Tortuga Logic

RISC-V Security Verification Using Tortuga Logic's Radix-S

Nicole Fern

Senior Hardware Security Engineer

April 10th, 2019

Outline

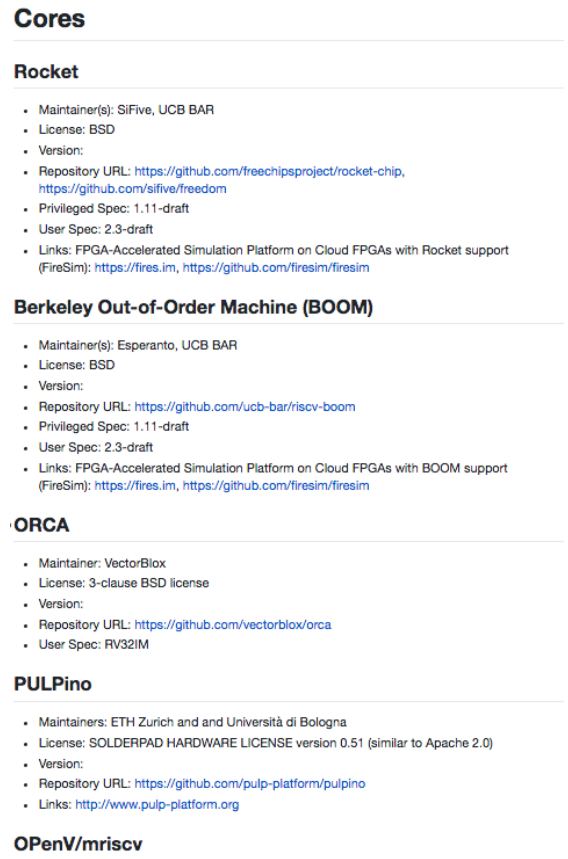
- Security and the RISC-V Ecosystem

- Overview of Security Verification Using Radix-S
  - Value Radix-S Provides
  - Information Flow Security Properties
  - Radix-S Flow

- Example: RISC-V Rocket
  1. Control and Status Registers (CSRs)
  2. Physical Memory Protections (PMPs)
  3. Cache Side-channels

RISC-V Architecture Vs. Microarchitecture

- Due to open source nature, design and microarchitecture **customized heavily per application**
- Realizations of an ISA are *implementations* which differ based on *microarchitecture*
- The microarchitecture is how the ISA is actually built:
  - Caches
  - Branch predictors
  - Speculative execution / prefetching
  - Out-of-order execution
- Microarchitectural details hidden from SW developer's point of view but greatly help performance



Cores

Rocket

- Maintainer(s): SiFive, UCB BAR
- License: BSD
- Version:
- Repository URL: https://github.com/freechipsproject/rocket-chip, https://github.com/sifive/freedom
- Privileged Spec: 1.11-draft
- User Spec: 2.3-draft
- Links: FPGA-Accelerated Simulation Platform on Cloud FPGAs with Rocket support (FireSim): https://fires.im, https://github.com/firesim/firesim

Berkeley Out-of-Order Machine (BOOM)

- Maintainer(s): Esperanto, UCB BAR
- License: BSD
- Version:
- Repository URL: https://github.com/ucb-bar/riscv-boom
- Privileged Spec: 1.11-draft
- User Spec: 2.3-draft
- Links: FPGA-Accelerated Simulation Platform on Cloud FPGAs with BOOM support (FireSim): https://fires.im, https://github.com/firesim/firesim

ORCA

- Maintainer: VectorBlox
- License: 3-clause BSD license
- Version:
- Repository URL: https://github.com/vectorblox/orca
- User Spec: RV32IM

PULPino

- Maintainers: ETH Zurich and and Università di Bologna
- License: SOLDERPAD HARDWARE LICENSE version 0.51 (similar to Apache 2.0)
- Version:
- Repository URL: https://github.com/pulp-platform/pulpino
- Links: http://www.pulp-platform.org

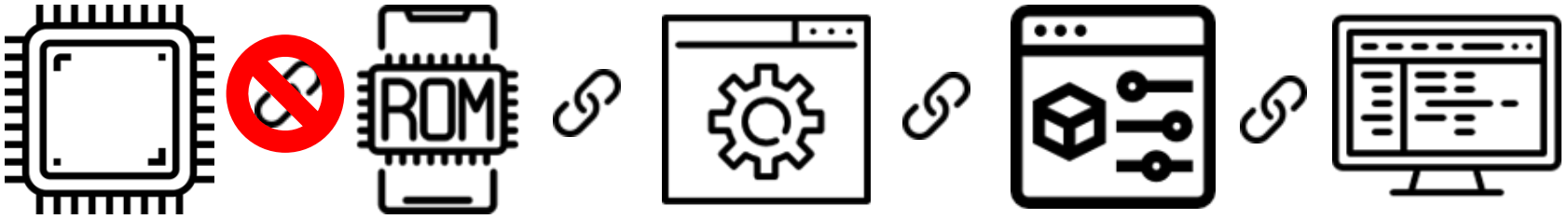
OPenV/mriscv

https://github.com/riscv/riscv-wiki/wiki/RISC-V-Cores-and-SoCs

Same ISA – Different implementations

RISC-V Security

- Even if the ISA is sound, the *implementation* can be vulnerable
  - This is true even if microarchitecture is formally proven to conform to the ISA
- Even if HW implementation is secure, usage/configuration of processor features can be incorrect at the system level when firmware/SW is added
  - Ex. Software code incorrectly programs PMP registers
- Security is a trust hand-off, any broken links can result in system-level exploit



Security Verification of RISC-V Based System

- *Processor-level*: Verify specific RISC-V implementation does not violate core-level security objectives
  - Information leakage through cache or other microarchitectural state
  - Implementation of privilege levels, access control to CSRs, etc.

