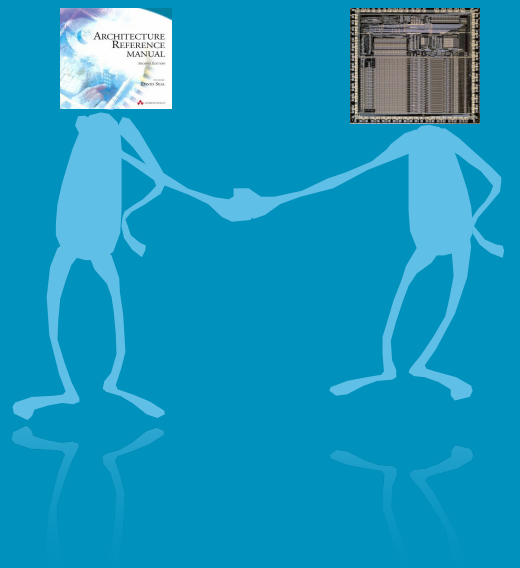


Creating Formal Specifications of the Arm Processor Architecture

Alastair Reid

Arm Research

@alastair_d_reid

© 2017 Arm Limited

Why should anybody care about formal specifications?

- Precise / unambiguous
- Enables formal reasoning about implementations

© 2017 Arm Limited

Real World Artifacts

Linux Kernel, C compilers, ARM processors, TCP/IP, WiFi, etc.

- Multiple implementations, suppliers, versions, configurations
- Important: commercial, security, …
- Long history, initial spec informal
- Formal spec not 100% welcome
- Backwards compatibility requirements
- Spec must include all quirks of recent versions of major implementations to be useful
- Conformance suites?

© 2017 Arm Limited

ARM Architecture - an example real-world artifact

Arm

- Founded in 1990
- Designs processors
- Designs architecture
- Licenses architecture
- 16B processors / year
- (also GPUs, IoT, …)

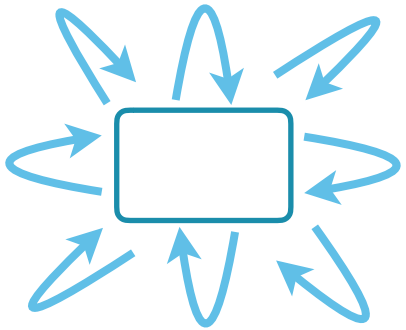
© 2017 Arm Limited

Current status of ARM specifications

- Formal specifications of A, R and M-class processor classes exist
- Integrated into ARM's official processor specifications
- Maintained by ARM's architecture team
- Used by multiple teams within ARM
  - Formal validation of ARM processors using Bounded Model Checking
  - Development of test suites
  - Designing architecture extensions
  - …
- Publicly released in machine readable form

© 2017 Arm Limited

Overview



1. What's different about Real World Artifacts?
2. ARM's formal processor specifications
   - Three experiences
   - Lessons learned
3. Conclusions

"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
https://alastairreid.github.io/papers/

© 2017 Arm Limited
arm

Creating trustworthy specifications

© 2017 Arm Limited

arm

Unstructured English Prose (A-class spec)

Concurrent modification and execution of instructions

The ARMv8 architecture limits the set of instructions that can be executed by one thread of execution as they are being modified by another thread of execution without requiring explicit synchronization.

Concurrent modification and execution of instructions can lead to the resulting instruction performing any behavior that can be achieved by executing any sequence of instructions that can be executed from the same Exception level, except where each of the instruction before modification and the instruction after modification is one of a `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, or `SVC` instruction.

For the `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, and `SVC` instructions the architecture guarantees that, after modification of the instruction, behavior is consistent with execution of either:

- The instruction originally fetched.
- A fetch of the modified instruction.

If one thread of execution changes a conditional branch instruction, such as `B` or `BL`, to another conditional instruction and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

© 2017 Arm Limited

Semi-structured English prose (M-class spec)

R_{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 exception.

R_{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 `0xEFFFFFFE`.
- EPSR.IT to be become UNKNOWN.

