Summary

1. Introduction
2. Arbiter design
3. Extending the design
4. Conclusions
Idea of arbitration

Definition

An arbiter is a circuit managing access of other independent 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).
Introduction

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

(Activate) (Request, {Attributes}) (Available, {Attributes}) (Grant) (Grant)
Task specification

- Design the $2 \times 2$ multi-resource arbiter
- Find SI, asynchronous implementation
- Allow concurrent handshakes
- Prevent conflicting handshakes
- Minimise response latency
Task specification

Arbiter design

Clients

C1
C2

R1
R2

H11
H21
H12
H22

2 × 2 Arbiter

Resources

C1g
C2r
R1g
R2g
C1r
C2g
R1r
R2r

H11
H21
H12
H22

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

ASYNC 2009
Arbiter design

Task specification

Clients

\[ \begin{array}{c}
\text{C1} \\
\text{C2}
\end{array} \]

Resources

\[ \begin{array}{c}
\text{R1} \\
\text{R2}
\end{array} \]

2 x 2 Arbiter

\[ \begin{array}{c}
\text{H11} \\
\text{H21} \\
\text{H12} \\
\text{H22}
\end{array} \]
Task specification

Arbiter design

Clients

C1
C2

H11
H21

C1r
C1g
C2r
C2g

R1
R2

H12
H22

2 × 2
Arbiter

Resources

H11→C1g
H12→R1g
C1g→R1r
R1r→C1r
C1r→C1g+
C1g+→C1r-
C2g→C2r-
C2r-→C2g+
C2g+→C2r-
R2g+→R2r-
R2r-→R2g+
R1g+→R1r+
R1r+→R1g-
R1g-→R1r-
R2g+→R2r+
R2r+→R2g-

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

Task specification

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

ASYNC 2009
Resolving conflict
Resolving conflict
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.
Arbiter design

Implementation

![Arbiter Design Diagram]

S. Golubcovs, D. Shang, F. Xia, A. Mokhov, A. Yakovlev
The initial state
All 4 request signals are coming at the same time. The mask is propagating the requests.
Both ME elements receive the signal and do the arbitration.
Client $C_1$ and resource $R_1$ are chosen for the handshake allowing the request controller to make unambiguous decision to activate $H_{11}$ next.
The asymmetric C-element activates outgoing handshake $H_{11}$ and blocks conflicting handshakes on $H_{12}$ and $H_{21}$.
The handshake $H11$ is activated followed by outgoing grant signals on $C1g$ and $R1g$. 
The handshake $H11$ is activated followed by outgoing grant signals on $C1g$ and $R1g$.
Mask hides the initial requests and allows the next pair of requests to propagate. (Suppose the mutual exclusion element on the right side is slower)
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 $H_{21}$ is blocked.
Now the request controller removes the request from the $H_{11}$ and initiates a non-conflicting handshake $H_{22}$.
Arbiter in action

Now the request controller removes the request from the $H_{11}$ and initiates a non-conflicting handshake $H_{22}$.
The signal flow proceeds similarly to $H_{11}$.
The signal flow proceeds similarly to $H_{11}$.
The signal flow proceeds similarly to $H_{11}$
The signal flow proceeds similarly to $H_{11}$
Now the signal is fully propagated and the circuit awaits changes on the inputs.
Modelling is done using UMC90nm technology
Average response latency for first pair of requests:
\(~ 450\text{ps}\)
Worst case for second pair of requests:
\(~ 960\text{ps} \) (if no metastability needs to be resolved)
Best response time for the second pair of requests:
\(~ 450\text{ps}\)
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 analysis on the obtained unfolding (MPSat)
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

- \(2 \times 2\) arbiter is used to propagate tokens
- Clients add and remove tokens from the loop
- Each token represents one resource
- The tokens can be taken concurrently
- Usual passive resources
Extending the design

How can we extend the design?
Extending the design

Overall structure

- Same basic structure as in $2 \times 2$ (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?
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
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).
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 than one handshake (can be decomposed)
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 than one handshake (can be decomposed)
Summary

- :) Found implementation for $2 \times 2$ 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 $\sim 500$ps for each new handshake)
Conclusions

Thank you!

Questions?
Conclusions

Key pictures...
Conclusions

Key pictures 2...

- 2x2 Arbiter
- Interface with the client
- Token in, Token out, Token get, Token put
- 2x2 ring cell
- Client controller
- Mux

S.Golubcovs, D.Shang, F.Xia, A.Mokhov, A.Yakovlev
V. Khomenko, M. Koutny, and A. Yakovlev.
Detecting state coding conflicts in STG unfoldings using SAT.

Victor Khomenko.
*Model Checking Based on Prefixes of Petri Net Unfoldings.*

Ivan Poliakov, Andrey Mokhov, Ashur Rafiev, Danil Sokolov, and Alex Yakovlev.
Automated verification of asynchronous circuits using circuit Petri nets.