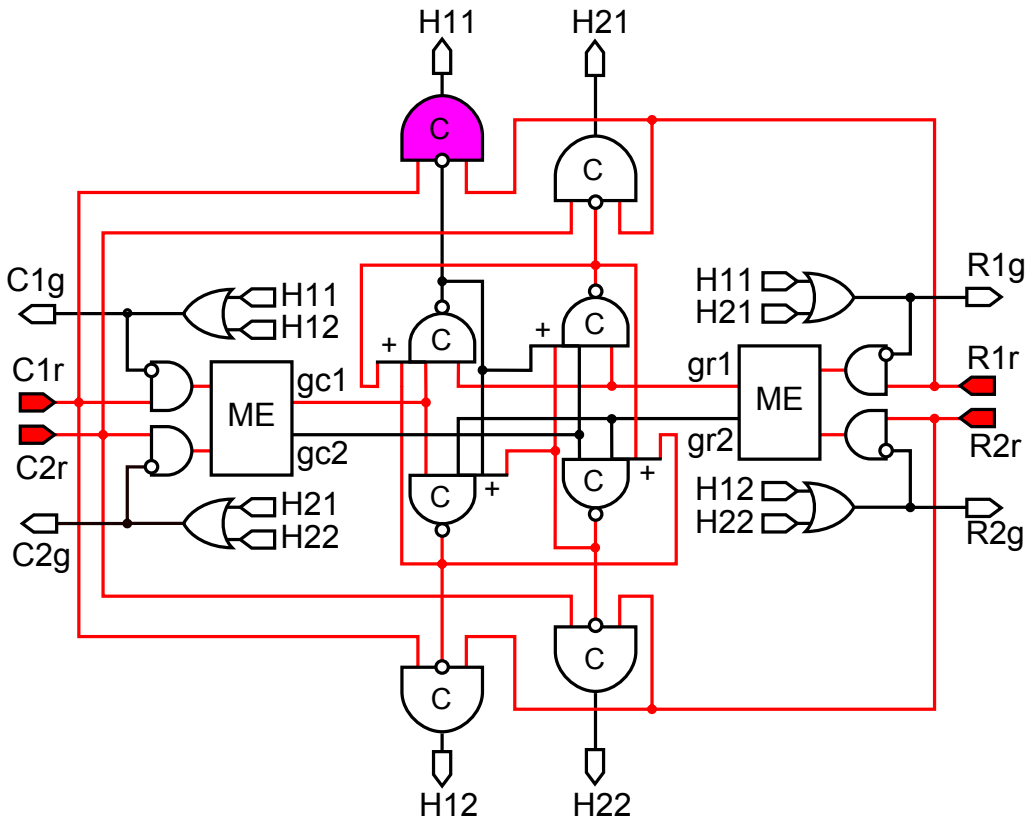
Both ME elements receive the signal and do the arbitration

Arbiter in action



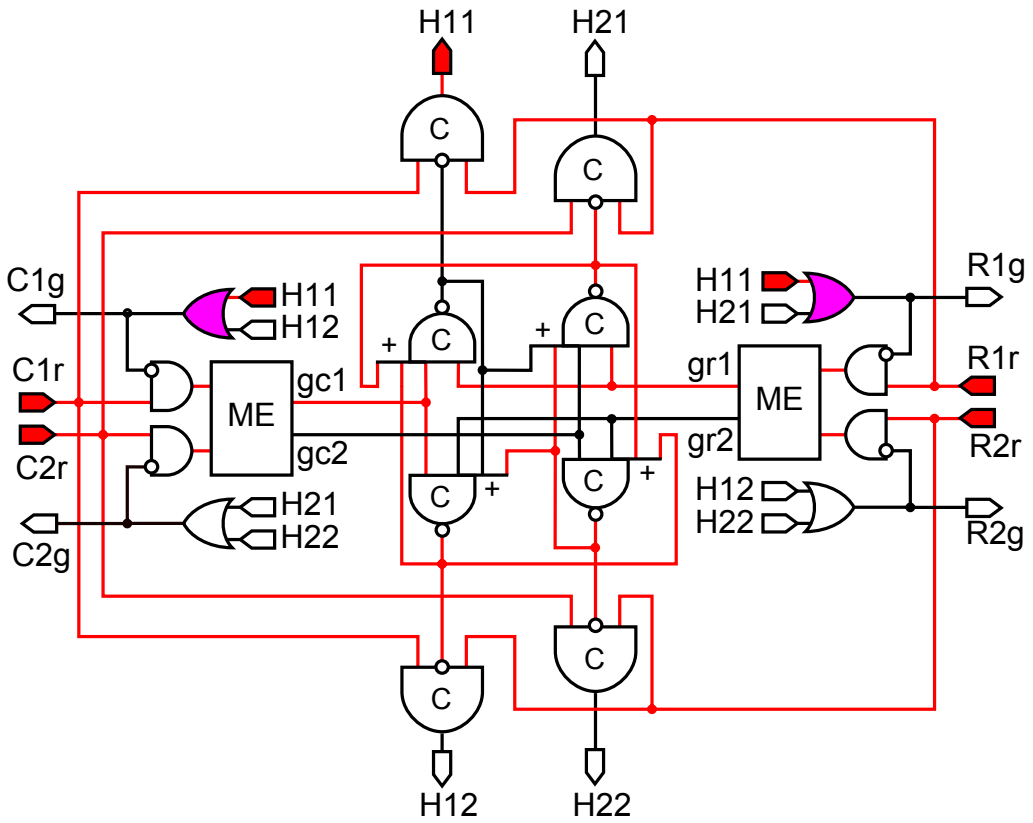
Client $C1$ and resource $R1$ are chosen for the handshake allowing the request controller to make unambiguous decision to activate $H11$ next