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

© 2017 Arm Limited
arm

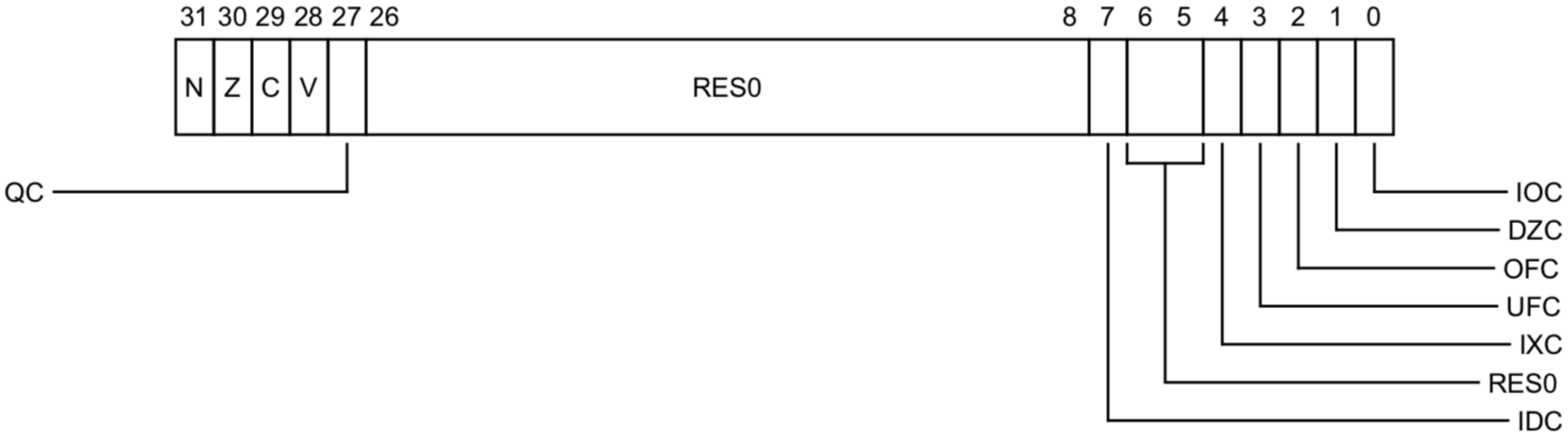
Tables - semistructured, not machine readable

Table B2-1 Encoding of the DMB and DSB <option> parameter

Accesses		Shareability domain			
Before the barrier	**After the barrier**	**Full system**	**Outer Shareable**	**Inner Shareable**	**Non-shareable**
Reads and writes	Reads and writes	SY	OSH	ISH	NSH
Writes	Writes	ST	OSHST	ISHST	NSHST
Reads	Reads and writes	LD	OSHLD	ISHLD	NSHLD

© 2017 Arm Limited
arm

Registers - structured, machine-readable



N, bit [31]

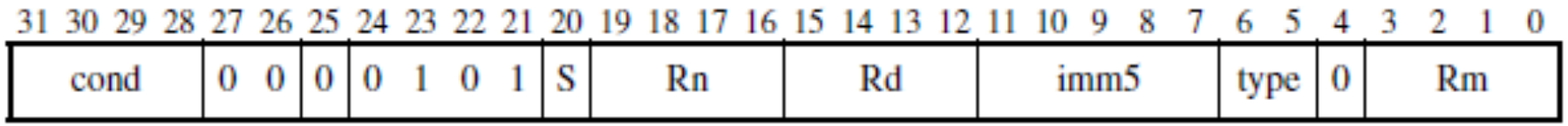
Negative condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.N flag instead.

Z, bit [30]

Zero condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.Z flag instead.

Pseudocode

```
ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
cond	0 0	0	0 1 0 1	S	Rn	Rd	imm5	type	0	Rm

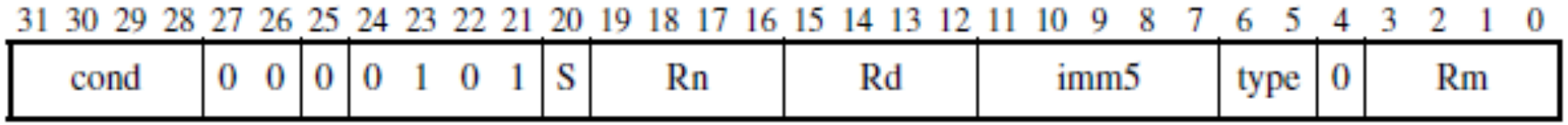
```
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);

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;
```


© 2017 Arm Limited
arm

Pseudocode

