Specifications: The Next Verification Bottleneck

Alastair Reid

Arm Research
@alastair_d_reid
Overview

1. What specifications do we need?
2. ARM’s formal processor specifications
3. Three steps I took to create good specifications
Designs processors, designs architecture, licenses architecture
16B processors / year
(also GPUs, IoT, ...)

Security Research Group
- Develop and analyse security extensions
- Create framework for verifying products
- We are hiring: full time, research internships
Applications
Libraries
Runtimes
Secure Services
MicroKernel
HAL
Architecture
MicroArchitecture
C Compiler / Linker
RTLD
## Specifications we need

**Linux sys calls**
- Processor page tables
- Interrupt handler
- Device driver API
- Filesystem format

**C stdlib**
- ISO C
- GCC/LLVM extensions
- Inline assembly
- ELF / linkerscript
- Weak memory model

**TCP/IP, UDP, ...**
- X11/Gtk+/

**TSL**
- Javascript, CSS, SVG, ...

**NTP, DNS, NFS, ...**
- PHP, ...

**WiFi, Bluetooth, Zigbee, ...**

**USB, SD card, ...**
Trusted Computing Base (≠ Trustworthy Computing Base)

a small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security

— Lampson

the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy

— Orange Book (US DoD)
Specifications for real world software/hardware

Unavoidable

Multiple implementations

Multiple versions of each implementation

Spec must include all quirks of recent versions of major implementations to be useful

Existing specification = English + Tables + Pseudocode

Existing community may not value formal spec at first
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
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 0xFFFFFFFF.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.
### Encoding A1
ARMv4*, ARMv5T*, ARMv6*, ARMv7

<table>
<thead>
<tr>
<th>ADC{S}&lt;c&gt;</th>
<th>&lt;Rd&gt;,&lt;Rn&gt;,&lt;Rm&gt;{,&lt;shift&gt;}</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
<td></td>
</tr>
<tr>
<td>cond</td>
<td>0</td>
</tr>
</tbody>
</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;
AArch64.DataAbort(bits(64) vaddress, FaultRecord fault)

route_to_el3 = HaveEL(EL3) && SCR_EL3.EA == '1' && IsExternalAbort(fault);
route_to_el2 = (HaveEL(EL2) && !IsSecure() && PSTATE.EL IN {EL0,EL1} &&
(HCR_EL2.TGE == '1' || IsSecondStage(fault)));

bits(64) preferred_exception_return = ThisInstrAddr();
vect_offset = 0x0;

exception = AArch64.Abort Syndrome(Exception_DataAbort, fault, vaddress);

if PSTATE.EL == EL3 || route_to_el3 then
  AArch64.TakeException(EL3, exception, preferred_exception_return, vect_offset);
elsif PSTATE.EL == EL2 || route_to_el2 then
  AArch64.TakeException(EL2, exception, preferred_exception_return, vect_offset);
else
  AArch64.TakeException(EL1, exception, preferred_exception_return, vect_offset);
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
## ARM Spec (lines of code)

<table>
<thead>
<tr>
<th>Category</th>
<th>v8-A</th>
<th>v8-M</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int/FP/SIMD</td>
<td>26,000</td>
<td>6,000</td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td>4,000</td>
<td>3,000</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>3,000</td>
<td>1,000</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>3,000</td>
<td>1,000</td>
</tr>
<tr>
<td><strong>Misc</strong></td>
<td>5,500</td>
<td>2,000</td>
</tr>
<tr>
<td>(Test support)</td>
<td>1,500</td>
<td>2,000</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>43,000</strong></td>
<td><strong>15,000</strong></td>
</tr>
<tr>
<td>Category</td>
<td>v8-A</td>
<td>v8-M</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Registers</td>
<td>586</td>
<td>186</td>
</tr>
<tr>
<td>Fields</td>
<td>3951</td>
<td>622</td>
</tr>
<tr>
<td>Constant</td>
<td>985</td>
<td>177</td>
</tr>
<tr>
<td>Reserved</td>
<td>940</td>
<td>208</td>
</tr>
<tr>
<td>Impl. Defined</td>
<td>70</td>
<td>10</td>
</tr>
<tr>
<td>Passive</td>
<td>1888</td>
<td>165</td>
</tr>
<tr>
<td>Active</td>
<td>68</td>
<td>62</td>
</tr>
<tr>
<td>Operations</td>
<td>112</td>
<td>10</td>
</tr>
</tbody>
</table>
Trustworthiness
Trustworthiness