Arbiter in action



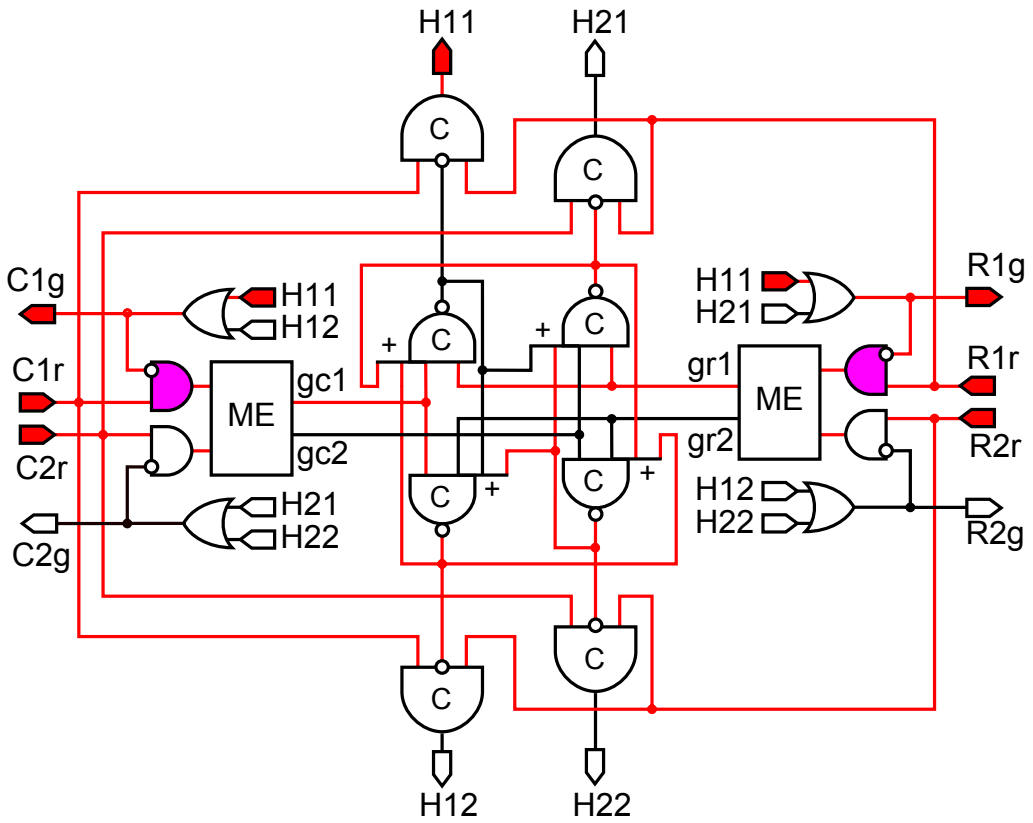
The asymmetric C-element activates outgoing handshake $H11$ and blocks conflicting handshakes on $H12$ and $H21$

Arbiter in action



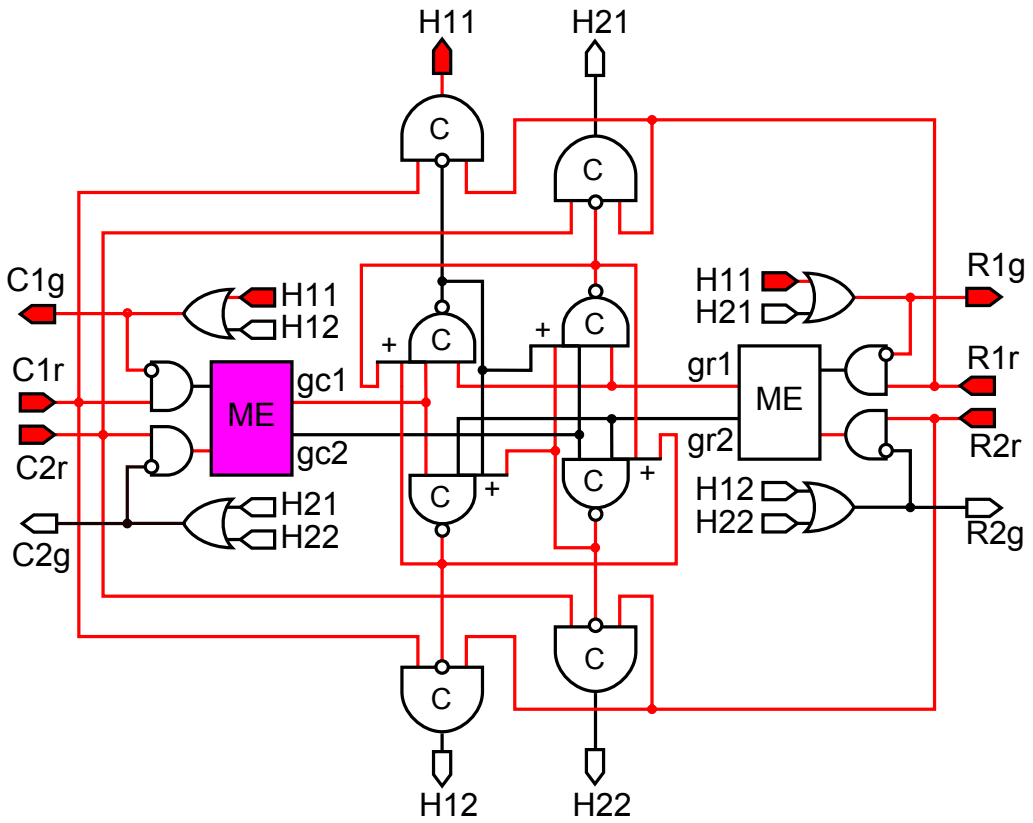
The handshake $H11$ is activated followed by outgoing grant signals on $C1g$ and $R1g$

Arbiter in action



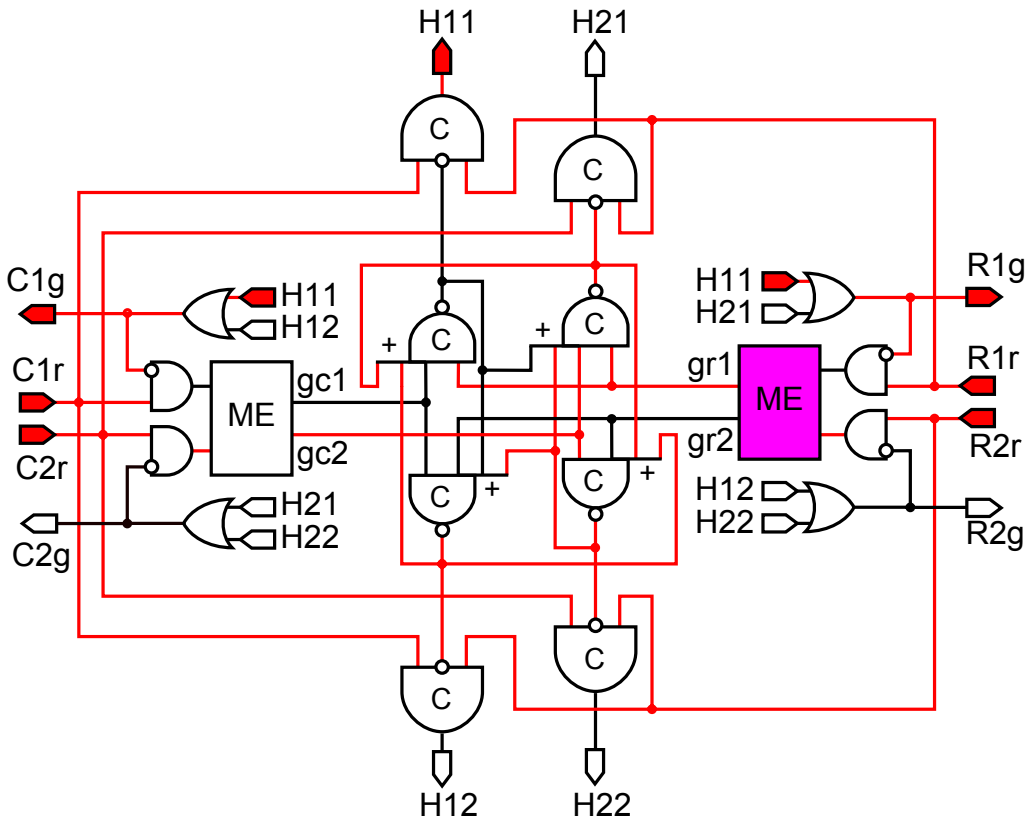
The handshake $H11$ is activated followed by outgoing grant signals on $C1g$ and $R1g$