```
ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
cond	0 0	0	0 1 0 1	S	Rn	Rd	imm5	type	0	Rm

```
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;
```

- Type Inference
- Unbounded Integers
- Enumerations
- Bit Vectors
- Indentation-based Syntax
- Dependent Types
- Imperative
- Exceptions

© 2017 Arm Limited
arm

Status at the start

- No tools (parser, type checker)
- Incomplete (around 15% missing)
  - “Specify by comment”
  - Many trivial errors (that confuse tools but not humans)
- Unexecuted, untested
- Experts believed that an executable spec was
  - Impossible
  - Not useful
  - Less readable
  - Less correct

© 2017 Arm Limited

Architectural Conformance Suite

Processor architectural compliance sign-off

Large

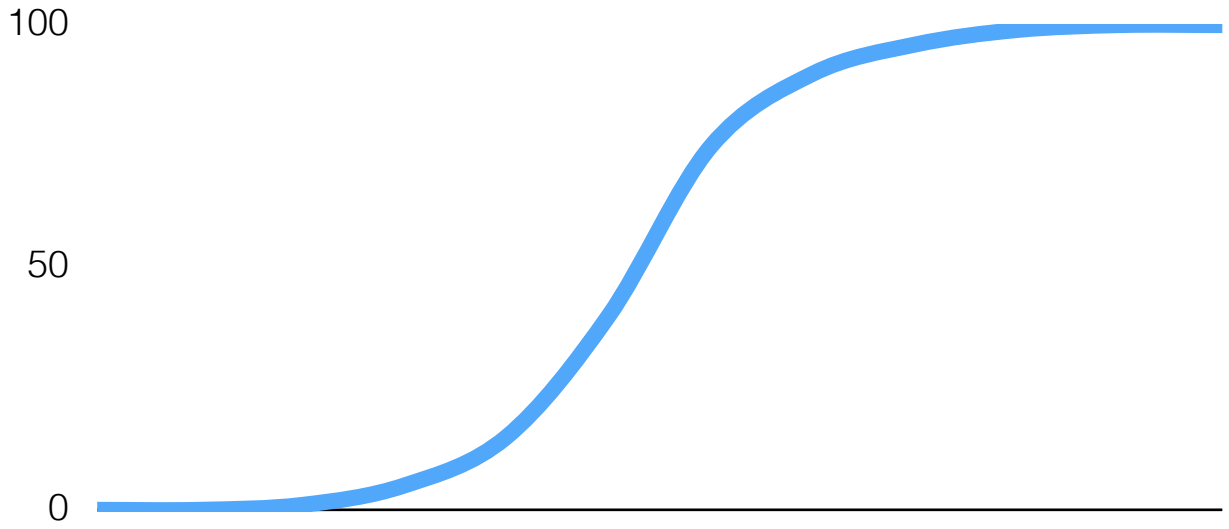
- v8-A 11,000 test programs, > 2 billion instructions
- v8-M 3,500 test programs, > 250 million instructions

Thorough

- Tests dark corners of specification

© 2017 Arm Limited
arm

Progress in testing Arm specification



- Does not parse, does not typecheck
- Can’t get out of reset
- Can’t execute first instruction
- Can’t execute first 100 instructions
- …
- Passes 90% of tests
- Passes 99% of tests
- …

