Synthesizing Formal Models of Hardware from RTL for Efficient Hardware Memory Model and Security Verification

Yao Hsiao¹, Dominic P. Mulligan², Nikos Nikoleris², Gustavo Petri², Caroline Trippel¹

¹Stanford University, ²ARM Research
Memory consistency models warm-up: What can this program print?

- Can it print “hello world”? Yes.
- How about “world”? Yes.
- Print nothing? Yes.

These executions obey Sequential Consistency (SC) [Lamport79], where outcomes correspond to some in-order interleaving of the instructions in individual threads.

Initially, x, y = 0

Core 0
0. x = 1;
1. y = 1;

Core 1
2. if (y == 1) print("hello");
3. if (x == 1) print("world");
Memory consistency models warm-up: What can this program print?

- Initially $x, y = 0$

Core 0
0. $x = 1$;
1. $y = 1$;

Core 1
2. if ($y == 1$) print("hello");
3. if ($x == 1$) print("world");

- And print "hello" only? Depends on MCM!

IBM arm ✓ SC × intel x86

1 2 3 0

Stanford University
Memory consistency models (MCMs) restrict the ordering and visibility of shared memory accesses on a multiprocessor.

Initially $x, y = 0$

Core 0

0 $x = 1$

1 $y = 1$

Core 1

2 if ($y == 1$)

3 if ($x == 1$)

print("hello");

print("world");

How do we verify that a particular microarchitecture complies with the ordering rules defined in its ISA’s MCM?

Can print “hello” only:

$\begin{array}{c}
0 \\
1 \\
\end{array}$

$\begin{array}{c}
1 \\
0 \\
\end{array}$
Check Tools: Automated tools for conducting formal verification of hardware MCM implementations

Axiomatic Model of Microarchitecture (μspec models)

Litmus Test Programs + RequiredResult

Axiom Ld_Exe_Path: forall microops i0, IsAnyRead i0 ⇒ AddEdges [(i0, ff_IF), (i0, ff_DX)), ((i0, ff_DX), (i0, ff_WB))]...

Check Tools¹

Complies with MCM

Violates MCM

¹http://check.cs.princeton.edu
Formally Verifying Hardware with Happens-Before Analysis: Limitations

- (Bounded) Litmus test-based memory consistency model verification.
- Unbounded memory consistency model verification.
- Bounded hardware security verification.

Check Tools

- TriCheck [ASPLOS’17]
- CheckMate [MICRO’18]
- COATCheck [ASPLOS’16]
- PipeCheck [MICRO’15]
- CCICheck [MICRO’15]
- RTLCheck [MICRO’17]
Formally Verifying Hardware with Happens-Before Analysis: Limitations

Check Tools

- TriCheck [ASPLOS’17]
- CheckMate [MICRO’18]
- COATCheck [ASPLOS’16]
- PipeCheck [MICRO’15]
- CCICheck [MICRO’15]
- RTLCheck [MICRO’17]

rtl2μspec: automated synthesis of μspec models from (System)Verilog

rtl2μspec [MICRO’21]
Formally Verifying Hardware with Happens-Before Analysis: Limitations

- (Bounded) **Litmus test-based** memory consistency model verification.
- Unbounded memory consistency model verification.
- Bounded hardware security verification.

Check Tools:
- TriCheck [ASPLOS’17]
- CheckMate [MICRO’18]
- COATCheck [ASPLOS’16]
- PipeCheck [MICRO’15]
- CCICheck [MICRO’15]

rtl2μspec [MICRO’21]

μspec model

Verilog design

Stanford University
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) W[x] = 1;
(i1) W[y] = 1;

Core 1
(i2) R[y] = 1;
(i3) R[x] = 0;

RISC-V multi-V-scale\(^1\)

---

Microarchitectural happens-before (μhb) analysis

Core 0
(i0) W[x] = 1;  (i2) R[y] = 1;
(i1) W[y] = 1;  (i3) R[x] = 0;

Core 1

Forbidden on SC

RISC-V multi-V-scale

μspec models
Microarchitectural happens-before ($\mu$hb) analysis

Core 0

(i0) \(W[x] = 1\);

(i1) \(W[y] = 1\);

Core 1

(i2) \(R[y] = 1\);

(i3) \(R[x] = 0\);

RISC-V multi-V-scale

Forbidden on SC

\[\mu\text{spec models}\]
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) W[x] = 1;
(i1) W[y] = 1;

Core 1
(i2) R[y] = 1;
(i3) R[x] = 0;

RISC-V multi-V-scale

Forbidden on SC

μhb nodes: microarchitectural events, <instruction, state element(s)> pair
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) W[x] = 1; (i2) R[y] = 1;
(i1) W[y] = 1; (i3) R[x] = 0;

Core 1

RISC-V multi-V-scale

Forbidden on SC

μhb edges: happens-before relationships between nodes
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) \( W[x] = 1 \);
(i1) \( W[y] = 1 \);