- *System-level*: RISC-V Core integrated into larger platform should not violate core-level or platform-level security objectives
  - If RISC-V implementation itself is highly configurable, verify exact HW configuration instantiated in system
  - Software usage/configuration of RISC-V features

Strategy must include both processor and system-level verification

Overview of Radix-S

Problem: Security Verification Difficult

- Security objectives naturally described in terms of *information flow*
  - Does configuration information or sensitive data flow from higher to lower privilege levels?
  - Can a lower privilege level corrupt privileged registers or data regions?
  - Is it possible to learn memory access patterns of other processes through cache timing side channels (ex. Meltdown and Spectre)?

- Standard pre-silicon verification techniques do not provide a mechanism for verifying security properties across hardware and software

Solution: Tortuga Logic Radix-S

- Provides mechanism to specify hardware information flow properties and verify during RTL simulation
- Integrates easily with existing pre-silicon verification environments
- Scalable to system level HW + SW verification
- Improves security coverage with low overhead
- Discover unknown vulnerabilities using existing test stimulus
  - Information flow properties + our tracking technology function as a “super checker”
  - Information flow checked, not signal values

Radix-S: Use Cases

- Hardware Roots of Trust (HRoTs)
- SoC Access control verification
- Secure boot analysis
- Timing side channels
- Encryption key leakage
- Configuration register read/write protection
- Memory Protection Unit (MPU) configuration
- JTAG disablement/analysis
- 3rd party/vendor IPs and interfaces

Radix-S Security Properties

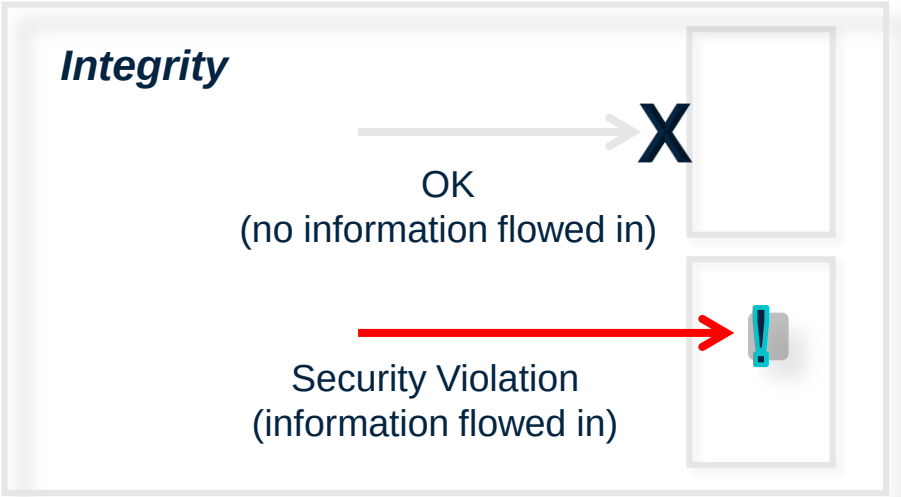
Information Flow Security Properties

Ensuring Integrity

Integrity is violated when an unauthorized logical (SW) or physical (HW) asset can modify a target register/memory state

`untrusted_signals =/=> trusted_signals`

"source" *"destination"*

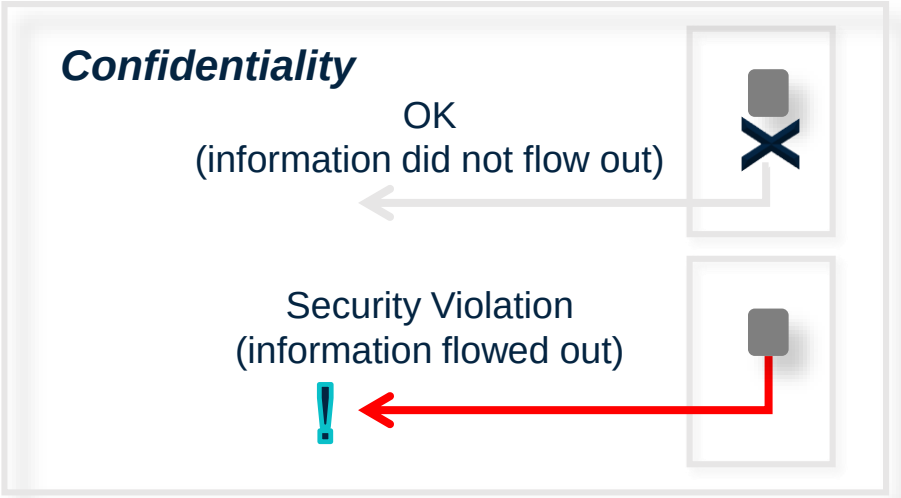


Ensuring Confidentiality

Confidentiality is violated when an unauthorized logical (SW) or physical (HW) asset can read the state of a target register/memory

`secret_signals =/=> output_signals`

"source" *"destination"*



Temporal Rules

A when X =/=> B

- Starts tracking A when condition X is true and rule will **fail** if A then flows to B.