Arbiter in action



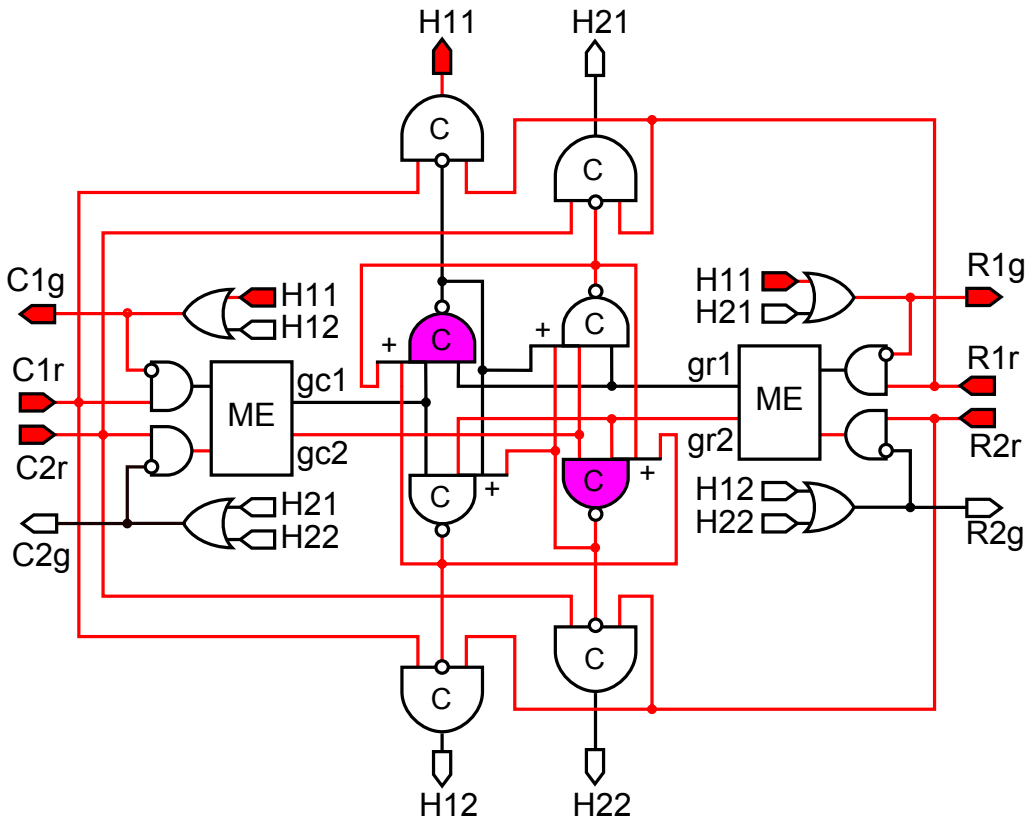
Mask hides the initial requests and allows the next pair of requests to propagate. (Suppose the mutual exclution element on the right side is slower)

Arbiter in action



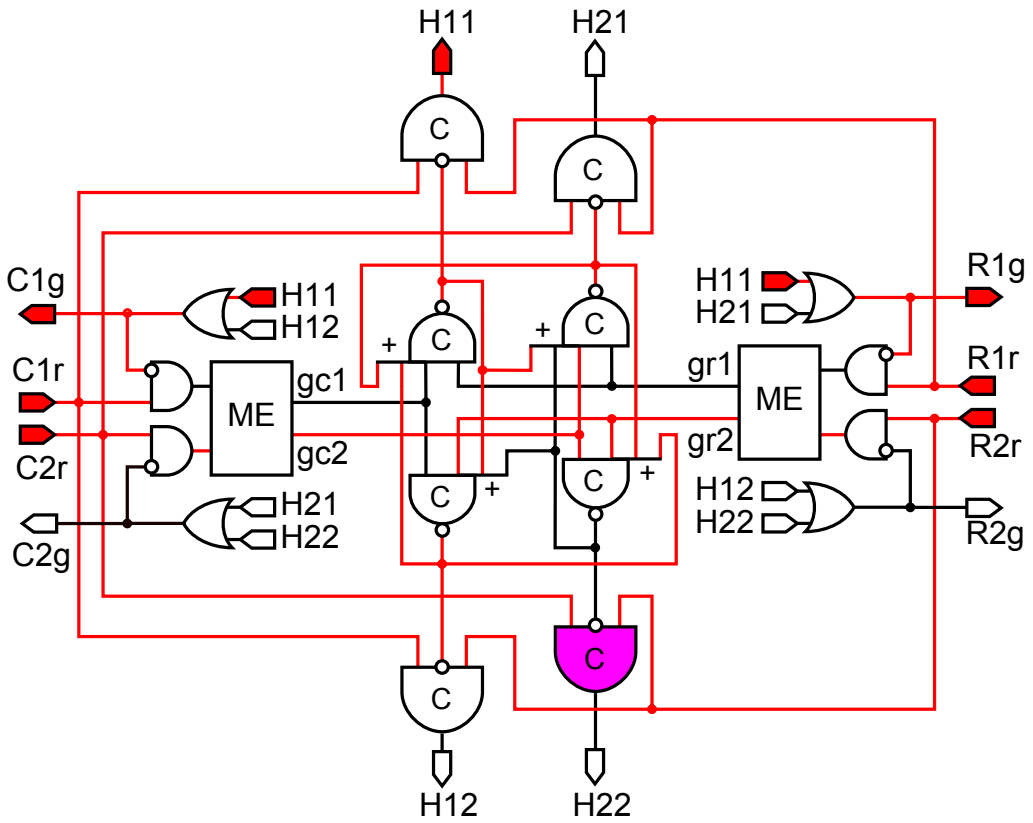
Left ME element has propagated the new client request, while the old resource request is active. That could be a glitch, however, asymmetric C-element for $H21$ is blocked