Core 1
(i2) \( R[y] = 1 \);
(i3) \( R[x] = 0 \);

RISC-V multi-V-scale

Forbidden on SC

Edge type 1: intra-instruction ordering on state updates which correspond to an instruction’s execution path
Microarchitectural happens-before (μhb) analysis

Core 0  Core 1
(i0) \( W[x] = 1 \);  (i2) \( R[y] = 1 \);
(i1) \( W[y] = 1 \);  (i3) \( R[x] = 0 \);

Forbidden on SC

Core 0
(i0) \( W[x] = 1 \);
(i1) \( W[y] = 1 \);

Core 1
(i2) \( R[y] = 1 \);
(i3) \( R[x] = 0 \);

Edge type 2: inter-instruction serialization due to structural dependencies
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) W[x] = 1; (i2) R[y] = 1;
(i1) W[y] = 1; (i3) R[x] = 0;

Core 1

Forbidden on SC

RISC-V multi-V-scale

Edge type 3: inter-instruction ordering due to communication through shared resources
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) W[x] = 1;  (i2) R[y] = 1;
(i1) W[y] = 1;  (i3) R[x] = 0;

Core 1

Forbidden on SC

RISC-V multi-V-scale

μspec models

A cycle in a μhb graph means that the execution is not observable on the microarchitecture
Microarchitectural happens-before (μhb) analysis

Core 0
(i0) \( W[x] = 1; \) (i2) \( R[y] = 1; \)
(i1) \( W[y] = 1; \) (i3) \( R[x] = 0; \)

Core 1

Forbidden on SC

RISC-V multi-V-scale

Microarchitecture specification (μspec) model of a hardware design specifies the space of all possible μhb graphs
Operational Verilog vs. Axiomatic μspec

Operational Verilog
How state elements are updated at each clock cycle

always @(posedge clk) begin
  if (~stall) begin
    ff_DX <= ff_IF;
    ff_WB <= ff_DX;
  end
end

Axiomatic μspec models
Which updates take place on behalf of each instruction (and in which order) + how instructions interact

Axiom Ld_Exe_Path:
for all microops i0,
IsAnyRead i0 ⇒ AddEdges [
((i0, ff_IF), (i0, ff_DX))
((i0, ff_DX), (i0, ff_WB))
].
μspec model synthesis: key research questions

- What are the building blocks of a complete μspec model?

- How do we automatically synthesize a complete and correct-by-construction μspec model from an RTL design?

- Can an automatically synthesized μspec model support efficient MCM verification with the Check Tools?
What are the building blocks of a complete μspec model?

- **μspec model**: a set of rules (or axioms) on instantiating μhb nodes and edges to describe hardware-specific program execution as a μhb graph.
- **μspec axioms**: encode happens-before invariants (HBIs) preserved by temporal Verilog for all executing programs.

```
Core 0
(i0) W[x] = 1;
(i1) W[y] = 1;
```

![Diagram of Intra- and Inter-instruction HBIs](diagram.png)
**Intra-instruction HBIs:** specify for each instruction (1) which state elements it updates, and (2) a partial order on its state updates

One axiom may include more than one HBIs

Axiom \( W \_path: \)

\[
\text{forall microops } i1, \text{ IsAnyWrite } i1 \Rightarrow \\
\text{AddEdges } [\\
((i1, \text{inst\_DX}), (i1, \text{sw\_in\_WB})); \% \text{hbi0} \\
((i1, \text{inst\_DX}), (i1, \text{lw\_in\_WB})); \% \text{hbi1} \\
((i1, \text{inst\_DX}), (i1, \text{wdata})); \% \text{hbi2} \\
((i1, \text{sw\_in\_WB}), (i1, \text{mem})); \% \text{hbi3} \\
((i1, \text{wdata}), (i1, \text{mem})); \% \text{hbi4}
\]

μspec axiom describing the execution path of \textbf{sw} on the RISC-V multi-V-scale
Inter-instruction structural HBIs encode the order in which different instructions update common state elements

- Updates to a common state element or set of elements must be serialized
- What order should these updates be in?

Axiom Structural_Spatial_no_fix_order: 
forall microops i0, i1, 
ProgramOrder i0 i1 ⇒ IsAnyRead i0 ⇒ IsAnyWrite i1 ⇒
AddEdge ((i0, lw_in_WB), (i1, lw_in_WB)) 
\/
AddEdge ((i1, lw_in_WB), (i0, lw_in_WB)).
Inter-instruction dataflow HBIs encode orderings that correspond to instruction communication through shared resources

- One instruction update a state element that is read from and effects the state update of the other instruction