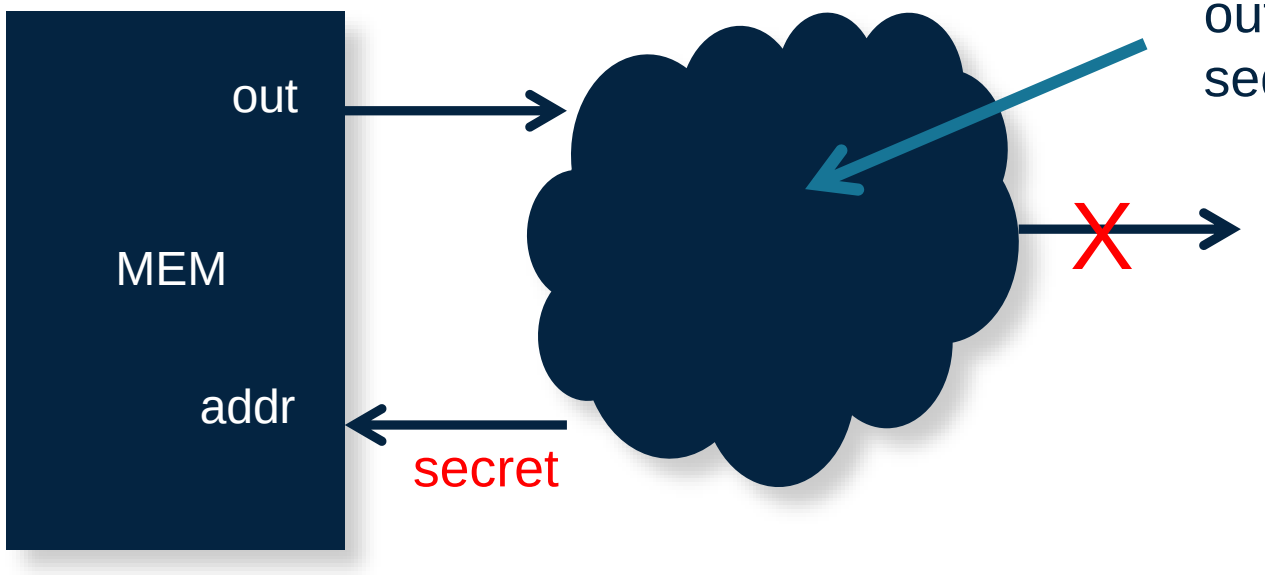
(A =/=> B) || X

- Starts tracking A at t=0. Rule will **pass** if A doesn't flow to B or X is true.

Memory Read Protection

mem.out when (addr == secret) =/=> $all_outputs

- Secret content being read out of memory should not reach any output

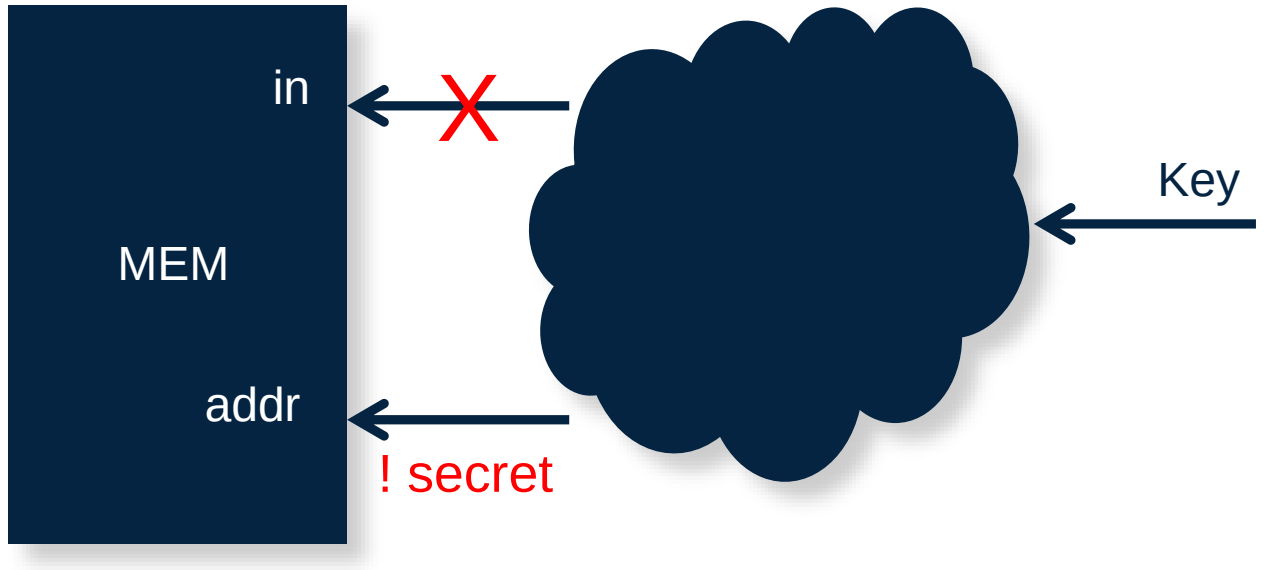


Radix-S tracks flow of memory output through arbitrary logic and sequential transformations

