Verification by the book: ISA Formal at ARM

Will Keen
Senior Engineer, CPU Group
ARM Ltd

TVS Formal Verification Conference
27/06/2017

©ARM 2017
Roadmap

- Problem:
  CPU verification is really hard!

- Verification by the book:
  What is ‘ISA Formal’?

- Success to date:
  What has ISA Formal achieved?

- Future aspirations:
  Where is ISA Formal going next?
Problem:
CPU verification is really hard!
CPU pipelines

- ARM is always looking to optimise execution pipelines for performance
  - Deep pipelining
  - Data forwarding
  - Out-of-order execution
  - Multi-issue/superscalar execution
  - Instruction fusing

- Great for performance…
  - Best-in-class cores

- … but increased design complexity | → |
  - Greater likelihood of design errors
  - Greater number of pathological corner cases
E.g. Cortex-M7

Aggressive Data Forwarding:
- Great for performance
- Nightmare for verification
E.g. Cortex-A75

- (Artist’s impression)
- No, seriously though:
  - Speculative state
  - Out-of-order
  - Dependencies between instructions
Verification impact

- Verification becomes much harder:
  - Hard to think of all corner cases
  - Very hard to stimulate/test all corner cases

- … all adds up to increased likelihood of errata
  - Our worst nightmare
  - Not an option when partners ship billions of units

- Can formal methods help?
  - ‘Breadth-first’ search of BMC will find corner cases…
  - … but needs suitable properties as checkers…

- ISA Formal to the rescue!
Verification by the book: What is ‘ISA Formal’?
Verification by the book

- … where ‘The Book’ is the ARM ARM

- Architecture defines precise semantics of software/hardware contract

- The Dream: Processors verified robustly against ARM Architecture using formal

- … all 5740 pages of it?!
ARM ARM by area

- AArch64 Instruction Set
- AArch32 Instruction Set
- Common pseudocode
- Around 50% of spec
- ISA Formal focuses just on ISA (surprisingly enough…)
Pseudocode definition

- Architecture sees instructions very simply

- Instruction is a function mapping state to next state

\[ \text{State}_0 \xrightarrow{\text{Instr}_0} \text{State}_1 \xrightarrow{\text{Instr}_1} \text{State}_2 \xrightarrow{\text{Instr}_2} \ldots \xrightarrow{\text{Instr}_N} \text{State}_{N+1} \]

- Every instruction specified by executable pseudocode
  - E.g. \texttt{ADD x0, x1, x2} (N.B. the below is heavily simplified)

\[ \text{State}_0 \xrightarrow{\text{ADD x0,x1,x2}} \text{State}_1 \]

\[ \text{State}_1.x0 = \text{State}_0.x1 + \text{State}_0.x2 \]
Implementation of architecture

- Real implementation not so straightforward!

- Many instructions simultaneously, ‘speculative’ views of architectural state

- But can we extract architectural state from real pipeline?
  - $State_N$: state before $Instr_N$ executes
  - $Instr_N$: opcode of instruction
  - $State_{N+1}$: state after $Instr_N$ executes
Don’t care about pipeline

All we need is:
- $Instr_N$: Observed opcode in WR (tracked down pipeline)
- $State_N$: Observed register values (from register file)
- $State_{N+1}$: Observed register writes
- Logic to give expected values
Architecture Explorer (ArchEx)

- ARM in-house tool developed by Alastair Reid, ARM Research

- Originally interpreter to execute ARM ARM pseudocode

- Translates pseudocode to combinational SystemVerilog (no flops, no persistent state)
ISA Formal in practice

DUT (CPU RTL)  

Bridge Component
- $State_N$ (PRE)
- $Instr_N$
- $State_{N+1}$ (POST)

ArchEx VIP
- Autogenerated SystemVerilog
- ARM Architecture
- $State_{N+1}$ (ARCH)
- Properties

Properties

Autogenerated SystemVerilog
ARM Architecture
Extracting architectural state (bridge)

- Difficult – µArch often obscures architecture for performance
  - E.g. view of memory, Out-of-Order completion, multi-issue, fusion etc…

- But possible – lots of low-hanging fruit
  - E.g. values in register file are architecturally committed x/w[0-30] values
  - Everything possible with hard work and tracking logic

