
Copyright:

© 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

DOI link to article:

http://dx.doi.org/10.1109/ASYNC.2015.25

Date deposited:

21/07/2015
Opportunistic Merge Element

Andrey Mokhov, Victor Khomenko, Danil Sokolov, Alex Yakovlev
Merge Element

**Purpose:** merge independent requests

**Example:** count the total number of requests

**Property:** requests are never lost, $I_1 + I_2 = 0$

**Requires arbitration**
- between requests
- better outside the critical path
Opportunistic Merge Element

Purpose: merge independent requests, *bundling closely arriving requests* together

Example: respond to an alarm (two sensors)

Property: \( \max(I_1, I_2) \leq O \leq I_1 + I_2 \)
OMs in the real world

Our motivation: on-chip power management
Conceptual specification

Merge
Signal a closes the window of opportunity

The bundle transition has no formal semantics!
Conceptual specification (unrolled)
Conceptual specification (unrolled)
Decomposing the bundle

OM with bundle

Decomposition
Problem: decomposed specification cannot be synthesised due to irreducible state encoding (CSC) conflicts between $s_1$ and $s_4$, and between $s_2$ and $s_3$
Decomposing the bundle

**Problem:** decomposed specification cannot be synthesised due to *irreducible state encoding (CSC) conflicts* between $s_1$ and $s_4$, and between $s_2$ and $s_3$. 

![Diagram showing state transitions and conflicts](Diagram)
Is this a dead end?

Decomposing the bundle \( \{a_1, a_2\} \) is highly non-trivial:

- Output-determinacy violations
- Non-commutativity of inputs
- Irreducible CSC conflicts
- ...
...then a miracle occurs...

I think you should be more specific here in step two.
STG specification

Key idea:
Arbitrate between \{a+, r1+\} and \{a+, r2+\}
CSC resolution (MPSAT)
CSC resolution (MPSAT)

Deadlock free
No hazards
Synthesisable
Fast response: no metastability on the critical path
Synthesised circuit (MPSAT)
Simplified (hacked up) circuit

New optimisation technique: fairness-based optimisation
Simplified (hacked up) circuit
Simplified (hacked up) circuit
Simplified (hacked up) circuit
Simplified (hacked up) circuit
Simplified (hacked up) circuit

Diagram:
- Node ①: r1
- Node ②: m1
- Node ③: gSet
- Node ④: gReset
- Node ⑤: a
- Node ⑤': r2
- Node ⑥: a1
- Node ⑦: r
- Node ⑧: a2
Simplified (hacked up) circuit

\[ r_1 \rightarrow m1 \rightarrow r_1 \quad g1 \rightarrow r_2 \quad g2 \rightarrow gSet \rightarrow 6 \rightarrow a1 \]

\[ a \rightarrow m2 \rightarrow r_2 \quad g2 \rightarrow gReset \rightarrow C \rightarrow r \rightarrow C \rightarrow a2 \]

\[ r_2 \rightarrow m1 \rightarrow r_1 \quad g1 \rightarrow 5 \rightarrow a \rightarrow 5' \]

\[ 6' \]
Simplified (hacked up) circuit

Scenario 1: acknowledgement a wins the arbitration
Scenario 1: acknowledgement a wins the arbitration
Scenario 1: acknowledgement a wins the arbitration
Simplified (hacked up) circuit

Scenario 1: acknowledgement a wins the arbitration
Simplified (hacked up) circuit

Scenario 1: acknowledgement a wins the arbitration
Scenario 1: acknowledgement a wins the arbitration
Simplified (hacked up) circuit

Scenario 1: acknowledgement a wins the arbitration
Scenario 2: request r2 wins the arbitration
Scenario 2: request r2 wins the arbitration
Scenario 2: request r2 wins the arbitration
Scenario 2: request r2 wins the arbitration
Simplified (hacked up) circuit

Scenario 2: request r2 wins the arbitration
Scenario 2: request r2 wins the arbitration
Scenario 2: request r2 wins the arbitration
Scenario 2: request r2 wins the arbitration
Scenario 3: sequential bundling of requests
Simplified (hacked up) circuit

Scenario 3: sequential bundling of requests
Scenario 3: sequential bundling of requests
Scenario 3: sequential bundling of requests
Simplified (hacked up) circuit

Fair mutexes do not permit sequential bundling
Scaling to more inputs
Scaling to more inputs

Can be decomposed
Scaling to more inputs

input channel 1
- r1
- a1
- r
- a

input channel 2
- r2
- a2

input channel 3
- r1
- a1
- r
- a

input channel 4
- r2
- a2

output channel
- r1
- a1
- r
- a
Conclusion

• New reusable asynchronous component – surprisingly difficult for just 3 handshakes!
• Fast implementation – no metastability on critical path
• Discovered *fairness-based optimisation*
• Scalable
• Formally verified using Workcraft and Versify
• To be integrated into a real multiphase buck

• Challenge for asynchronous community:
  
  *Design OM in a non-monolithic way*  
  *(how to design it without a miracle?)*
Thank you!

Opportunistic bundling of questions is encouraged (fairness assumption on the session chair to prevent sequential bundling) 😊