Memory Write Protection

(key =/=> mem.in) || addr == secret

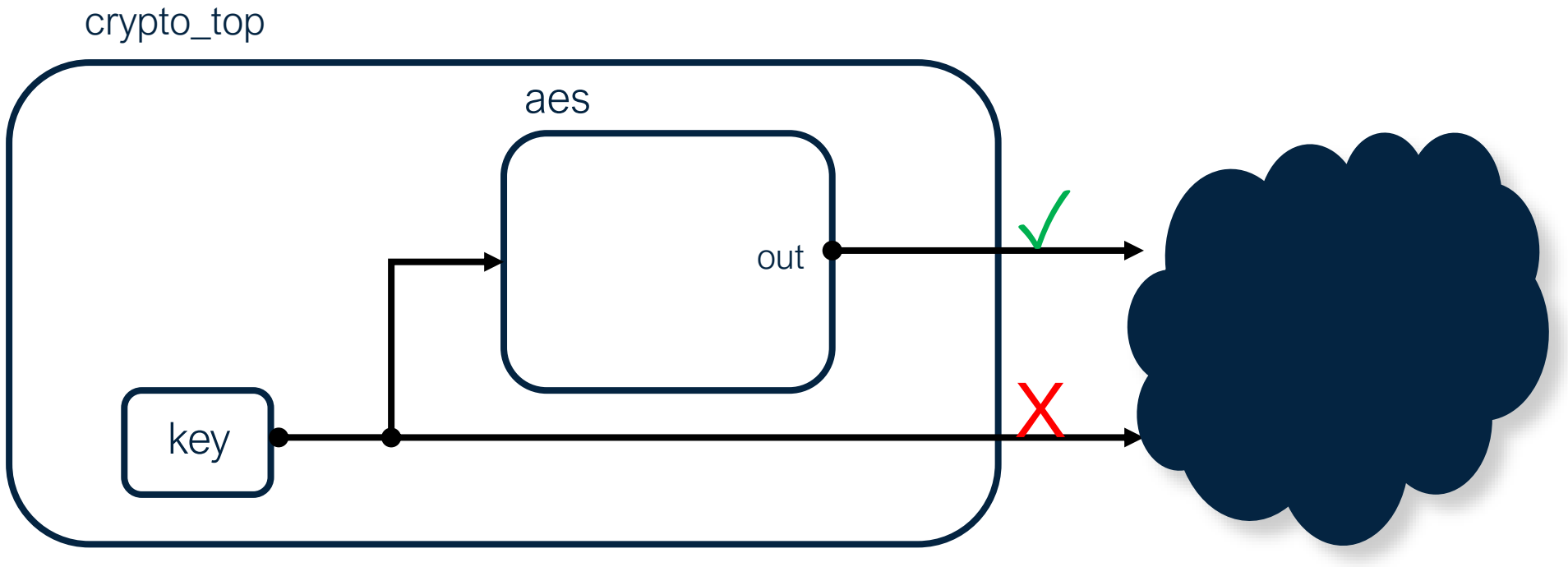
- Secret content (i.e. key) should not be written to non-secure memory region. Flow to mem.in allowed if addr == secret.



Ignoring Condition

key =/=> $all_outputs ignoring aes.out

- Specifies that aes.out is “secure.” Ignores information flow.



Radix-S Verification Flow

How to use Radix-S

Step 1 - Define the Threat Model

- Specify assets to protect and intended use of those assets
- Specified in Tortuga Logic's rule set

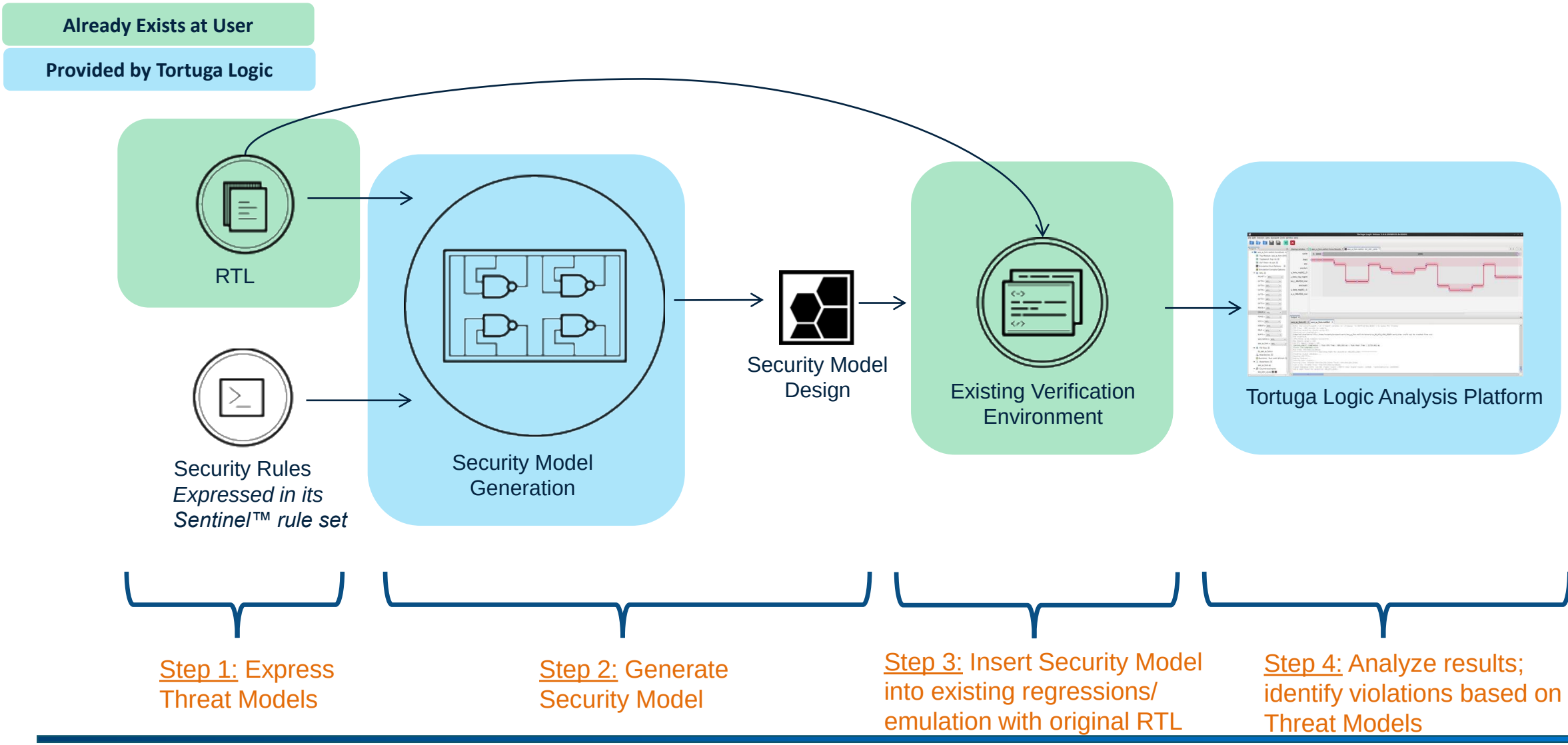
Step 2 - Hardware design (RTL) and Threat Model are analyzed to produce our Security Model Design (SMD), a synthesizable hardware IP.