Arbiter in action



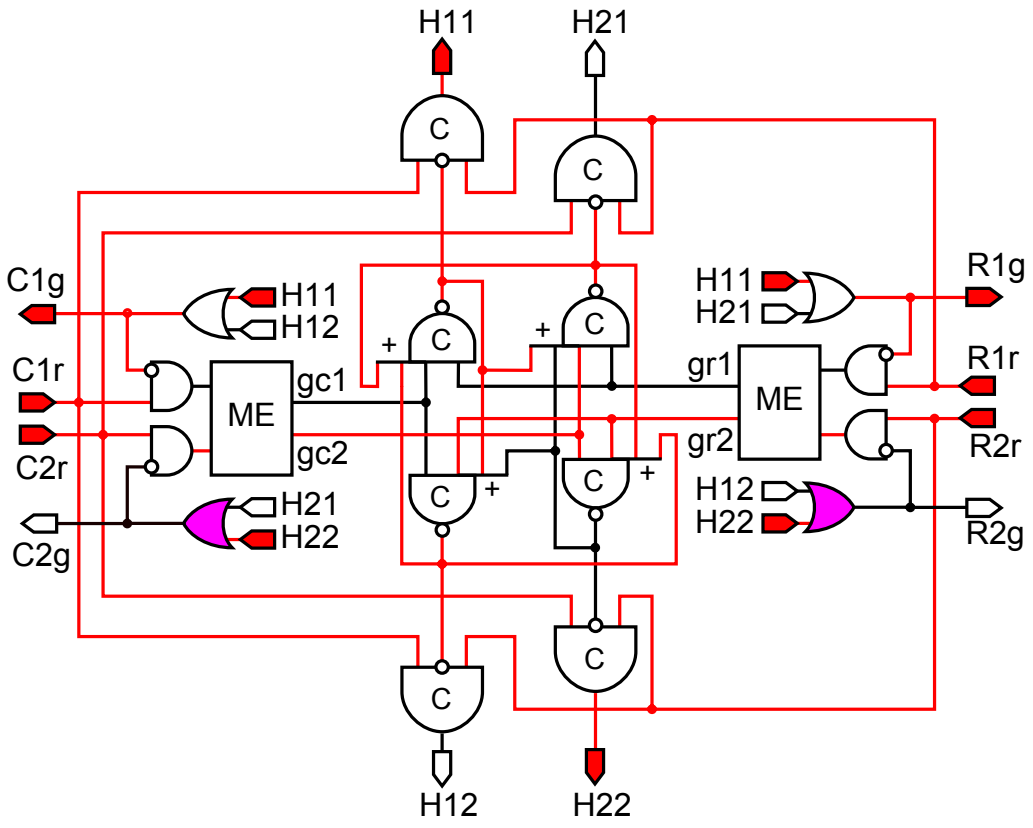
Now the request controller removes the request from the $H11$ and initiates a non-conflicting handshake $H22$

Arbiter in action



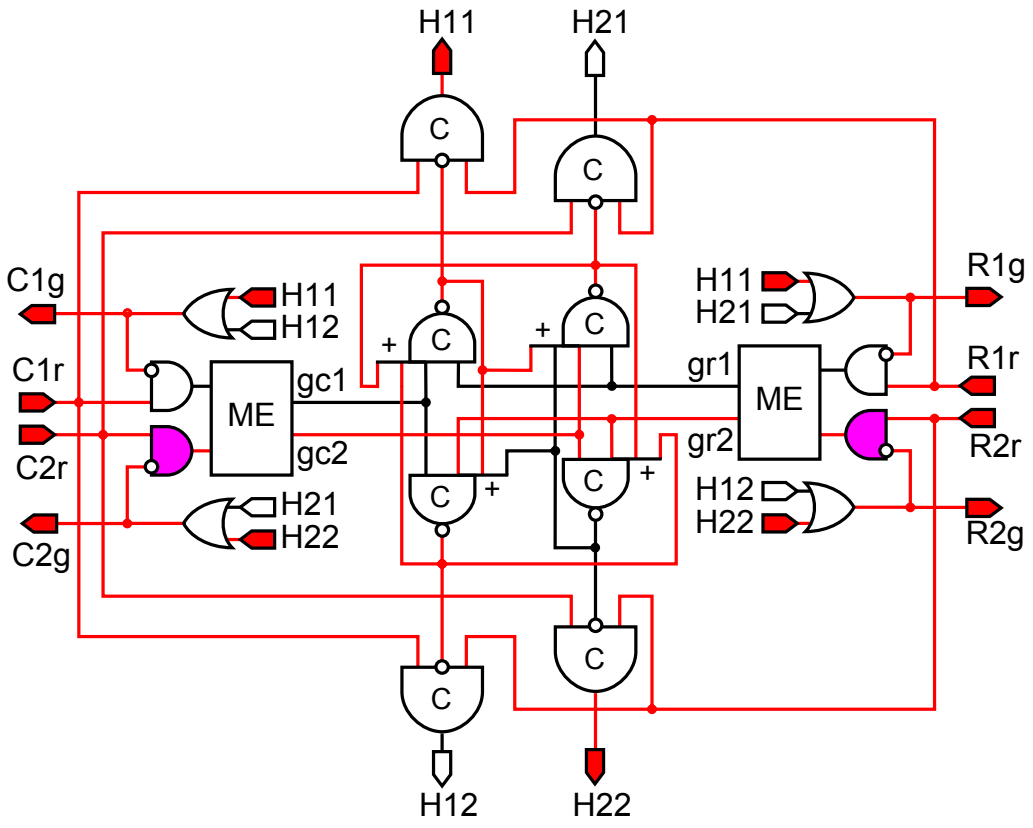
Now the request controller removes the request from the $H11$ and initiates a non-conflicting handshake $H22$

Arbiter in action



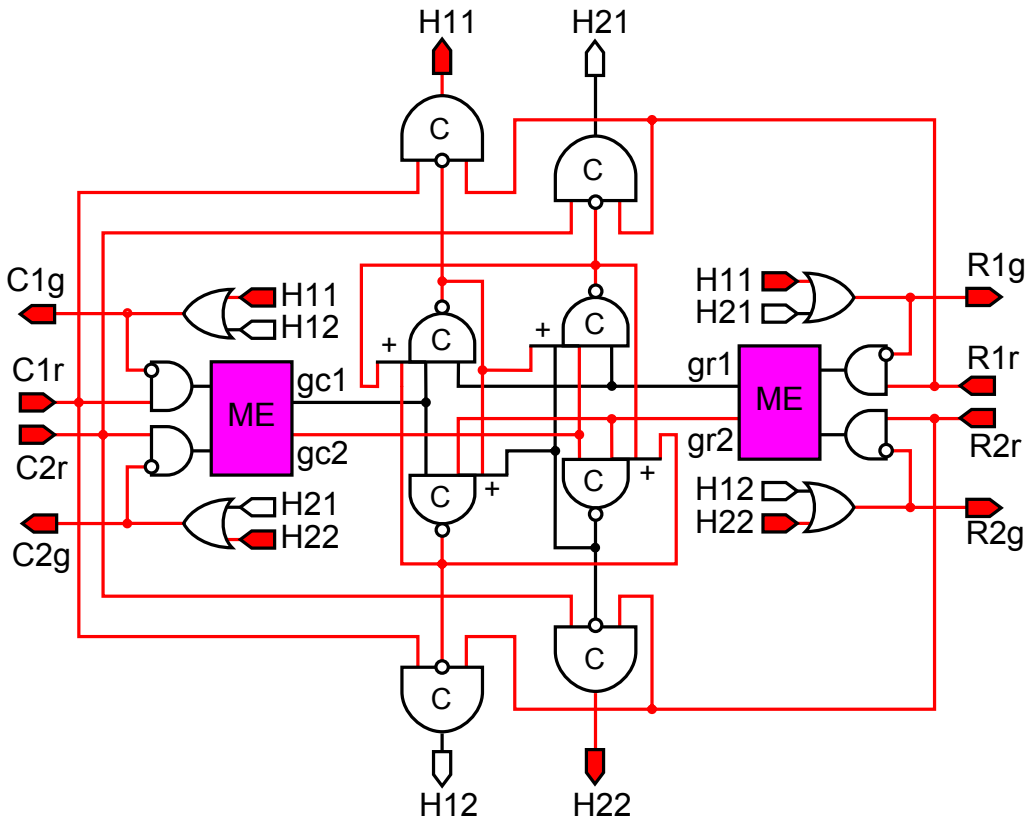
The signal flow proceeds similarly to $H11$