ARM’s specification is correct by definition
Trustworthiness

ARM’s specification is correct by definition
Trustworthiness

Does the specification match the behaviour of all ARM processors?
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
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
Measuring architecture coverage of tests

Untested: \( \text{op1*op2} == -3.0, \text{FPCR.RND} = -\text{Inf} \)

```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 = FPOnePointFivc(0);
        elseif 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);
            end
            return result;
    end
```
Creating a Virtuous Cycle

ARM Spec

- Random Instruction Sequences
- Information Flow Analysis
- Specification Verification
- Processor Verification
- Boot OS
- Fuzzing Firmware
- Testcase Generation
- ARM Conformance TestSuite
“End to End Verification of ARM processors with ISA Formal,” CAV 2016

Formal validation of processors
Checking an instruction

ADD
Checking an instruction

Context

CMP  LDR  ADD  STR  BNE
Errors ISA-Formal can catch

- Errors in decode
- Errors in data path
- Errors in forwarding logic
- Errors in register renaming
- Errors in exception handling
- Errors in speculative execution
Specifying ADD

assign ADD_retiring = (pre.opcode & 16'b1111_1110_0000_0000) == 16'b0001_1000_0000_0000;
assign ADD_result = pre.R[pre.opcode[8:6]] + pre.R[pre.opcode[5:3]];
assign ADD_Rd = pre.opcode[2:0];

assert property (@(posedge clk) disable iff (~reset_n) ADD_retiring |-> (ADD_result == post.R[ADD_Rd]));
ISA Formal

- Finds complex bugs in processor pipelines
- Applied to wide range of μArchitectures
- Uses translation of ARM’s internal ISA specification
Challenges

- Complex Functional Units
  - FP
  - Memory
- Dual Issue
- Instruction Fusion
- Register Renaming
- Out-of-order Retire
# FP Subset Behaviour

<table>
<thead>
<tr>
<th>FPAdd</th>
<th>-∞</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>-1</td>
<td>-∞</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-∞</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>-∞</td>
<td>0</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
ISA Formal

- Finds complex bugs in processor pipelines
- Applied to wide range of μArchitectures
- Uses translation of ARM’s internal ISA specification
# ISA-Formal Properties

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[],D[],V[]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SysRegRW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## ISA-Formal Properties

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[],D[],V[]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SysRegRW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# ISA-Formal Properties

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>S[],D[],V[]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SysRegRW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# ISA-Formal Properties

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>…</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>...</td>
<td>✔️</td>
</tr>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>...</td>
<td>✔️</td>
</tr>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>...</td>
<td>✔️</td>
</tr>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>...</td>
<td>✔️</td>
</tr>
</tbody>
</table>

- ADC
- ADD
- B
- …
- YIELD

- R[]
- NZCV
- SP
- PC
- S[], D[], V[]
- FPSR
- MemRead
- MemWrite
- SysRegRW
- ELR
- ESR
- …
But this is slow and inconsistent
## ISA-Formal Properties

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

- **R[]**
- **NZCV**
- **SP**
- **PC**
- **S[],D[],V[]**
- **FPSR**
- **MemRead**
- **MemWrite**
- **SysRegRW**
- **ELR**
- **ESR**
- **...**
## ISA-Formal Properties

<table>
<thead>
<tr>
<th></th>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>NZCV</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SP</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>PC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>S[], D[], V[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SysRegRW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## ISA-Formal Properties

<table>
<thead>
<tr>
<th></th>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>NZCV</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SP</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>PC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>S[],D[],V[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FPSR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MemRead</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MemWrite</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SysRegRW</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ELR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ESR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## ISA-Formal Properties

<table>
<thead>
<tr>
<th></th>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>NZCV</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SP</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>PC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>S[],D[],V[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FPSR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MemRead</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MemWrite</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SysRegRW</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ELR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ESR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>...</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
## ISA-Formal Properties

<table>
<thead>
<tr>
<th></th>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>NZCV</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SP</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>PC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>S[],D[],V[]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FPSR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MemRead</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MemWrite</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SysRegRW</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ELR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ESR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Architecture Specification → ASL to Verilog → Combinational Verilog

- Specialize
- Monomorphize
- Constant Propagation
- Width Analysis
- Exception Handling
- ...

