How Can You Trust Formally Verified Software?

Alastair Reid

Arm Research
@alastair_d_reid
Formal verification

Of libraries and apps
- Verified Software Toolchain

Of compilers
- COMPCERT
- Vellvm (verified LLVM)
- CAKEML (A Verified Implementation of ML)

Of operating systems
- sel4 (Security, Performance, Proof)
- CertiKOS
Fonseca et al., An Empirical Study on the Correctness of Formally Verified Distributed Systems, Eurosys ‘17
Formally Verified Software

- Verification Tool
- Shim Code
- Formal Specifications

Fonseca et al., An Empirical Study on the Correctness of Formally Verified Distributed Systems, Eurosys ‘17
Takeaway #1: 3 key questions to ask

1. What specifications does your proof rely on?
2. Why do you trust those specifications?
3. Does anybody else use these specifications?
Takeaway #2: Specifications must have multiple uses
Takeaway #2: Specifications must have multiple uses
How can you trust formally verified software?

Specifications are part of the TCB

3 key questions

Specifications must have multiple users

How can you trust formal specifications?

Testing specifications

Verifying processors

Verifying specifications

How can you trust formally verified software?
Creating trustworthy specifications

“Trustworthy Specifications of the ARM v8-A and v8-M architecture,” FMCAD 2016
Arm Architecture Reference Manual (ARMARM)

32-bit / 64-bit Instructions
Exceptions / Interrupts
Privilege / Security
Virtual Memory
System registers
Debug / Trace
Profiling
...

Pages: 0, 1600, 3200, 4800, 6400
Years: 1996, 2007, 2018
Exit from lockup is by any of the following:

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

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 0xFFFFFFFFE.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.
Pseudocode

Encoding A1

ARMv4*, ARMv5T*, ARMv6*, ARMv7
ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}

<table>
<thead>
<tr>
<th>cond</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>imm5</th>
<th>type</th>
<th>0</th>
<th>Rm</th>
</tr>
</thead>
</table>

if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd); n = UInt(Rn); m = UInt(Rm); setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);

if ConditionPassed() then
    EncodingSpecificOperations();
    shifted = Shift(R[m], shift_t, shift_n, APSR.C);
    (result, carry, overflow) = AddWithCarry(R[n], shifted, APSR.C);
    if d == 15 then // Can only occur for ARM encoding
        ALUWritePC(result); // setflags is always FALSE here
    else
        R[d] = result;
        if setflags then
            APSR.N = result<31>;
            APSR.Z = IsZeroBit(result);
            APSR.C = carry;
            APSR.V = overflow;
Arm Architecture Specification Language (ASL)

Indentation-based syntax

Imperative

First-order

Strongly typed (type inference, polymorphism, dependent types)

  Bit-vectors

  Unbounded integers

  Infinite precision reals

  Arrays, Records, Enumerations

Exceptions
ASL Spec -> Lexer Parser Typechecker -> C Backend Interpreter
Architectural Conformance Suite

Processor architectural compliance sign-off

Large

• v8-A 11,000 test programs, > 2 billion instructions
• v8-M 3,500 test programs, > 250 million instructions

Thorough

• Tests dark corners of specification
Testing Pass Rate
(Artists Impression)
Measuring architecture coverage of tests

