Who guards the guards?

Formal validation of the Arm v8-M Architecture Specification

Alastair Reid
Arm Research
@alastair_d_reid

© 2017 Arm Limited
Uses of formal processor specifications

Writing compilers, operating systems, ...
Formally verifying compilers, operating systems, ...
Program synthesis
Security analysis
Malware analysis
Formally verifying processor implementations
The state of most processor specifications

Large (1000s of pages)

Broad (10+ years of implementations, multiple manufacturers)

Complex (exceptions, weak memory, ...)

Informal (mostly English prose)

We are all just learning how to (retrospectively) formalize specifications
Arm Processor Specifications

**A-class** (phones, tablets, servers, ...)  
- 6,000 pages  
- 40,000 line formal specification  
- Instructions (32/64-bit)  
- Exceptions / Interrupts  
- Memory protection  
- Page tables  
- Multiple privilege levels  
- System control registers  
- Debug / trace

**M-class** (microcontrollers, IoT)  
- 1,200 pages  
- 15,000 line formal specification  
- Instructions (32-bit)  
- Exceptions / Interrupts  
- Memory protection  
- Page tables  
- Multiple privilege levels  
- System control registers  
- Debug / trace
Is my specification correct?
Is my specification correct?
Is my specification correct?

Formal Validation of CPUs

Testing
Is my specification correct?

- Formal Validation of CPUs
- Multiple Users
- Testing
Executable Specification

Defines what *is* allowed

- **Animation** → Check spec matches expectation
- **Testable** → Compare spec against implementation
Executable Specification

Defines what *is* allowed

- Animation → Check spec matches expectation
- Testable → Compare spec against implementation

Does *not* define what *is not* allowed

- e.g., Impossible states, impossible actions/transitions, security properties
- No redundancy
- Problem when extending specification
Creating a specification of disallowed behaviour

Where to get a list of disallowed behaviour?

How to formalise this list?

How to formally validate specification against spec of disallowed behaviour?

(This may look familiar from formal specification of software)
Execute
Halted = FALSE

Execute

Halted = TRUE

Debug
Halt
LockedUp = FALSE

Halted = FALSE

Execute

Debug

Halt

LockedUp = TRUE

Halted = TRUE

Lockup

Debug

Lockup
LockedUp = FALSE

Halted = FALSE

Execute

LockedUp = TRUE

Halted = TRUE

Debug Halt

Debug Lockup
Rule JRJC

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority processor exception.
Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D
Rule R

State Change X is by any of the following:

• Event A
• Event B
• State Change C
• Event D

And cannot happen any other way
Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D

And cannot happen any other way

Rule R: \( X \rightarrow A \lor B \lor C \lor D \)
<table>
<thead>
<tr>
<th>State Change X</th>
<th>Exit from lockup</th>
<th>Fell(LockedUp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event A</td>
<td>A Cold reset</td>
<td>Called(TakeColdReset)</td>
</tr>
<tr>
<td>Event B</td>
<td>A Warm reset</td>
<td>Called(TakeReset)</td>
</tr>
<tr>
<td>State Change C</td>
<td>Entry to Debug state</td>
<td>Rose(Halted)</td>
</tr>
<tr>
<td>Event D</td>
<td>Preemption by a higher priority processor exception</td>
<td>Called(ExceptionEntry)</td>
</tr>
</tbody>
</table>
Fell(LockedUp) → Called(TakeColdReset)
   ∨ Called(TakeReset)
   ∨ Rose(Halted)
   ∨ Called(ExceptionEntry)
Rule VGNW

Entry to lockup from an exception causes

- Any Fault Status Registers associated with the exception to be updated.
- No update to the exception state, pending or active.
- The PC to be set to 0xFFFFF000.
- EPSR.IT to become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.
v8-M Spec

Rules

Z3 SMT Solver

Proof

Counterexample
Temporal Operators

Fell(LockedUp) → Called(TakeColdReset)

∨ Called(TakeReset)

∨ Rose(Halted)

∨ Called(ExceptionEntry)
Fell(LockedUp) → Called(TakeColdReset)

✓ Called(TakeReset)
✓ Rose(Halted)
✓ Called(ExceptionEntry)
Fell(LockedUp) → Called(TakeColdReset)
✓ Called(TakeReset)
✓ Rose(Halted)
✓ Called(ExceptionEntry)
Temporal Operators

- **Fell(e)**
  - Past(e) > e

- **Stable(e)**
  - Past(e) = e

- **Rose(e)**
  - Past(e) < e
Temporal Operators

Fell(LockedUp)

\[
\text{\_\_Past\_LockedUp} = \text{LockedUp};
\]

\[
\text{FunctionUnderTest();}
\]

\[
\ldots \text{\_\_Past\_LockedUp} > \text{LockedUp} \ldots
\]
Event Operators

Called(TakeReset)

TakeReset()
{
    __Called_TakeReset = TRUE;
    ...
}

Rule JRJC
Exit from lockup is by any of the following:
• A Cold reset.
• A Warm reset.
• Entry to Debug state.
• Preemption by a higher priority processor exception.

Fell(LockedUp) → Called(TakeColdReset)
  ∨ Called(TakeReset)
  ∨ Rose(Halted)
  ∨ Called(ExceptionEntry)

__Called_TakeColdReset = FALSE;
__Called_TakeReset = FALSE;
__Called_TakeExceptionEntry = FALSE;
__Past_LockedUp = LockedUp;
__Past_Halted = Halted;

assert((__Past_LockedUp > LockedUp) =>
(  __Called_TakeColdReset
 || __Called_TakeReset
 || __Past_Halted < Halted
 || __Called_ExceptionEntry));
Arm Specification Language

- Arithmetic operations
- Boolean operations
- Bit Vectors
- Arrays
- Functions
- Local Variables
- Statements
  - Assignments
  - If-statements
  - Loops
  - Exceptions

SMT

- Arithmetic operations
- Boolean operations
- Bit Vectors
- Arrays
- Functions
- Local Variables
- Statements
  - Assignments
  - If-statements
  - Loops
  - Exceptions
Results (more in paper)

Most properties proved in under 100 seconds

Found 12 bugs in specification:
- debug, exceptions, system registers, security

Found bugs in English prose:
- ambiguous, imprecise, incorrect, ...
Summary

Formalization of large, complex specifications

Executable specifications have a fatal flaw

Need specification of disallowed behaviour

Manually formalized structured English prose

Used SMT checker to find bugs in both spec and prose
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

See also:
“Trustworthy Specifications of the ARM v8-A and v8-M architecture,” FMCAD 2016
“End to End Verification of ARM processors with ISA Formal,” CAV 2016