Axiom Dataflow:
\[
\text{forall microops } i_0, i_1, \\[
\text{IsAnyWrite } i_0 \Rightarrow \text{IsAnyRead } i_1 \Rightarrow \\[
\text{SamePA } i_0 \text{ i1 } \Rightarrow \text{SameData } i_0 \text{ i1 } \Rightarrow \\[
\text{NoWritesInBetween } i_0 \text{ i1 } \Rightarrow \text{AddEdge } \\[
((i_0, \text{mem}), (i_1, \text{regfile})). \text{ % hbi0}
\]
rtl2μspec overview

Verilog design

JasperGold

HBI Hypotheses

Proven HBIs

Netlist

μspec

Proven HBIs

Stanford University
Key observation: Hardware-level dataflow relation is a type of happens-before relation
Key observation: Hardware-level dataflow relation is a type of happens-before relation

If an instruction *provably* updates both state elements at the ends of the dataflow relation then the induced dataflow relation forms an HBI.

Dataflow Graph
The dataflow graph representation of RTL design over-approximates intra-instruction HBIs.
Determine which state elements are updated by each instruction type with intra-instruction HBI hypotheses

Intra-instruction HBI hypotheses: Is a load capable of updating regfile during its execution?

Load word set of state elements

Full-design dataflow graph (DFG)
Determine which state elements are updated by each instruction type with intra-instruction HBI hypotheses

Intra-instruction HBI hypotheses:
Is a store capable of updating regfile during its execution?

Full-design dataflow graph (DFG)

State elements updated by **lw**

State elements updated by **sw**
Extract relevant dataflow relations to produce per-instruction dataflow graphs

Full-design dataflow graph (DFG)

DFG for lw

DFG for sw
Per-instructions dataflow graphs imply a full set of intra-instruction HBIs

**DFG for lw**

- **regfile**
- **wdata**
- **inst_DX**
- **lw_in_WB**

**DFG for sw**

- **regfile**
- **wdata**
- **inst_DX**
- **sw_in_WB**
- **lw_in_WB**

---

Intra-instruction HBIs
Per-instructions dataflow graphs imply a full set of intra-instruction HBIs

**DFG for lw**
- regfile
- inst_DX
- lw_in_WB
- mem
- wdata

**DFG for sw**
- regfile
- inst_DX
- lw_in_WB
- mem
- wdata

Intra-instruction HBIs

Store word

Axiom sw_path:
forall microops i0
IsAnyWrite i0 => AddEdge ((i0, wdata), (i1, mem)).
% hbi0
Pairwise compare per-instruction dataflow graphs to generate **structural HBI hypotheses**

**Structural HBI hypotheses:**
For a pair of instructions that are ordered by some reference order, do they always update some common state element in an order consistent with reference order?
Pairwise compare per-instruction dataflow graphs to generate **structural HBI hypotheses**

**Structural HBI hypotheses:**
For all loads \( l \) and all stores \( s \), if \( l \) precedes \( s \) in program order, is \( l \) guaranteed to update \( wdata \) before \( s \)?
Pairwise compare per-instruction dataflow graphs to generate **structural HBI hypotheses**

**HBI Hypotheses**
- Structural dependencies
- Dataflow dependencies

**JasperGold**

**Program order**

Load word specific DFG

Store word specific DFG

**wdata** is updated by both **lw** and **sw**

Full set of proven inter-instruction HBIs
Does the RISC-V multi-V-scale indeed implement **sequential consistency**?

- **rtl2μspec**: evaluates a **correct-by-construction** μspec model with Check Tools
- **RTLCheck**: Verifies Verilog compliance to a **manually written** μspec model with respect to a given litmus test

---

rtl2\textmu spec case study: results

rtl2\textmu spec [MICRO’21]:
\begin{itemize}
  \item Amortized synthesis time: 7.33 seconds
  \item Average Check verification time: 0.03 seconds
  \item 100% proof coverage
\end{itemize}

RTLCheck [MICRO’17]:
\begin{itemize}
  \item Average verification time: 1.61 hours
  \item Incomplete proofs
\end{itemize}
More details in our MICRO ‘21 paper

- Processor design metadata
- Scope of input processor design
  - Assumption: single execution path
- Instantiation of hypotheses in SystemVerilog Assertion (SVAs)
- Artifact evaluation appendix
- New bug in the multi-V-scale
Key Takeaways & Conclusions

- Define what constitutes complete μspec model
  - \( \mu\text{spec} \) = a set of ordering rules
  - Intra-Instruction HBIs
  - Inter-Instruction HBIs

- Propose the \( \text{rtl2μspec} \) approach and tool for \( \mu\text{spec} \) model auto-synthesis
  - \( \text{ rtl2μspec: Incremental synthesis of HBIs proven with JasperGold } \)

- Verify the MCM implementation of RISC-V multi-V-scale with \( \text{rtl2μspec-synthesized } \mu\text{spec model} \)
  - RISC-V multi-V-scale → 6.84 mins (> 780x) → ISA MCM
  - \( \mu\text{spec} \) → Check Tools

Stanford University
Questions?