Arbiter in action



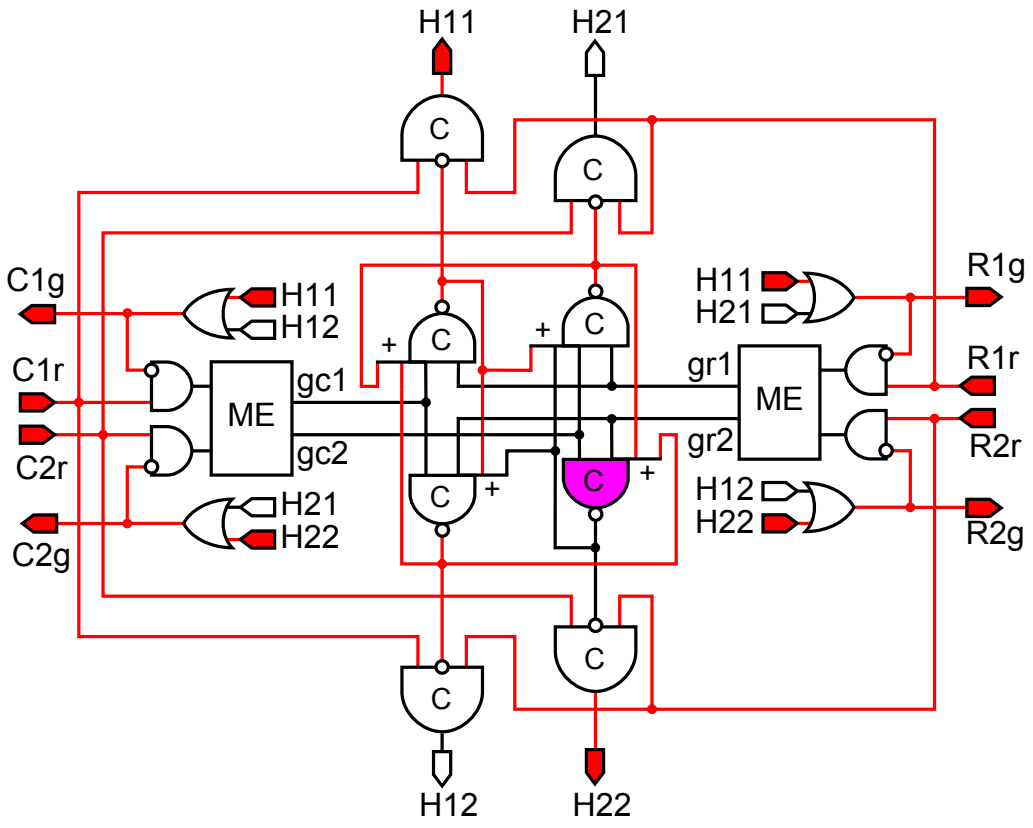
The signal flow proceeds similarly to $H11$

Arbiter in action



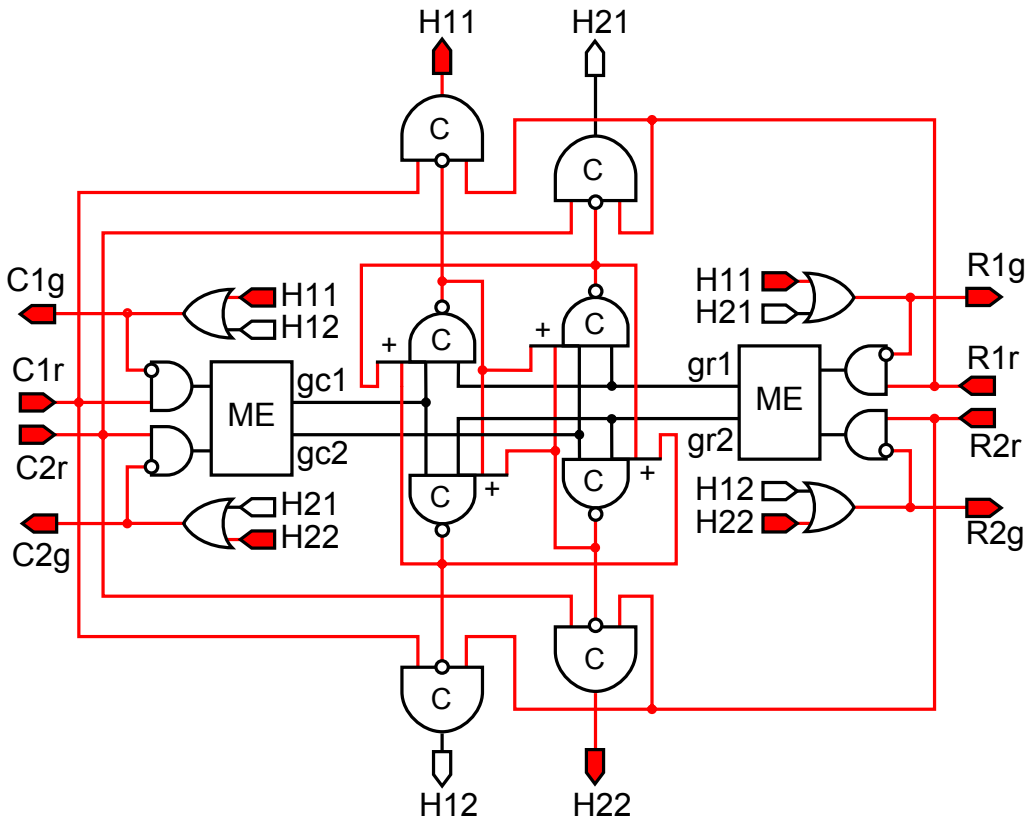
The signal flow proceeds similarly to $H11$

Arbiter in action



The signal flow proceeds similarly to $H11$

Arbiter in action



Now the signal is fully propagated and the circuit awaits changes on the inputs

Simulation in SPECTRE

- Modelling is done using UMC90nm technology
- Average response latency for first pair of requests: $\sim$ 450ps
- Worst case for second pair of requests: $\sim$ 960ps (if no metastability needs to be resolved)
- Best response time for the second pair of requests: $\sim$ 450ps

