RISCV DEBUG SECURITY

Joe Xie, Aote Jin @nVidia

Agenda

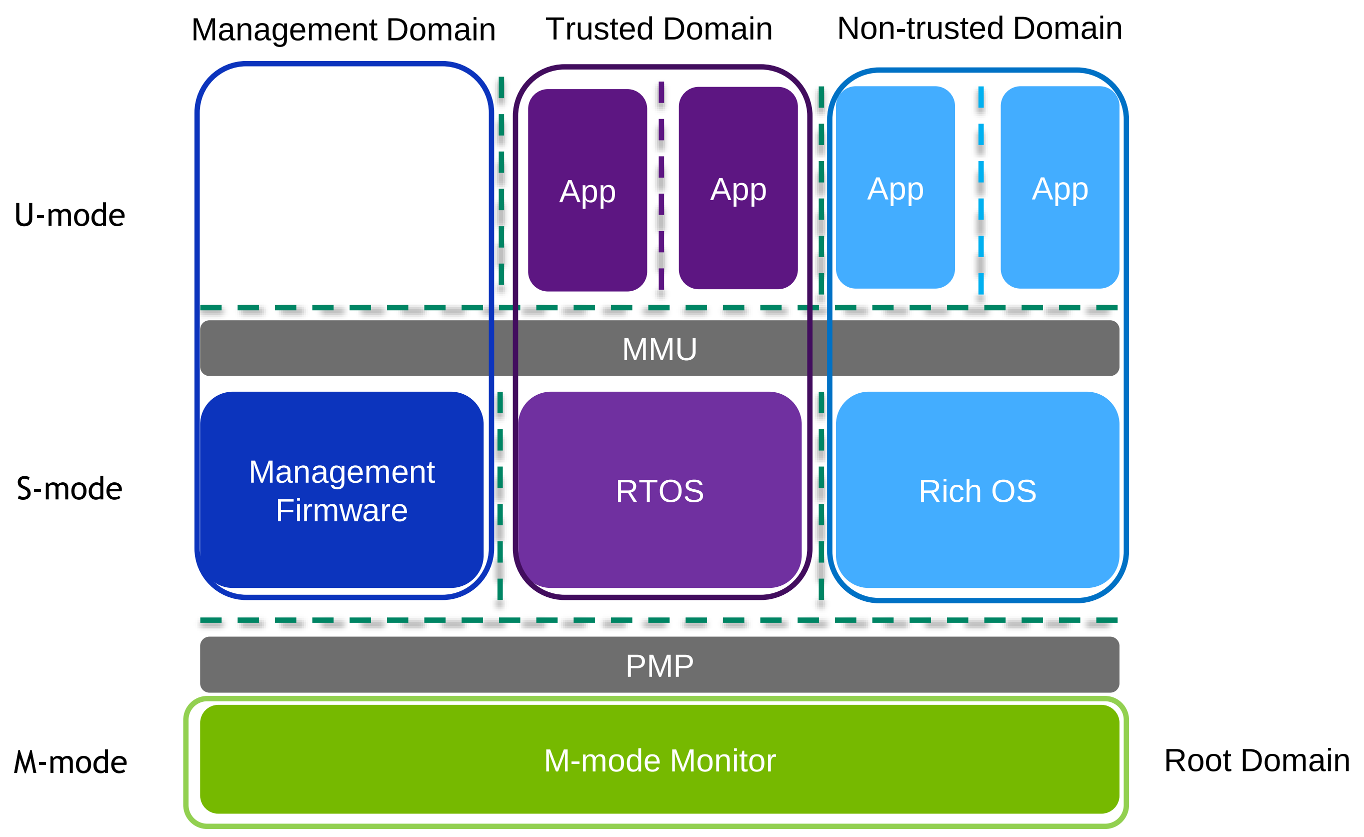
The issue

The (draft) proposal

The issue

RISC-V SECURITY MODEL

