Engineering Formal Specifications of the Arm Processor Architecture

Alastair Reid
Arm Research
@alastair_d_reid
Engineering (Wikipedia)

**Engineering** is the creative application of science, mathematical methods, and empirical evidence to the innovation, design, construction, operation and maintenance of structures, machines, materials, devices, systems, processes, and organizations for the benefit of humankind.
The IoT security problem
The IoT security problem
The IoT security problem

- Apps
- OS + Network
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical
- Rowhammer
- CLKScrew
- DPA

Fog

DataCenter
The IoT security problem

- Apps
- OS + Network
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical

- DataCenter

- Timing Side Channels
- Spectre
- Meltdown
- CLKScrew
- Rowhammer
- DPA

DPA
Rowhammer
CLKScrew
Meltdown
Spectre
Timing Side Channels

Fog
The IoT security problem

- Apps
- OS + Network
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical

- SGX Bomb
- Timing Side Channels
- Rowhammer
- DPA
- MGX
- Spectre
- Meltdown
- CLKScrew
- DataCenter
- Fog

DPA
Rowhammer
CLKScrew
The IoT security problem

Apps
OS + Network
TEE
Firmware
Architecture
MicroArchitecture
Physical

SGX Bomb
Timing Side Channels
Rowhammer

DPA

Nintendo Switch
MGX
Spectre
Meltdown
CLKScrew

Fog

DataCenter

DPA

Rowhammer

SGX Bomb

Timing Side Channels

Nintendo Switch

MGX

Spectre

Meltdown

CLKScrew

DPA
The IoT security problem
The IoT security problem

- MicroArchitecture
- Architecture
- Physical
- Firmware
- TEE
- OS + Network
- Apps
- Policies

- DPA
- CLKScrew
- Meltdown
- Spectre
- SGX Bomb
- MGX
- Trusting user data
- Nintendo Switch
- Qualcomm TZ
- Heartbleed
- Timing Side Channels
- Rowhammer
- DPA

DataCenter

Fog

3
The IoT security problem

- Physical
  - Firmware
  - Architecture
  - MicroArchitecture
  - DPA
  - Rowhammer
  - Timing Side Channels
  - CLKScrew
  - SGX Bomb
- TEE
- OS + Network
  - Apps
    - Trusting user data
  - Qualcomm TZ
- Policies
  - OS + Network
  - TEE
  - Architecture
  - MicroArchitecture
  - Physical
  - MGX
  - Spectre
  - Meltdown
- DataCenter
The IoT security problem

- Apps
- Trusting user data
- OS + Network
- TEE
- Firmware
- Nintendo Switch
- Architecture
- MGX
- MicroArchitecture
- Spectre
- Physical
- Meltdown
- CLKScrew
- DPA

Apps
- Kubernetes
- OS
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical
The IoT security problem

- Apps
- OS + Network
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical

- Trusting user data
- Heartbleed
- Qualcomm TZ
- SGX Bomb
- Timing Side Channels
- Rowhammer

- Nintendo Switch
- MGX
- Spectre
- Meltdown
- CLKScrew
- DPA

- Policies
- OS + Network
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical

- Apps
- Kubernetes
- OS
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical
The IoT security problem

- Apps
  - Trusting user data
- Operating System + Network
- TEE
- Firmware
  - Nintendo Switch
- Architecture
- MicroArchitecture
  - MGX
  - Spectre
  - Meltdown
- Physical
  - CLKScrew
  - DPA
- Rowhammer

- Policies
- OS + Network
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physics

- Apps
  - Kubernetes
- OS
- TEE
- Firmware
- Architecture
- MicroArchitecture
- Physical
The IoT security problem

MicroArchitecture

Architecture

Firmware

OS + Network

Apps

Physical

Firmware

TEE

SGX Bomb

Nintendo Switch

Rowhammer

CLKScrew

DPA

BGP Poisoning

SSL Certificate Attack

DNS Spoofing

Kubernetes

Apps

OS

TEE

Architecture

MicroArchitecture

Physical

Policies

OS + Network

TEE

Architecture

MicroArchitecture

Physic

Trustin user data

Heartbleed

Qualcomm TZ

Spectre

Meltdown

BGP Poisoning

SSL Certificate Attack

DNS Spoofing

Nintendo Switch

SGX Bomb

MGX

ClK Screw

Meltdown

Spectre

Timing Side Channels

Rowhammer

DPA

Heartbleed

Qualcomm TZ

SGX Bomb

MGX

ClK Screw

Meltdown

Spectre

Timing Side Channels

Rowhammer

DPA
Reasoning about software and hardware

- Programming
- Reverse Engineering
- Simulation
- Glitching
- Exploit Detection
- Bug Finding
- Automatic Test Generation
- Formal Verification
- Fuzz Testing
- DPA
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
Overview

1. Reasoning about programs
2. ARM’s formal processor specifications
   - Three experiences
   - Lessons learned
   - (Work in progress)
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/
Creating trustworthy specifications
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, HVIC, ISB, NOP, SMC, or SVC instruction.