Step 3 – Run SMD in parallel with RTL inside existing pre-silicon functional verification environments (Formal Verification, Simulation, or Emulation)

- Tortuga Logic's Radix-S product is used in standard RTL simulation environments

Step 4 – Analyze results and identify security violations

Radix-S User Flow



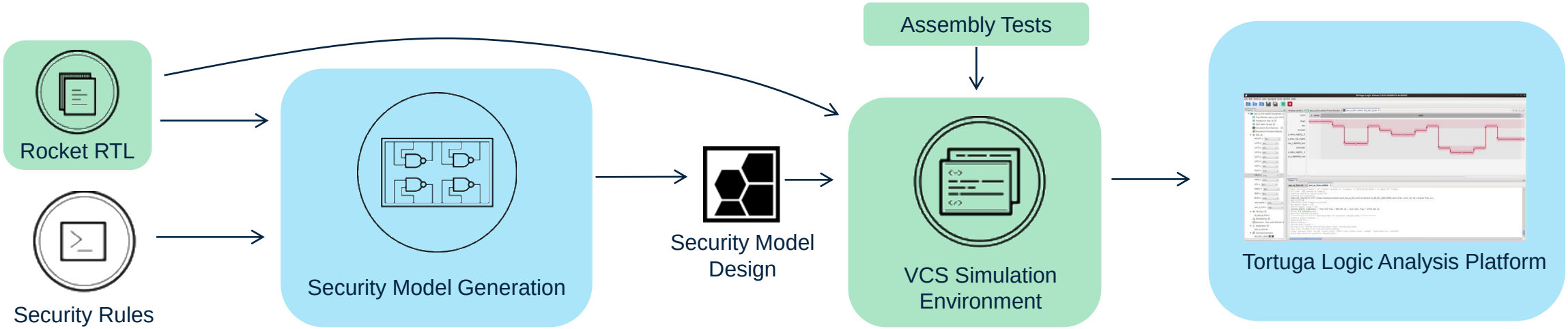
Security Verification of RISC-V Rocket Chip using Radix-S

RISC-V Rocket

- Verilog RTL for Rocket System generated from Chisel using default configuration
- Security Verification performed for this specific configuration
- Security rules generalize to other configurations
- Security Objectives:
  1. Confidentiality and integrity of control and status registers (CSRs)
  2. Access control and functionality of Physical Memory Protections (PMPs)
     - Platform specific as well as general properties
  3. Detect cache timing side channels

Integration into Rocket Simulation Flow

- Design Scope for Simulation: Rocket System
- Security Model Scope
  - Rocket Core: CSR and PMP verification
  - Rocket DCache: Data cache side-channel detection
- Tests: 122 rv64*-p-* assembly tests



1. Confidentiality and integrity of control and status registers (CSRs)

Rocket CSRs Analyzed

- Read-only CSRs
  - Hardware Thread ID (HART_ID)
- Writable only in privileged modes
  - Memory access privilege bits in mstatus (MPRV, MXR, and SUM bits)
- Readable only in privileged modes
  - Machine-mode Interrupt Enable Register (MIE)

Security rules for CSRs can be re-used for different RISC-V implementations and configurations

Integrity Property for Memory Access Privilege Bits

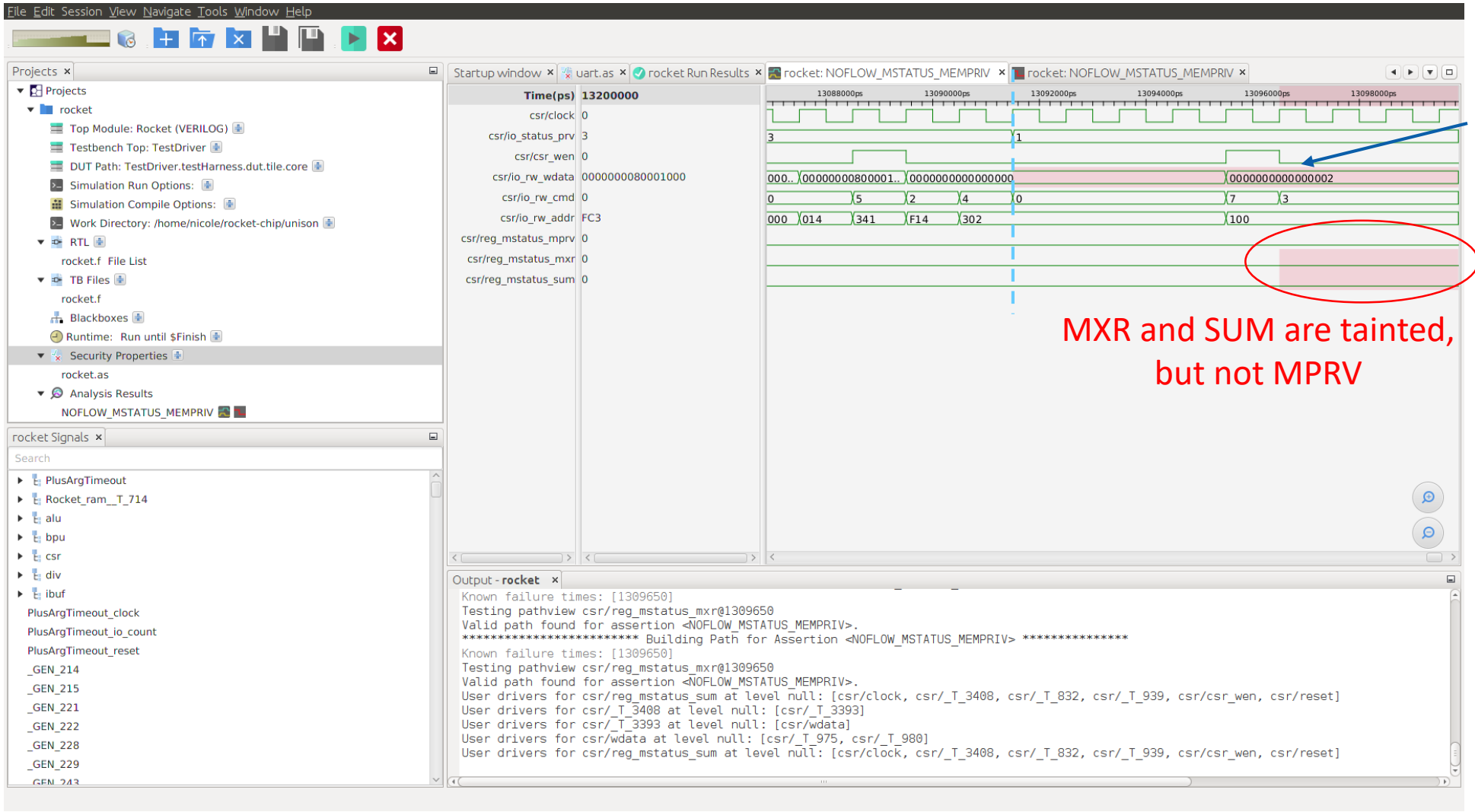
MPRV, MXR, and SUM bits can only be written in machine mode:

csr.io_rw_wdata when (*csr.io_status_prv* != MACHINE_MODE)

=/=> {csr.reg_mstatus_mprv, csr.reg_mstatus_mxr, csr.reg_mstatus_sum}

- Property fails for 4 out of 122 tests
  - Failing Tests: rv64mi-p-illegal, rv64si-p-csr, rv64si-p-scall, rv64si-p-wfi
  - Failing tests exercise the **supervisor**-level status register (sstatus)
- Re-examined specification and realized MXR and SUM bits can be set in supervisor mode by writing to sstatus register
  - Section 4.1.3 of *Volume II: RISC-V Privileged Architectures V1.10*

Analyzing Failing Test in Radix-S



Write sstatus (CSR Address 0x100)

MXR and SUM are tainted, but not MPRV

Sentinel Properties for MIE Register

Integrity: Write data to CSR module controlled by lower privilege levels than machine mode should never flow to MIE:

csr.io_rw_wdata when (*csr.io_status_prv* < MACHINE_MODE)

=/=> {*csr.reg_mie*[11], *csr.reg_mie*[7], *csr.reg_mie*[3]}

Confidentiality: MIE should never reach read data CSR output unless in machine mode:

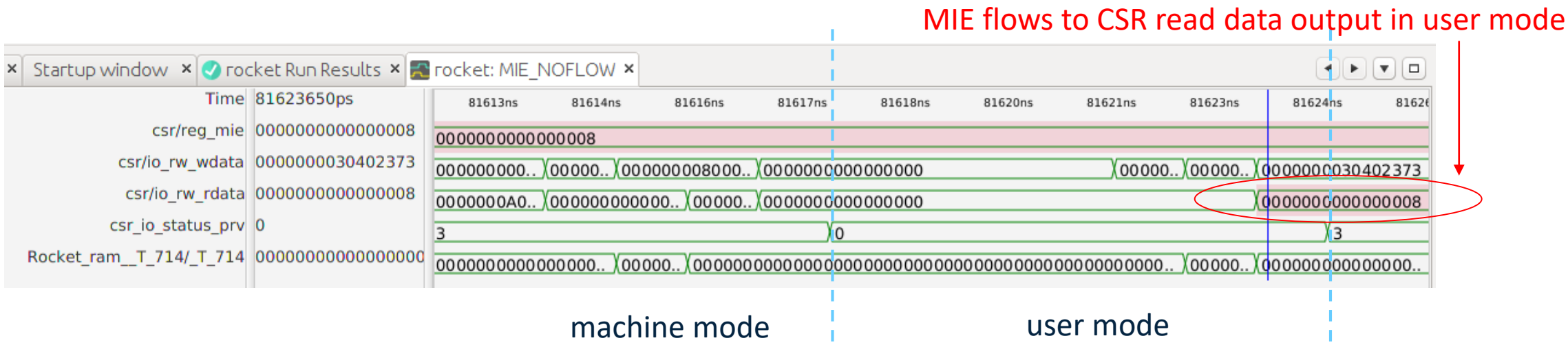
{*csr.reg_mie*[11], *csr.reg_mie*[7], *csr.reg_mie*[3]}

=/=> *csr.io_rw_rdata* || *csr.io_status_prv* == MACHINE_MODE

* Bits 11, 7, and 3 of reg_mie signal correspond to machine mode interrupts

MIE Verification Results

- Integrity property PASSES for all 122 rv64-p assembly tests
- Confidentiality property FAILED
  - Created a custom testcase to read MIE in user mode and verified that MIE reaches the read data CSR output before causing exception
  - Changed property to check flows to register file and it PASSES



2. Access control and functionality of Physical Memory Protections (PMPs)

RISC-V Physical Memory Protections

- Enables privileged software to specify access control policies for physical memory
- Used to create trusted execution environments
  - Keystone Project, Hex Five MultiZone, custom solutions, etc.
  - All can verified with Radix-S