- Demonstrated to be possible on different ARM cores
  - A-class – big/LITTLE, in-order/out-of-order
  - R-class – subset of A-class
  - M-class – hardware-level exception/interrupt handling, security
ISA Formal – why it works

- Checks architectural functionality
  - E.g. does an ADD work properly?
  - Useful for bringup
  - Finds architectural corners (e.g. ARMv8-M on Cortex-M33)

- By implication, checks μArchitectural functionality
  - E.g. integer arithmetic with data forwarding
  - In CPU pipeline, done based on forwarding (speculative architectural state)
  - In ISA Formal, done based on register file (committed architectural state)
  - If final register write mismatches due to bad forwarding, we get a counterexample

- Works the same way for all ‘committed’ vs ‘speculative’ architectural state
  - E.g. branch prediction/resolution, conditional execution, pointer-chasing, fusion, OoO, etc.
ISA Formal – why it works (contd.)

- ArchEx architectural model
  - Autogenerated – straight from spec to checker, no human misinterpretation
  - Stateless – good for formal tool
  - Reusable – many projects work off same VIP generation flow
  - Coverage – auto-generate required coverage tracking information for signoff

- Applying formal verification
  - Corner cases – tool using ‘breadth-first’ search finds bugs dynamic TBs don’t
  - Incremental checks – can begin applying checks as instructions are brought up
  - Assume/guarantee – properties per-instruction, based on current state of RTL
Success to date:
What has ISA Formal achieved?
## ARM CPUs with ISA Formal

<table>
<thead>
<tr>
<th>A-class</th>
<th>R-class</th>
<th>M-class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-A53</td>
<td>Cortex-R52</td>
<td>Cortex-M4</td>
</tr>
<tr>
<td>Cortex-A32</td>
<td>Next generation</td>
<td>Cortex-M7</td>
</tr>
<tr>
<td>Cortex-A35</td>
<td></td>
<td>Cortex-M33</td>
</tr>
<tr>
<td>Cortex-A55</td>
<td></td>
<td>Next generation</td>
</tr>
<tr>
<td>Next generation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Rolling out globally to other design centres

- Sophia, France – Cortex-A75 (partial)
- Austin, USA – TBA
- Chandler, USA – TBA
ISA Formal case study

- Example project: Cortex-M33
  - New Architecture (ARMv8-M)
  - New μArchitecture

- Effort: one Graduate Engineer
  - Applied mid-project cycle

- Successes:
  - Many unique bugs, not found by simulation TB
  - Compute-efficient
  - Resource-efficient (team size)
Cortex-M33 ISA Formal bug curve
Cortex-M33 bugs/week per methodology
Cortex-M33 CPU hours/bug per methodology
Further thoughts

- Quality of bugs, not just quantity
  - Lots of horrible architectural (and μArchitectural) corner cases
  - End-to-end nature finds many issues you wouldn’t expect

- High RoI
  - Mostly done by less experienced engineers, who become ‘experts’
  - Also boosts Sim TB guys with cross-communication on issues found

- Highly portable
  - All ARM architecture profiles covered
  - Very small to very large cores (M4–A75)
Future aspirations:  
Where is ISA Formal going next?
Sharing our success

- Deployment to all ARM CPU projects
  - Happening as we speak!

- ‘Democratising’ ISA Formal expertise
  - Experts providing training/mentoring
  - Goal – ‘push-button’ flow
  - Enable non-experts to deploy on future projects

- Consolidating ISA Formal collateral
  - Tool is maintained by ARM Technology Services Group (Daryl Stewart)
  - Enables easier deployment on projects
  - Enables leveraging improvements
Broadening our horizons...

- ISA is just one part of architecture
  - Can we deploy further?
  - ‘Architecture Formal’

- Identifying other candidates for similar treatment
  - Memory model
  - MMU
  - Decoders
  - Debug
  - Etc.

- Spreading out to other groups within ARM
  - Encourage automated spec → formal VIP work across the company
Summary
ISA Formal summary

- Verification by the book
  - Automatically generate Formal VIP from the architecture

- Proven bug-hunting track record
  - Has found bugs on all applications before product release

- Vast scope of possibility
  - ‘Architecture Formal’ could be far-reaching in impact
Questions?