For the B, BL, BRK, HVIC, 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.
Semi-structured English prose (M-class spec)

**RJRJC**

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.

**RVGNW**

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 \(0xFFFE\).
- EPSR.IT to be become unknown.

In addition, HFSR.FORCED is not set to 1.
Tables - semistructured, not machine readable

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

<table>
<thead>
<tr>
<th>Accesses</th>
<th>Shareability domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full system</td>
</tr>
<tr>
<td><strong>Before the barrier</strong></td>
<td><strong>After the barrier</strong></td>
</tr>
<tr>
<td>Reads and writes</td>
<td>Reads and writes</td>
</tr>
<tr>
<td>Writes</td>
<td>Writes</td>
</tr>
</tbody>
</table>
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>}

<table>
<thead>
<tr>
<th>cond</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;
```
Pseudocode

```plaintext
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>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;
  
  else
    // More code...
```

- **Type Inference**
- **Unbounded Integers**
- **Enumerations**
- **Bit Vectors**
- **Indentation-based Syntax**
- **Dependent Types**
- **Imperative**
- **Exceptions**
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
- Senior architects believed that an executable spec was
  - Impossible
  - Not useful
  - Less readable
  - Less correct
Architectural Conformance Suite

Processor architectural compliance sign-off

Large

• v8-A 32,000 test programs, billions of 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: op1*op2 == -3.0, FPCR.RND=-Inf

```c
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 FOR 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 FPCRounding() == FPRounding_NEGINF then '1' else '0';
                result = FPZero(sign, N);
            else
                result = FPRound(result_value, FPCRounding());
            return result;
```
Creating a Virtuous Cycle

ARM Spec
Lessons learned about engineering a specification

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
Formal validation of processors and of specifications
Checking an instruction

ADD
Checking an instruction

Context

CMP  LDR  ADD  STR  BNE
Formal framework (deterministic specs)

Stimulus -> Implementation -> Specification

? == ?

Bounded model checker
# 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

Cambridge Projects
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
Formal validation of specifications
One Specification to rule them all?

- Compliance Tests
- Architecture Spec
- Processors
- Reference Simulator
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

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

\[
\text{Fell(LockedUp)} \rightarrow \text{Called(TakeColdReset)}
\]
\[
\lor \text{Called(TakeReset)}
\]
\[
\lor \text{Rose(Halted)}
\]
\[
\lor \text{Called(ExceptionEntry)}
\]
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 0xEFFFFFFFE.

- EPSR.IT to become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.
~10,000 lines

v8-M Spec + Rules

Convert

Counterexample

~1,000,000 lines

Z3 SMT Solver
Lessons Learned from 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

Needed to translate specification to another language to let us use other tools
Work in progress:
Security of architecture specifications
Validating security of processor architectures

Scope

- Compositional Attacks
- Hardware-based Security Enforcement
- Confidentiality, Integrity, Availability (and more?)

Challenges

- Cyclic dependencies between HSEs
- Microarchitectural storage/timing channels
ARM Processor

ARM Specification

Translate to Verilog

Verilog Model Checker

v8-M Spec + Properties

Translate to SMT

Z3 SMT Solver
Engineering Formal Specifications of Real World Artifacts

Plan for adoption into official specs

Apply standard engineering practices
- Test, review, CI, ...
- Understand approximations and limitations

Build a virtuous cycle
- Look for early adopters
- What is “killer app” of your spec?
- Ensure specifications have many uses
  (Don’t write spec in Coq / HOL / ACL2 / ...)
Public release of machine readable Arm specification

Enable formal verification of software and tools


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
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

@alastair_d_reid

arm
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Mixed Mode Verification
Mixed Mode Verification

Idle

ADD
BNE
CMP
LDR
STR
ADD
Mixed Mode Verification
Mixed Mode Verification
Mixed Mode Verification

Idle

ADD

Idle

CMP

BNE

LDR

STR
Mixed Mode Verification

- Idle
- ADD
- CMP
- BNE
- LDR
- STR

alastair.reid@arm.com @alastair_d_reid
Mixed Mode Verification

- Idle
- ADD
- CMP
- BNE
- LDR
- STR