© 2017 Arm Limited

arm

Measuring architecture coverage of tests

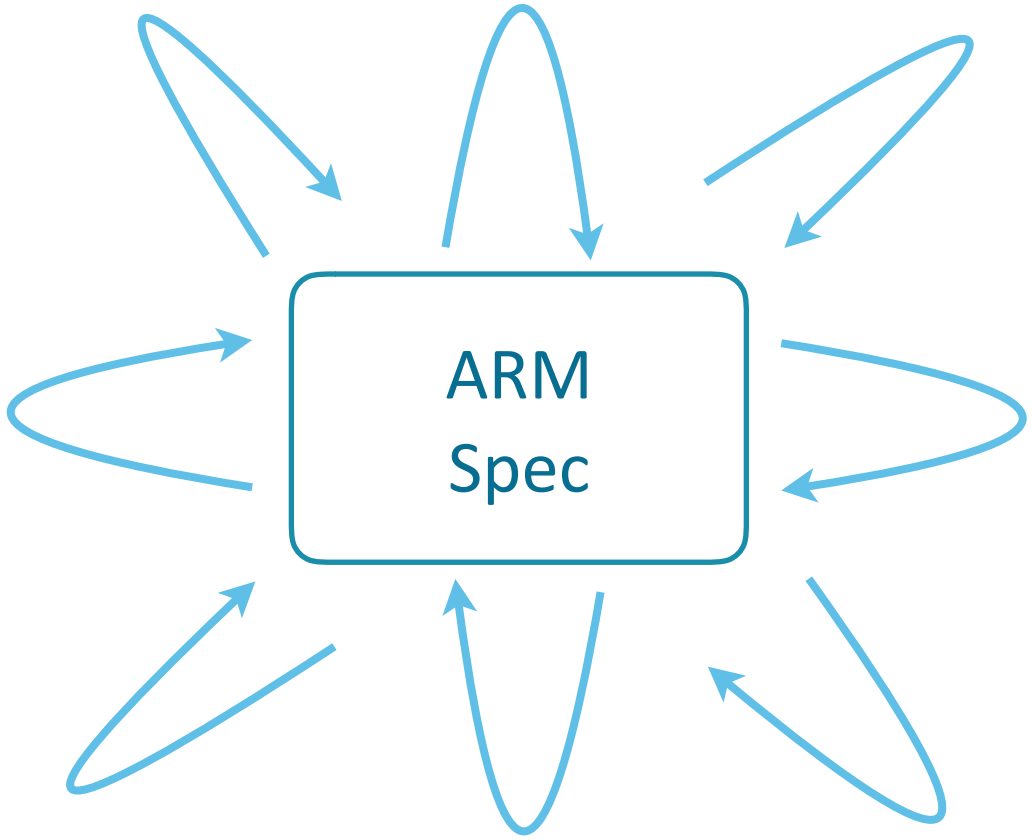
Untested: op1*op2 == -3.0, FPCR.RND=-Inf


```
bits(N) FPRSqrtStepFused(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');
        elsif 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 FPCRRounding() == FPRounding_NEGINF then '1' else '0';
                result = FPZero(sign, N);
            else
                result = FPRound(result_value, FPCRRounding());
    return result;
```


© 2017 Arm Limited

arm

Creating a Virtuous Cycle



© 2017 Arm Limited
arm

Lessons learned

- Specifications contain bugs
- Huge value in being able to run existing test suites
  - Need to balance against benefits of non-executable specs
- Find ways to provide direct benefit to other users of spec
  - They will do some of the testing/debugging for you
  - They will support getting your changes/spec adopted as master spec
  - Creates Virtuous Cycle