Two main aspects to verify:

1. *Hardware* implementation of PMP functionality is secure
2. *Software* PMP configuration does not violate security objectives (security properties are platform specific)

1. *Hardware*-focused Security Properties

- Integrity: Who can modify the PMP registers?
  - S and U modes should never be able to modify PMPs
  - When PMP entry is locked it can't be modified until reset
- R/W/X access control implemented correctly for regions as specified by PMP registers
- Generalize to different RISC-V implementations
- Re-usable during many stages of verification
  - RISC-V processor level testing
  - After integration of RISC-V core into larger system
  - Running actual software used to configure PMPs (ex. TCB code)

Properties pass for rv64-p assembly tests

2. *Software*-focused Security Properties

- Hardware-centric properties assume PMP registers programmed correctly for desired security objectives
- Software configuration security properties are created with platform-specific knowledge about the memory map
  - Check that PMPs programmed correctly
- Example: PMPs used to designate region of memory from address 0x0000 to 0x8000 as secure

`mem.data_out when (mem.addr < 0x8000) =/=> {IR, regfile} || priv == M_MODE`

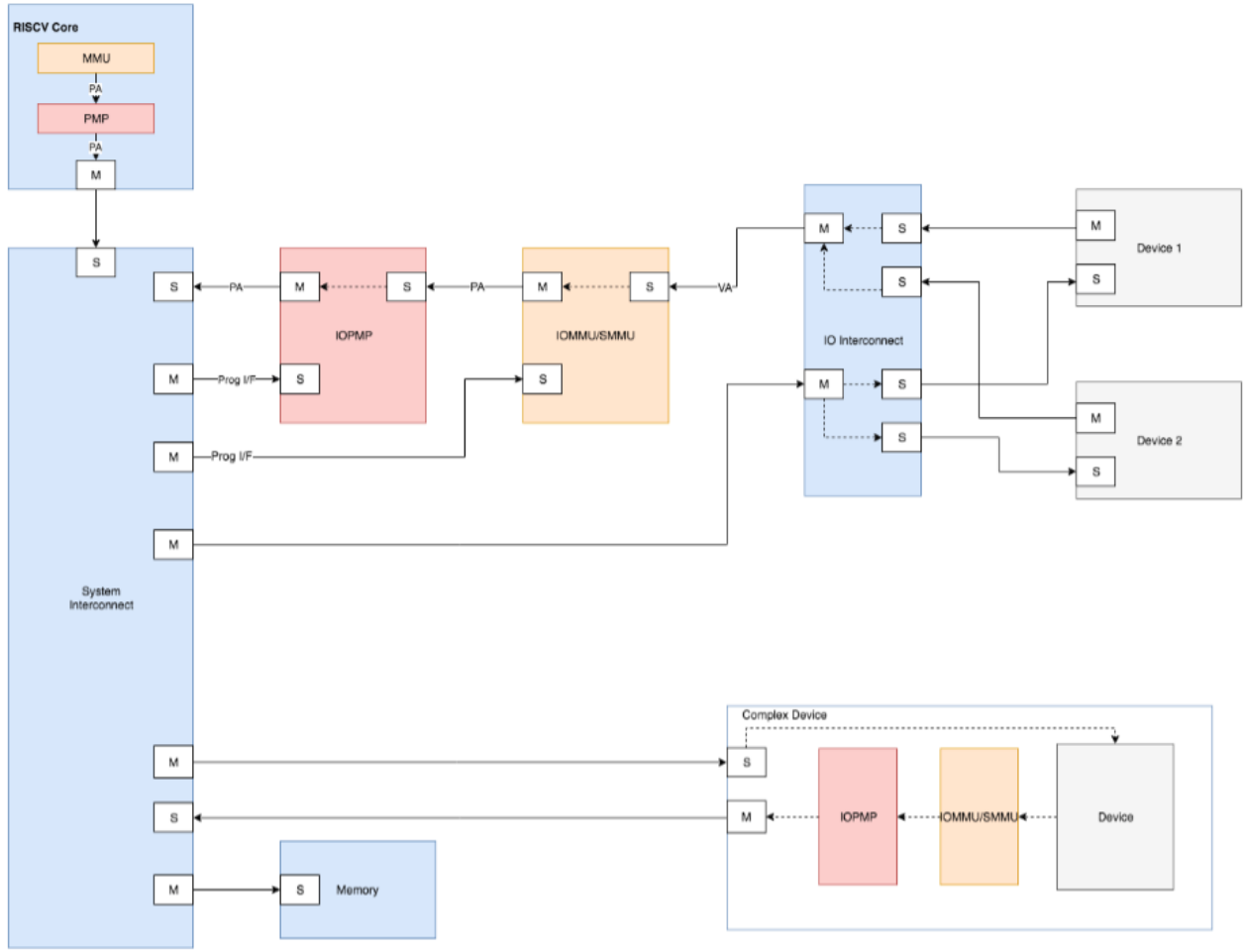
No read or execute

`regfile when (priv != M_MODE) =/=> mem.data_in || (mem.addr > 0x80000)`

No write

System-Level Verification of Memory/Peripheral Access

- Given a high level security objective verify:
  - Hardware IOPMP and interconnect configuration
  - Software programming of IOPMP regions
- Example: Complex Device should never be able to access Device 2
  - Verify no information flows between these 2 blocks



