Formal Deadlock Verification of On-Chip Communication Fabrics

Sebastiaan Joosten (TUE Eindhoven / Radboud Nijmegen)
Freek Verbeek (OU Heerlen)
Julien Schmaltz (TUE Eindhoven)
Intel 8 cores
~2.3 Bill. T. on 6.8cm²

AMD Opteron 12 cores
~1.8 Bill. T. on 2x3.46cm²

Sun Niagara3 16 cores
~1 Bill. T. on 3.7cm²

Intel SCC 48 cores
~1.3 Bill. T. on 5.6cm²

Intel 4 cores
~582 Mio. T. on 2.86cm²

Intel Research 80 cores
~100 Mio. T. on 2.75cm²

Tilera TILEPro64 64 cores
~100 Mio. T. on 2.75cm²

Intel 2 cores
~167 Mio. T. on 1.1cm²
Growing complexity of communication fabric

- Heterogeneous topology
- Flow control mechanisms
- Advanced routing function
- Protocol level message dependencies
- Virtual channels
Formal Verification Challenge

We need tools to check for deadlocks
- in large systems
- cross-layer
- parametric
- quick

Formal Verification should not be a side-salad!

Mike Muller - ARM, Inc. (DAC’12 Keynote)
WickedxMAS: a new way for deadlock verification

Deadlock configuration
No deadlock!
Outline

• xMAS
• Deadlocks
• Deadlock Detection
  – In the tool
  – Behind the scenes (short)
xMAS: eXecutable Micro-Architectural Specifications

- Fair sinks and sources
- Fair merges (no starvation)
- Restricted joins
- Synchronous execution semantics
xMAS: example
Examples
Examples

\[ h.y \neq Y \]
\[ h.x = X \]
\[ h.x \neq X \]
\[ h.y = Y \]
\[ h.x > X \]
\[ h.x < X \]
\[ h.y < Y \]
\[ h.y > Y \]
Examples

\[
\begin{align*}
\text{h}.y & \neq Y \\
\text{h}.x & = X \\
\text{h}.x & \neq X \\
\text{h}.y & = Y \\
\text{h}.x & > X \\
\text{h}.x & < X \\
\text{h}.y & < Y \\
\text{h}.y & > Y
\end{align*}
\]
Examples
Deadlocks
Deadlocks

\( \Diamond (c.irdy \wedge \Box \neg c.trdy) \)

The challenge:

Find a configuration in which:
- a packet is permanently stuck
- that is reachable
Deadlock detection

In the tool
WickedxMAS

1. Define packet types
2. Add components
3. Add wires
4. Verify design
• Define packet types
• Add components
• Add wires
• Verify design
WickedxMAS

1. Define packet types
2. Add components
3. Add wires
4. Verify design

Insert types of packets injected at this source.
The domain of all packets is available through PacketDomain.
Eg:
\[ (p \text{ in PacketDomain} | p_X < 5 \& p_Y > 2) \]

Insert function.
The language is a subset of C with math operators +,-,*,/.
Eg:
\[ \text{return } p_X > 10; \]

[Functions and details]

Open Universiteit
www.ou.nl
WickedxMAS

1. Define packet types
2. Add components
3. Add wires
4. Verify design

Diagram with labeled components and wires.
- Define packet types
- Add components
- Add wires
- Verify design

\[
\begin{align*}
\text{(set-logic QF_LIA)} \\
\text{(declare-fun q1 () Int) (assert (<= 0 q1))} \\
\text{(declare-fun q1.type0 () Int) (assert (<= 0 q1.type0))} \\
\text{(declare-fun q1.type1 () Int) (assert (<= 0 q1.type1))} \\
\text{(declare-fun q2 () Int) (assert (<= 0 q2))} \\
\text{(declare-fun q2.type0 () Int) (assert (<= 0 q2.type0))} \\
\text{(declare-fun q2.type1 () Int) (assert (<= 0 q2.type1))} \\
\text{(assert (= 0 (+ (~ q1) q1.type0 q1.type1))} \\
\text{(assert (= 0 (+ (~ q2) q2.type0 q2.type1))} \\
\text{(assert (<= 0 (+ q1 (~ q2.type1))))} \\
\text{(assert (<= 0 (+ q1.type1 (~ q2.type1))))}
\end{align*}
\]
WickedxMAS

1. Define packet types
2. Add components
3. Add wires
4. Verify design
Deadlock detection

Behind the scenes
Deadlock Detection Algorithm

From network...

... to ILP instance ...

Structural
Deadlocks

#q₁.req > 0
#q₂.rsp = 0
#q₁ = #q₁.size

or

#q₂.rsp > 0
#q₁ = 0
#q₂ = #q₂.size

Invariants

#q₁.req = #q₁.size
#q₂.rsp = 0

Sanity
Constraints

#q₁ ≤ #q₁.size
#q₂ ≤ #q₂.size
#q₁ ≥ 0
#q₂ ≥ 0

Typing
Constraints

#q₁ = #q₁.req + #q₁.rsp
#q₂ = #q₂.req + #q₂.rsp

... to solution:

#q₁.req = #q₁.size
#q₂.rsp = 0
Deadlock Detection Algorithm

From network...

... to ILP instance ...

... to solution:

\[
#q_1\text{.req} = #q_1\text{.size} \\
#q_2\text{.rsp} = 0
\]
Deadlock Detection Algorithm

Structural Deadlocks

WiCKeD xMAS UI

Network Inspection

Network Components

Flatten

Flat JSON

fJSON to C

Network as C Library

Invariants

Packet Type Information

Simulate

Invariant Generation

Simplify

Deadlock Detection

SMT Solver (Z3)

Deadlock Detection

Invariants

Open Universiteit www.ou.nl
Conclusion

- WickedxMAS: design & verification tool
  - Automatic deadlock detection
  - xMAS: language on high-level of abstraction
Future Work

• Many small extensions & improvements
  – Enumerations in packet types
  – Better deadlock visualization
  – Better layout of wires
  – Feedback when something goes wrong
• Dealing with unrestricted joins
• Allowing unfair merges
• Adding custom components
• Hierarchical design & verification: networks-of-networks
• Verified compilers from xMAS to Verilog
• Apply to real examples