Multi-Domain Security Model



RISC-V EXTERNAL DEBUG OVERVIEW

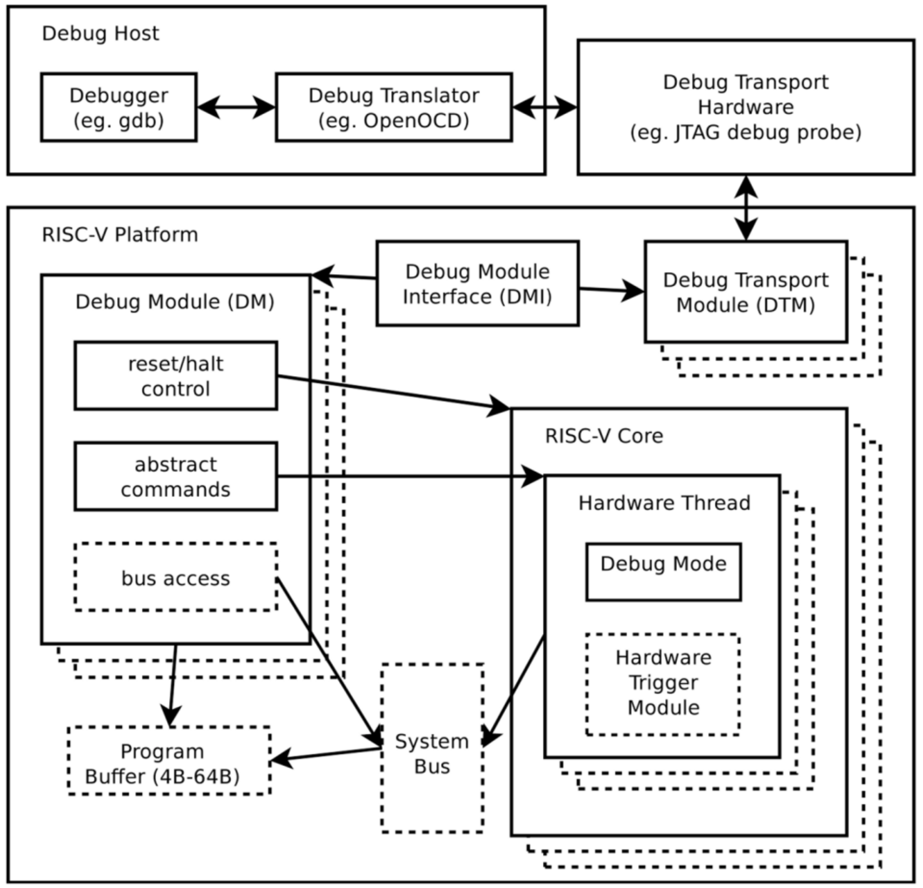
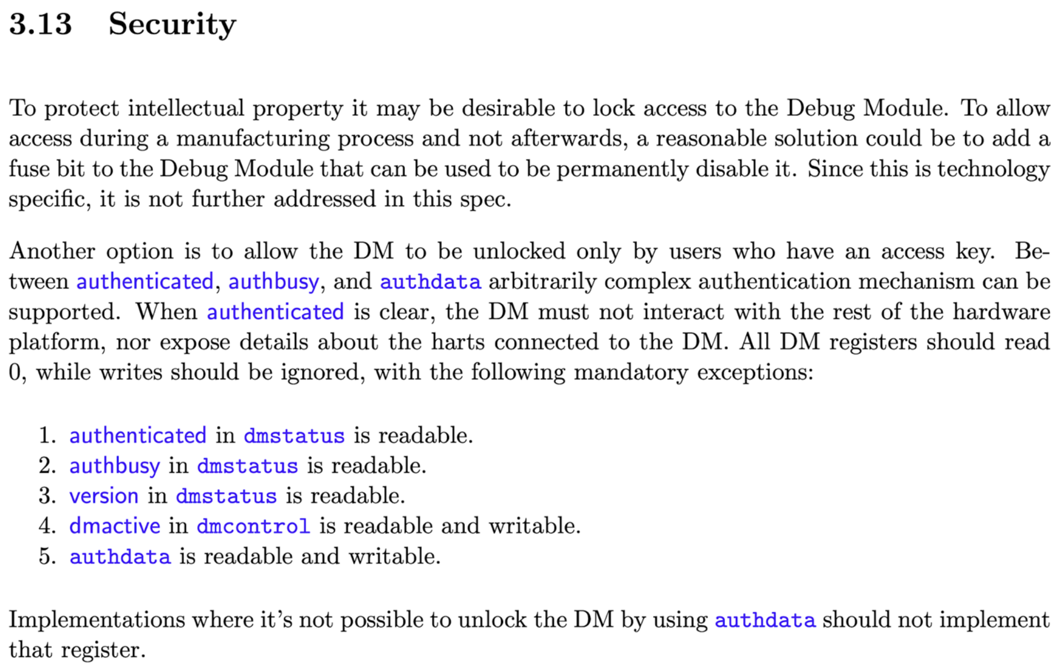