System from IOPMP IOMMU Proposal

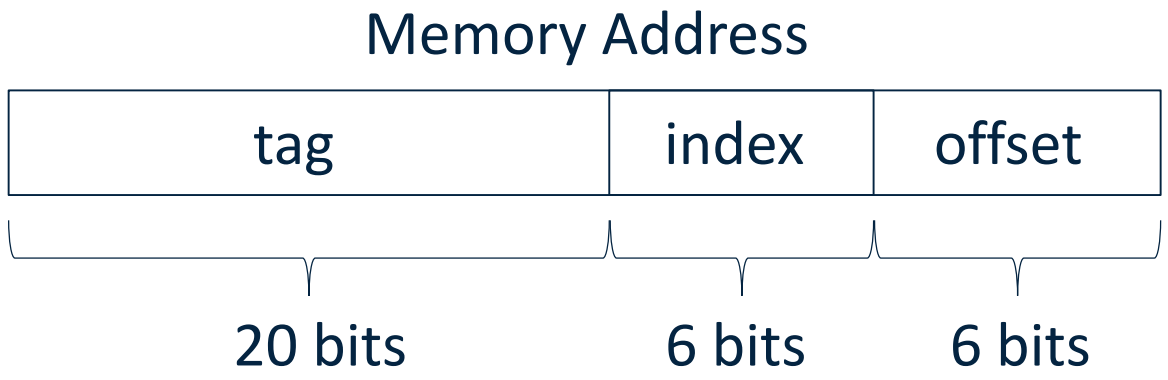

3. Detect cache timing side channels

Rocket Data Cache Timing Side Channel

- Cache is a shared resource amongst privilege levels
- No cache flush instruction
  - Machine mode must manually evict all sensitive data from the cache before entering supervisor or user mode
- No speculative loads/stores in Rocket, but data access patterns in machine mode still leak to lower privilege levels
  - Relevant for applications where data itself is public (ex. AES look-up table) but access patterns reveal secret information
  - Access patterns observable via cache timing side channel

Rocket Data Cache Parameters (DefaultConfig)

- Number of ways: 4
- Number of sets: 64
- Cache line size: 64 bytes



Example

- Address 0x80003000
  - tag = 0x80003
  - index = 0, offset = 0
- Address 0x80004000
  - tag = 0x80004
  - index = 0, offset = 0
- **Map to same cache set**

Detecting Timing-based Side Channels

- Tortuga Logic technology capable of detecting *indirect* information flows, including timing-based side channels
- Cache timing side channel is an unwanted flow of memory addresses accessed to the cache “hit or miss” control signal
- Security property tracks flow of metadata related to the memory address (ex. cache entry tag) across privilege levels

Rule for Detecting Cache Side-channel in Rocket

Cache metadata (the tag) written to the cache tag array while in machine mode should never flow out of the data cache unless in machine mode

```
assert iflow ( dcache.tag_array.wdata
                    when (dcache.dprv == MACHINE_MODE)
                     =/=>  dcache.$all_outputs
                    || dcache.dprv == MACHINE_MODE );
```

Cache Side-channel Verification Results

- Detecting cache side-channels using Radix-S does *not* require custom tests which actively exploit the side-channel
- Property PASSES for 12 of the rv64-p tests, FAILS for **110** tests
- Any test which does not manually flush the cache before switching to user mode and accessing data memory will cause a security property violation

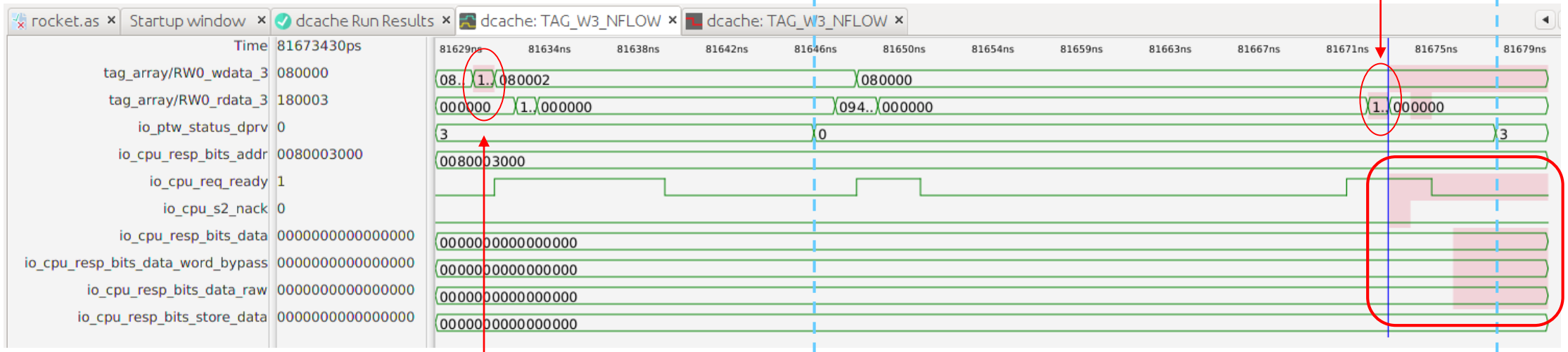
Radix-S Waveform Illustrating Cache Tag Leakage

Security Rule: Address information written cache tag array while in m-mode should never flow out of the data cache unless in m-mode

Tag 0x80003 read out during comparison leading to cache hit

Machine Mode

User Mode



Tag 0x80003 written to cache when machine mode accesses 0x8003000

Data cache outputs tainted after user accesses 0x80003000

Summary

- Radix-S can be used to develop a common library of information flow properties for the RISC-V privileged spec. and extensions
  - CSRs, PMPs, timing channels (cache, encryption cores, etc.)
  - Properties verified for specific RTL implementations
- System-level verification of systems incorporating RISC-V cores possible with Radix-S
  - Verify processor-level properties in full system context
  - Verify system-level security properties
    - Secure debug
    - TEEs HW mechanism + SW (system monitor code)
    - Memory and peripheral access control policies

Tortuga Logic

Thanks!