© 2017 Arm Limited

"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

Formal validation of processors and of specifications

© 2017 Arm Limited

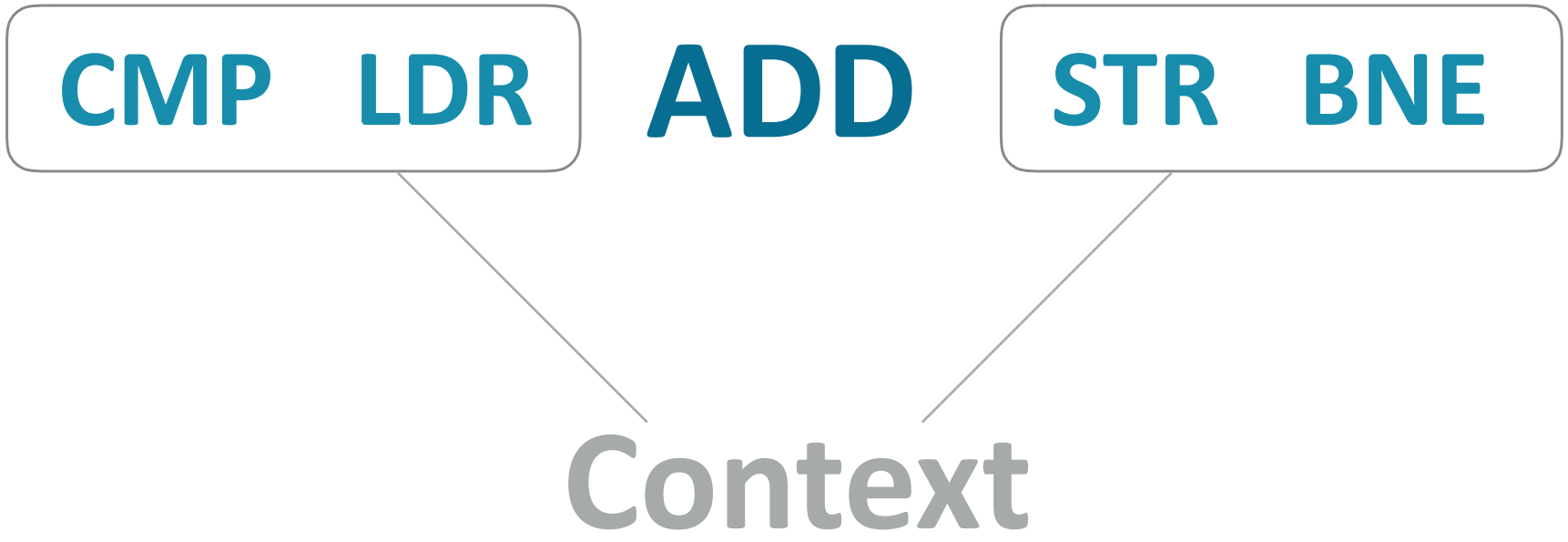
arm

Checking an instruction

ADD

© 2017 Arm Limited

Checking an instruction



© 2017 Arm Limited
arm

Lessons Learned from validating processors

- Very effective way to find bugs in implementations
- Formally validating implementation is effective at finding bugs in spec
  - Try to find most of the bugs in your spec before you start
- Huge value in being able to use spec to validate implementations
  - Helps get formal specification adopted as part of official spec

© 2017 Arm Limited

“Eyeball Closeness”

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.

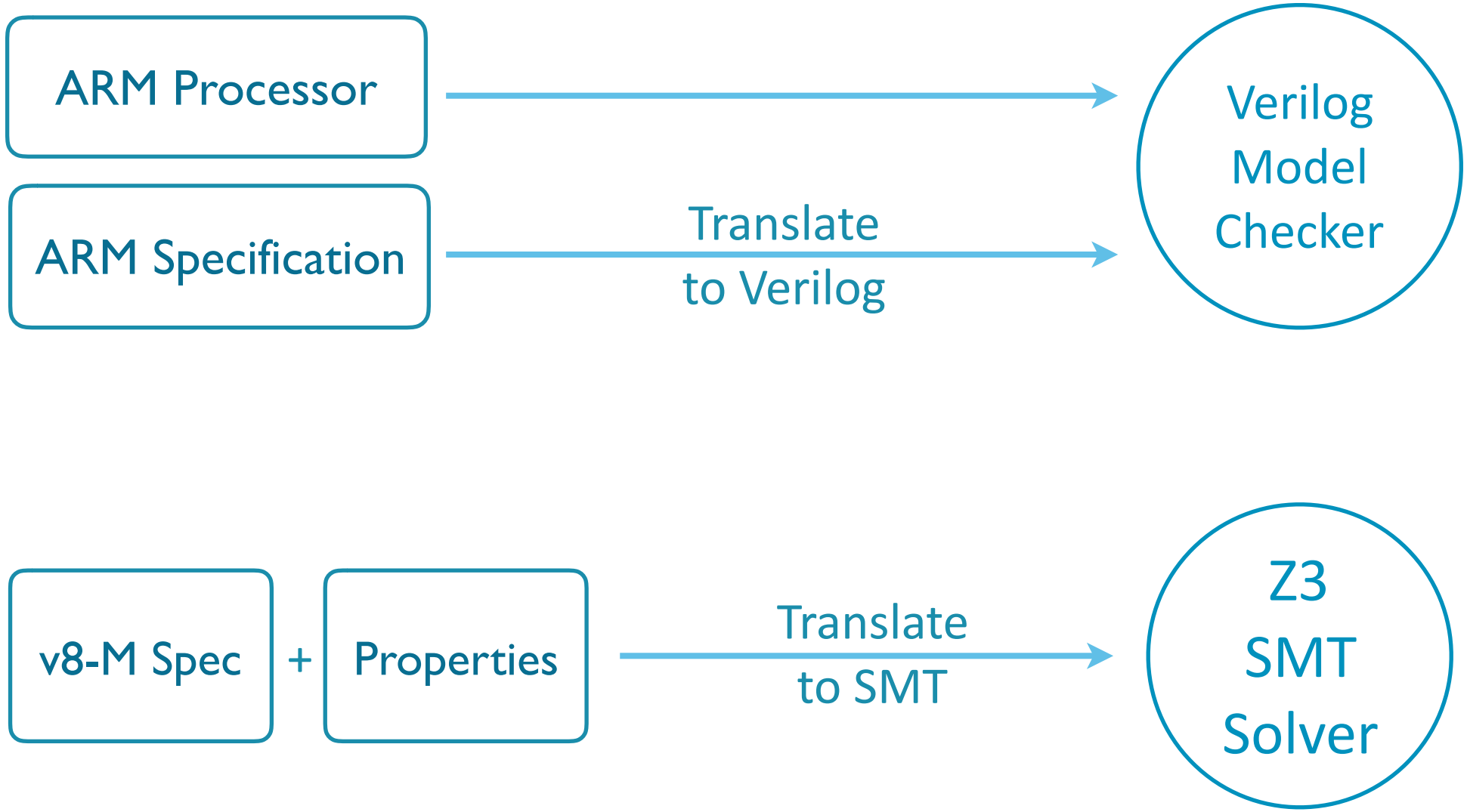
$$
\begin{aligned}
&\mathrm{Fell}(\mathrm{LockedUp}) \\
&\rightarrow \\
&\quad \mathrm{Called}(\mathrm{TakeColdReset}) \\
&\quad \vee\ \mathrm{Called}(\mathrm{TakeReset}) \\
&\quad \vee\ \mathrm{Rose}(\mathrm{Halted}) \\
&\quad \vee\ \mathrm{Called}(\mathrm{ExceptionEntry})
\end{aligned}
$$