Untested: op1*op2 == -3.0, FPCR.RND=-Inf

```c
bits(N) FPRsrtStepFused(bits(N) op1, bits(N) op2)
    assert N IN {32, 64};
    bits(N) result;
    op1 = FPNeg(op1); // per FMSUB.FMLS
    (type1,sign1,value1) = FPUnpack(op1, FPCR);
    (type2,sign2,value2) = FPUnpack(op2, FPCR);
    (done,result) = FPProcessNaNs(type1, type2, op1, op2, FPCR);
    if (!done then
        inf1 = (type1 == FPType_Infinity);
        inf2 = (type2 == FPType_Infinity);
        zero1 = (type1 == FPType_Zero);
        zero2 = (type2 == FPType_Zero);
        if (inf1 && zero2 || (zero1 && inf2) then
            result = FPOnePointFive("0");
        else if inf1 || inf2 then
            result = FPInfinity(sign1 EOR sign2, N);
        else
            // Fully fused multiply-add and halve
            result_value = (3.0 + (value1 * value2)) / 2.0;
            if result_value == 0.0 then
                // Sign of exact zero result depends on rounding mode
                sign = if FPCR.Rounding() == FPRounding_NEGINF then '1' else '0';
                result = FPZero(sign, N);
            else
                result = FPRound(result_value, FPCR.Rounding());
            return result;
    ```
Formal verification of processors
Checking an instruction

ADD
Checking an instruction

Context

CMP  LDR  ADD  STR  BNE
Memory

R0 – R15

Decode

IF

ID

EX

MEM

WB
ARM Research

Combinational Verilog

Architecture Specification

ASL to Verilog

Specialize
Monomorphize
Constant Propagation
Width Analysis
Exception Handling
...

Combinational Verilog
Arm CPUs verified 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
Formal validation of specifications
Suppose...

Last year: audited all accesses to privileged registers
• Specification: Added missing privilege checks
• Testsuite: Added new tests to test every privilege check
• Formal testbench: Verify every check

This year: add new instruction but accidentally omit privilege check

How many tests in the test suite will fail on new specification?
Can we formally verify specification?

Specification of the specification

- Disallowed behaviour
- Invariants
- Cross-cutting properties

Tools that can prove properties of ASL specifications
Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority exception.
State change: Exit from lockup is by any of the following:
- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority exception.
State change: Exit from lockup is by any of the following:

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

Event
Exit from lockup is by any of the following:

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

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

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

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

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

**rule** lockup_exit

```plaintext
assume Fell(LockedUp);
Called(TakeColdReset)
∨ Called(TakeReset)
∨ Rose(InDebugState())
∨ Called(ExceptionEntry);
```
Converting ASL to SMT

Functions
Local Variables
Statements
  Assignments
  If-statements
Exceptions
Arithmetic operations
Boolean operations
Bit Vectors
Arrays
Formally Validating Specifications

v8-M Spec → Verification

Property → Verification

CEX → Bug in Spec

Proof
Formally Validating Specifications

v8-M Spec → Verification

Property → Verification

12 Bugs Found so far

CEX → Proof
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 0xFFFFFFFFE.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.
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 0xEFFFFFFE.
- EPSR.IT to be become UNKNOWN.

In addition, **HFSR.FORCED** is not set to 1.

```
rule lockup entry
  assume Rose(LockedUp);
  assume ¬Called(TakeReset);

property a HaveMainExt() ⇒ CFSR != 0;
property b1 Stable(ExnPending);
property b2 Stable(ExnActive);
property c PC == 0xEFFFFFFFE;
property e HFSR.FORCED == 0;
```
Entry to lookup 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 \(0x\text{EFFFFFFFFFFE}\).
- EPSR.IT to be become UNKNOWN.

In addition, \textbf{HFSR.FORCED} is not set to 1.

\begin{verbatim}
rule lockup entry
  assume Rose(LockedUp);
  assume ¬Called(TakeReset);
  property a HaveMainExt() ⇒ CFSR != 0;
  property b1 Stable(ExnPending);
  property b2 Stable(ExnActive);
  property c PC == 0x\text{EFFFFFFFFFFE};
  property e HFSR.FORCED == 0;
  Stable(HFSR.FORCED);
\end{verbatim}
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 0xEFFFFFFE.
- **EPSR.IT** to be become **UNKNOWN**.

In addition, **HFSR.FORCED** is not set to 1. is not changed.

```plaintext
rule lockup entry
  assume Rose(LockedUp);
  assume ¬Called(TakeReset);

property a HaveMainExt() ⇒ CFSR !≠ 0;
property b1 Stable(ExpPending);
property b2 Stable(ExpActive);
property c PC == 0xEFFFFFFFE;
property e HFSR.FORCED !⇒ 0;
```

Debug view of

Public release of machine readable Arm specification

Enable formal verification of software and tools

Releases

  April 2017: v8.2
  July 2017: v8.3

Working with Cambridge University REMS group to convert to SAIL

  Backends for HOL, OCaml, Memory model, (hopefully Coq too)

Tools: https://github.com/alastairreid/mra_tools

Talk to me about how I can help you use it
Potential uses of processor specifications

Verifying compilers
Verifying OS page table / interrupt / boot code
Verifying processor pipelines
Verification and discovery of peephole optimizations
Automatic generation of binary translators
Automatic generation of test cases
Decompilation of binaries
Abstract interpretation of binaries
etc.
How can you trust formally verified software?
How can you trust formal specifications?

Test the specifications you depend on

Ensure specifications have multiple uses

Create meta-specifications

https://xkcd.com/1416/
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

“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
“Who guards the guards? Formal Validation of ARM v8-M Specifications” OOPSLA 2017