Figure 2.1: RISC-V Debug System Overview

RISC-V DEBUG SECURITY

- All-or-nothing, controlled via fuse or authdata and authenticated
- Once enabled, external debugger has highest priority regardless of target privilege



3.13 Security

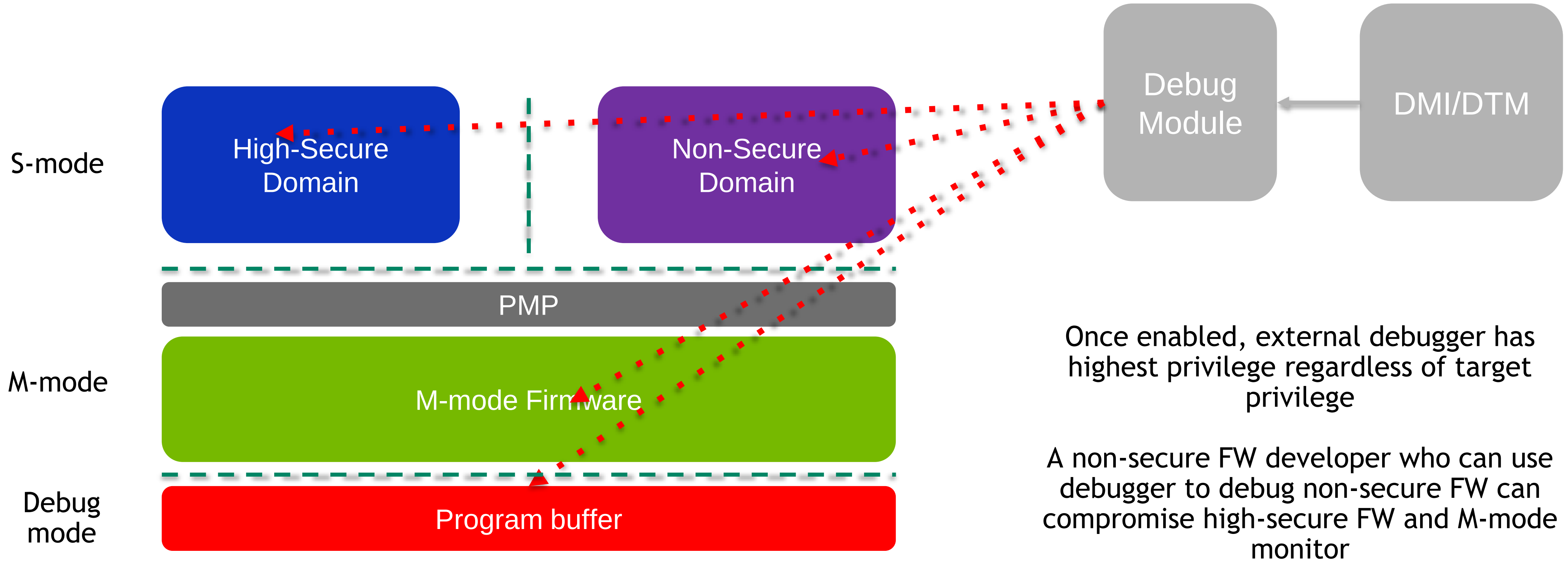
To protect intellectual property it may be desirable to lock access to the Debug Module. To allow access during a manufacturing process and not afterwards, a reasonable solution could be to add a fuse bit to the Debug Module that can be used to be permanently disable it. Since this is technology specific, it is not further addressed in this spec.

Another option is to allow the DM to be unlocked only by users who have an access key. Between authenticated, authbusy, and `authdata` arbitrarily complex authentication mechanism can be supported. When authenticated is clear, the DM must not interact with the rest of the hardware platform, nor expose details about the harts connected to the DM. All DM registers should read 0, while writes should be ignored, with the following mandatory exceptions:

1. authenticated in `dmstatus` is readable.
2. authbusy in `dmstatus` is readable.
3. version in `dmstatus` is readable.
4. dmactive in `dmcontrol` is readable and writable.
5. `authdata` is readable and writable.

Implementations where it's not possible to unlock the DM by using `authdata` should not implement that register.

WHAT'S THE ISSUE?