© 2017 Arm Limited

Lessons Learned from formally validating specifications

- Redundancy essential for detecting errors
  - Detected subtle bugs in security, exceptions, debug, …
  - Found bugs in English prose
- Need set of ‘orthogonal’ properties
  - Invariants, Security properties, Reachability properties, etc.
- Eyeball closeness

© 2017 Arm Limited



© 2017 Arm Limited
arm

Creating Formal Specifications of Real World Artifacts

Plan for adoption into official specs

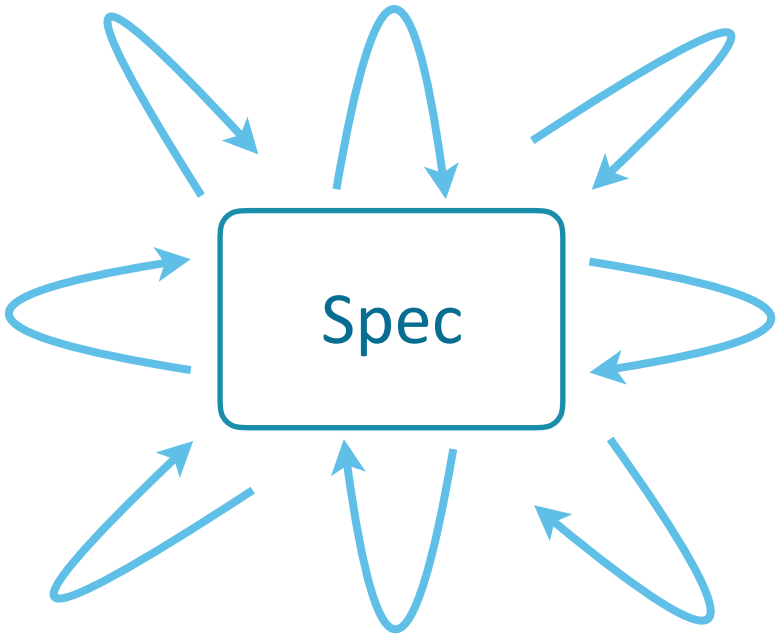
Test your specification

Build a virtuous cycle

- Look for early adopters
- What is “killer app” of your spec?

    Formally validation of implementations?
- Ensure specifications have many uses

  Don’t write spec in Coq/HOL/ACL2/…



© 2017 Arm Limited
arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

@alastair_d_reid

arm

“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

© 2017 Arm Limited

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)

Specification: https://developer.arm.com/products/architecture/a-profile/exploration-tools

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

(See also: https://github.com/herd/herdtools7/blob/master/herd/libdir/aarch64.cat)

Talk to me about how I can help you use it