ARM Research
The Architecture for the Digital World®
Cumulative Defects %

Time after ISA-Formal starts (weeks)

<table>
<thead>
<tr>
<th>Area</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP/SIMD</td>
<td>25%</td>
</tr>
<tr>
<td>Memory</td>
<td>21%</td>
</tr>
<tr>
<td>Branch</td>
<td>21%</td>
</tr>
<tr>
<td>Integer</td>
<td>18%</td>
</tr>
<tr>
<td>Exception</td>
<td>8%</td>
</tr>
<tr>
<td>System</td>
<td>7%</td>
</tr>
</tbody>
</table>

Defect Detection by Area
## 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>

### Cambridge Projects

**Rolling out globally to other design centres**

Sophia, France - Cortex-A75 (partial)

Austin, USA - TBA

Chandler, USA - TBA
Formal validation of specifications
One Specification to rule them all?

Compliance Tests

Architecture Spec

Processors

Reference Simulator
Creating a redundant specification

Where to get a list of redundant properties from?
How to formalise this list?
How to formally validate specification against properties?

(This may look familiar from formal specification of software)
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 0xEFFFFFFE.
- EPSR.IT to become UNKNOWN.

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

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

Event Operators
Temporal Operators

- Fell(e) \[\text{Past}(e) > e\]
- Stable(e) \[\text{Past}(e) = e\]
- Rose(e) \[\text{Past}(e) < e\]
Temporal Operators

Fell(LockedUp)

__Past_LockedUp = LockedUp;

FunctionUnderTest();

... __Past_LockedUp > LockedUp ...

© 2017 Arm Limited
Event Operators

Called(TakeReset)

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

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

FunctionUnderTest();

assert((__Past_LockedUp > LockedUp)
    ==>
    ( __Called_TakeColdReset
      || __Called_TakeReset
      || __Past_Halted < Halted
      || __Called_ExceptionEntry));
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.

\[\text{Fell(LockedUp)} \rightarrow \text{Called(TakeColdReset)} \lor \text{Called(TakeReset)} \lor \text{Rose(Halted)} \lor \text{Called(ExceptionEntry)}\]

```c
__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 ➡️ SMT

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

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

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

Talk to me about how I can help you use it
Specifications: The next bottleneck

We will need a lot of specs
Of real world s/w + h/w
Specs are a large part of TCB
How are we going to create them?
How are we going to trust them?

Test the specifications you depend on
Formally validate/verify implementations
Create redundant specifications
Ensure specifications have many uses
Don’t write spec in Coq/HOL/ACL2/…
Try to influence official specification
Thanks

Alasdair Armstrong (Cambridge U.)
Alex Chadwick (ARM)
Ali Zaidi (ARM)
Anastasios Deligiannis (ARM)
Anthony Fox (Cambridge U.)
Ashan Pathirane (ARM)
Belaji Venu (ARM)
Bradley Smith (ARM)
Brian Foley (ARM)
Curtis Dunham (ARM)
David Gilday (ARM)
David Hoyes (ARM)
David Seal (ARM)
Daniel Bailey (ARM)
Erin Shepherd (ARM)
Francois Botman (ARM)
George Hawes (ARM)
Graeme Barnes (ARM)
Isobel Hooper (ARM)
Jack Andrews (ARM)
Jacob Eapen (ARM)
Jon French (Cambridge U.)
Kathy Gray (Cambridge U.)
Krassy Gochev (ARM)
Lewis Russell (ARM)
Matthew Leach (ARM)
Meenu Gupta (ARM)
Michele Riga (ARM)
Milosch Meriac (ARM)
Nigel Stephens (ARM)
Niyas Sait (ARM)
Peng Wang (ARM)
Peter Sewell (Cambridge U.)
Peter Vrabel (ARM)
Richard Grisenthwaite (ARM)
Rick Chen (ARM)
Simon Bellew (ARM)
Thomas Grocutt (ARM)
Will Deacon (ARM)
Will Keen (ARM)
Wojciech Meyer (ARM)
(and others)
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