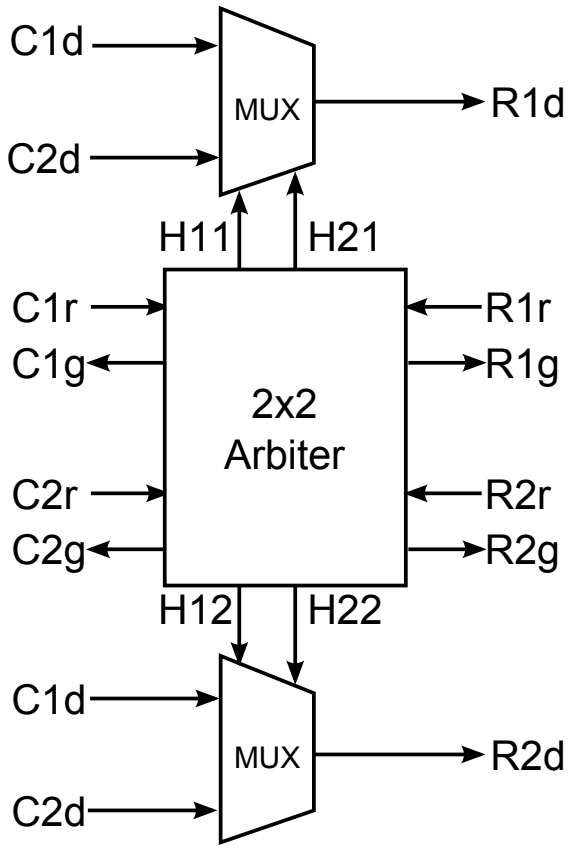
Newcastle University

Proving to be Speed-Independent

- The arbiter was verified using the approach presented in [ASYNC'08]:
  - Design gate-level model (Workcraft)
  - Generate circuit Petri Net (Workcraft)
  - Composition with the environment STG (Workcraft)
  - Unfolding of the obtained Petri net (Punf)
  - Reachibility analisys on the obtained unfolding (MPSat)

Newcastle University

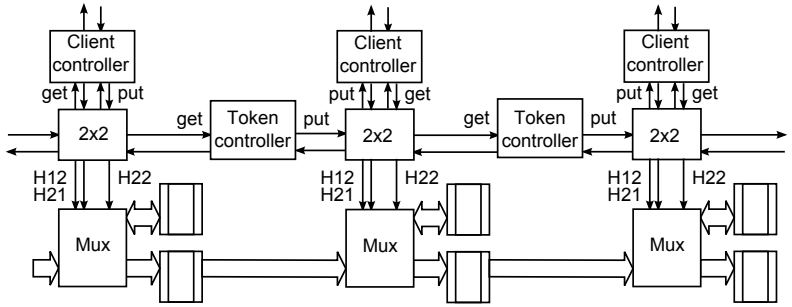
Example of usage 1
Data transportation

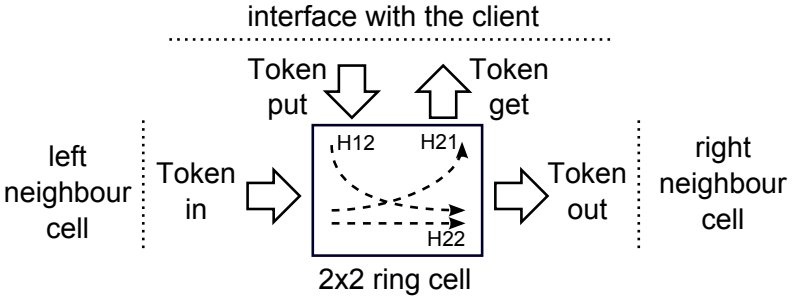


- Handshakes form dual rail signal
- Multiplexers select corresponding channel for data transport from clients to resources
- Useful for data producer/consumer interactions, for instance, processors executing instructions

Example of usage 2

Multi-resource arbiter with passive resources



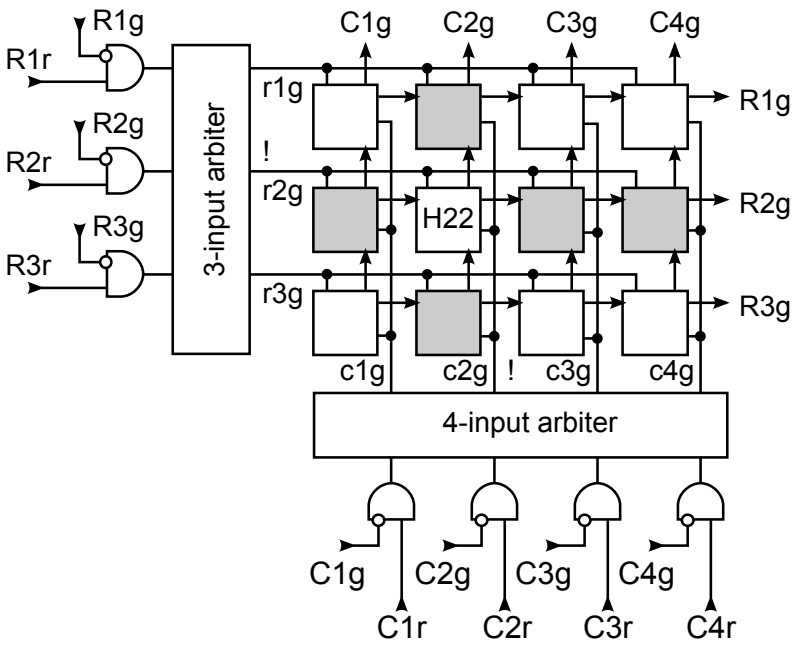


- 2×2 arbiter is used to propagate tokens
- Clients add and remove tokens from the loop
- Each token represents one resouce
- The tokens can be taken concurrently
- Usual passive resources

Extending the design

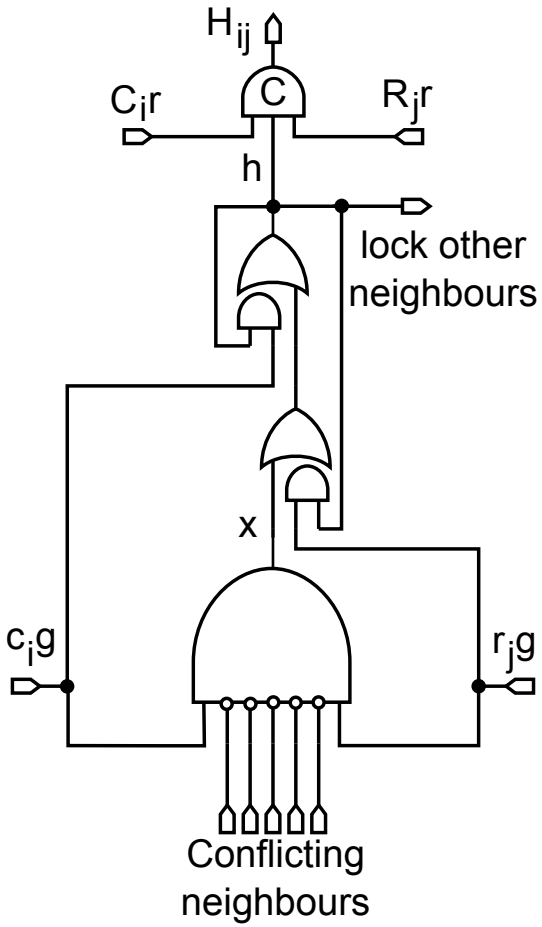
How can we extend the design?

Overall structure



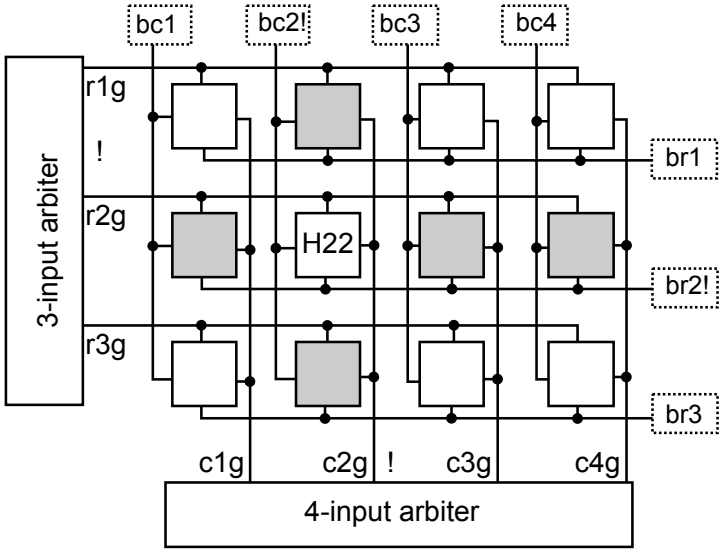
- Same basic structure as in 2×2 (we have masks, arbiters and the controllers)
- The request and the grant controller are constructed of $N \times M$ modules
- All conflicting handshakes are on the same row and column
- The N- and M-input arbiters are needed
- What each of the modules would look like?

Direct approach



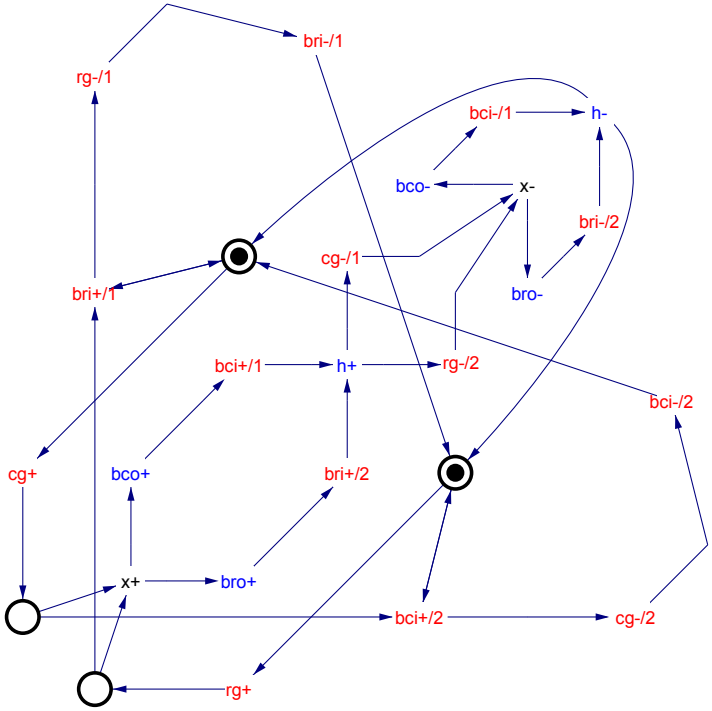
- Asymmetric C-element can be decomposed
- It has an increasing fan-in for each new conflicting handshake!
- Still can be useful for the designs with timing assumptions

Column and row blocks



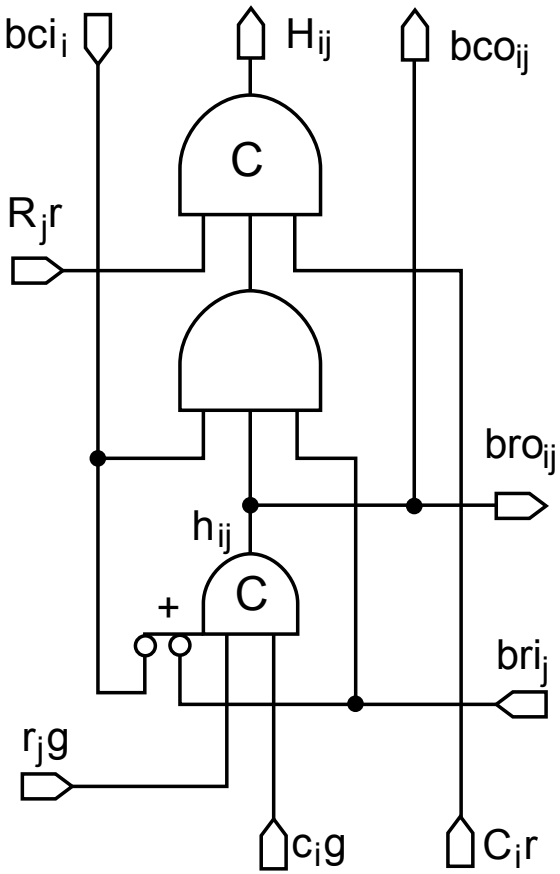
- Separate block components can be used
- The design is created by modelling a Petri net
- Block components are simply OR-ing the block signal and are activated by not more that one handshake (can be decomposed)

Column and row blocks



- Separate block components can be used
- The design is created by modelling a Petri net
- Block components are simply OR-ing the block signal and are activated by not more that one handshake (can be decomposed)

Column and row blocks



- Separate block components can be used
- The design is created by modelling a Petri net
- Block components are simply OR-ing the block signal and are activated by not more that one handshake (can be decomposed)

Summary

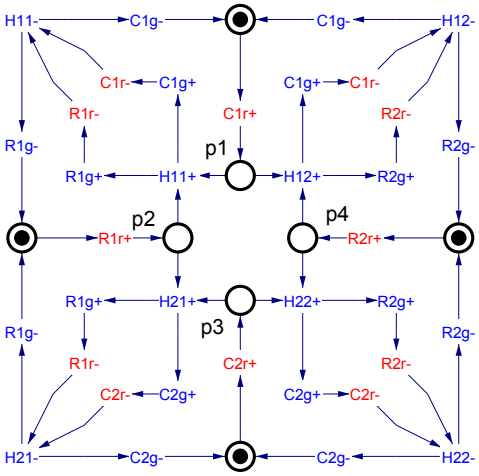
- :) Found implementation for 2×2 and general $N \times M$ arbiter with active resources
- :) The arbiter is dead-lock/live-lock free
- :) The multi-client/resource arbiter fairness depends on the implementation of the internal arbiters
- :) The block based implementation of request controller response increases logarithmically for each new client or resource
- :) Can be used for creating the N-client arbiter with M passive resources
- :(Worst case response latency linearly increases with $\min(N, M)$ (additional ~ 500ps for each new handshake)

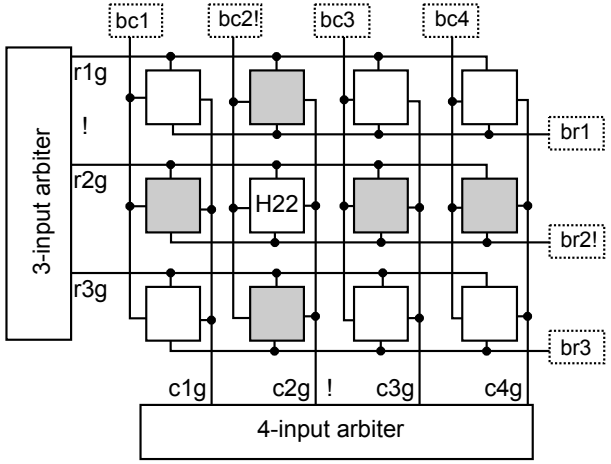
Newcastle University

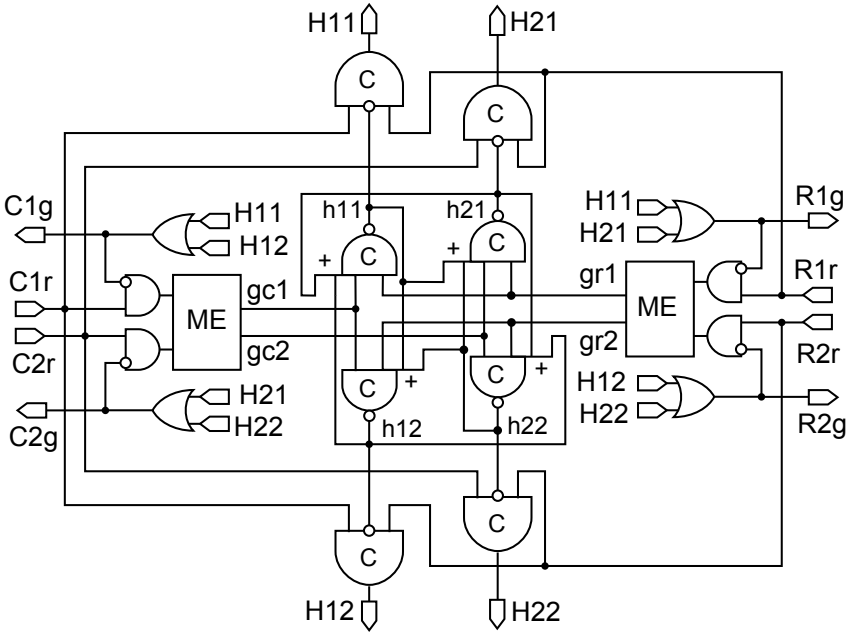
Thank you!

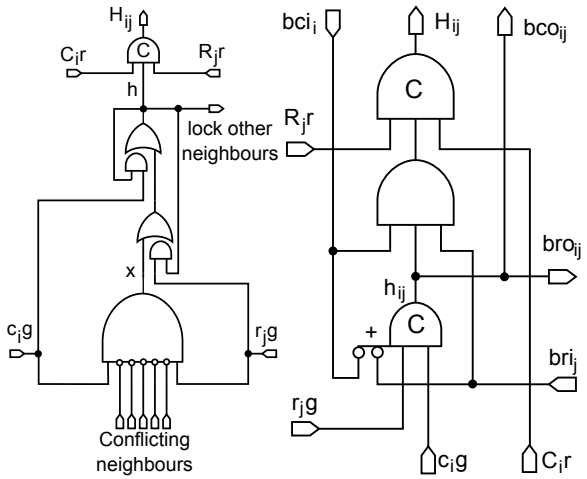
Questions?

Key pictures...

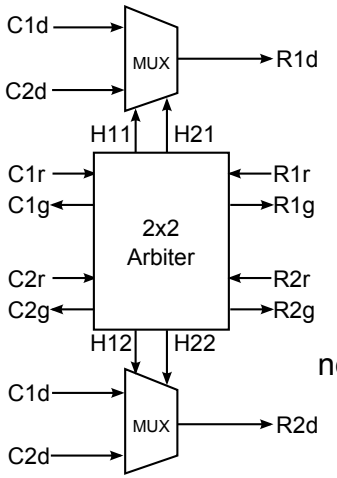


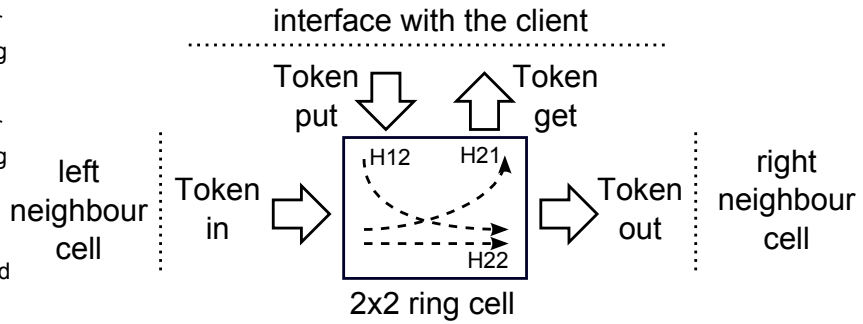


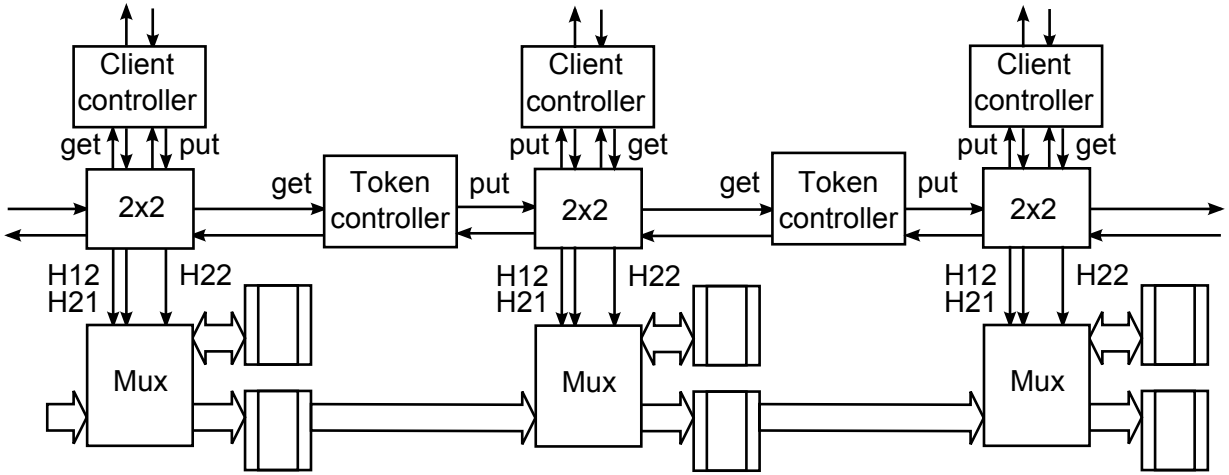




Key pictures 2...







Bibliography

V. Khomenko, M. Koutny, and A. Yakovlev.
Detecting state coding conflicts in STG unfoldings using SAT.
Application of Concurrency to System Design, 2003. Proceedings. Third International Conference on, pages 51–60, June 2003.

Victor Khomenko.
Model Checking Based on Prefixes of Petri Net Unfoldings.
PhD thesis, University of Newcastle upon Tyne, February 2003.

Ivan Poliakov, Andrey Mokhov, Ashur Rafiev, Danil Sokolov, and Alex Yakovlev.
Automated verification of asynchronous circuits using circuit Petri nets.
Asynchronous Circuits and Systems, 2008. ASYNC '08. 14th IEEE International Symposium on, pages 161–170, April 2008.