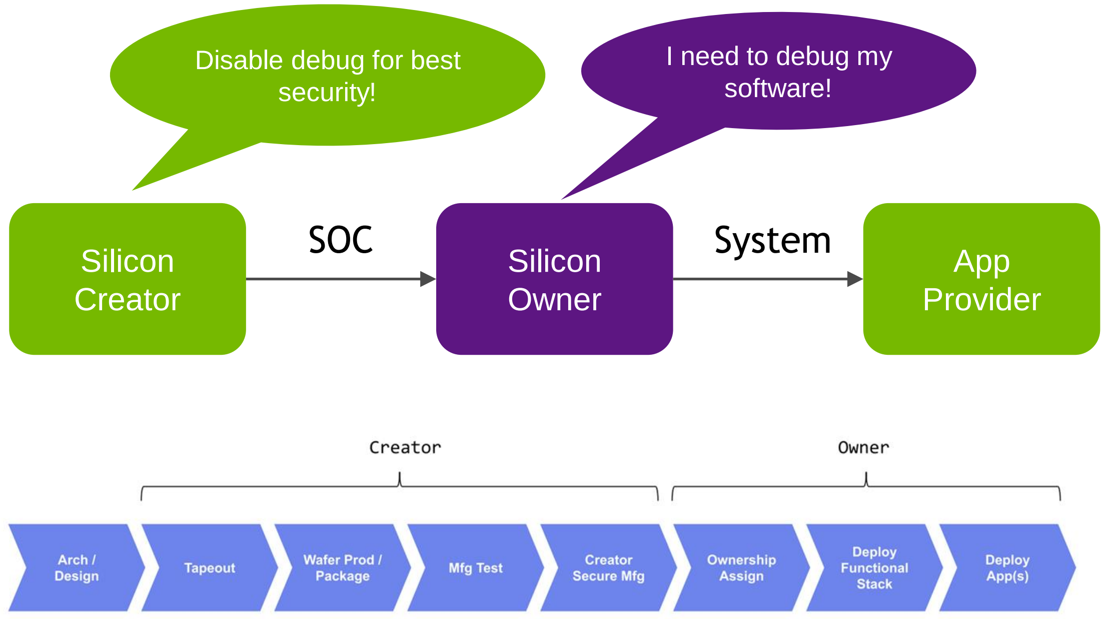
Once enabled, external debugger has highest privilege regardless of target privilege

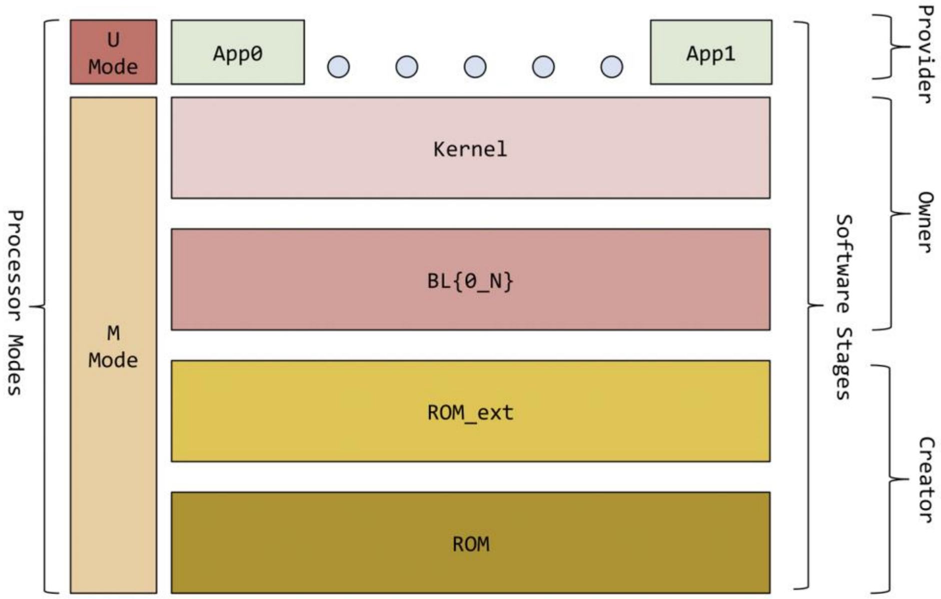
A non-secure FW developer who can use debugger to debug non-secure FW can compromise high-secure FW and M-mode monitor

WHY THIS IS A PROBLEM?

Modern SOC software development consists of multiple actors, they all need external debug

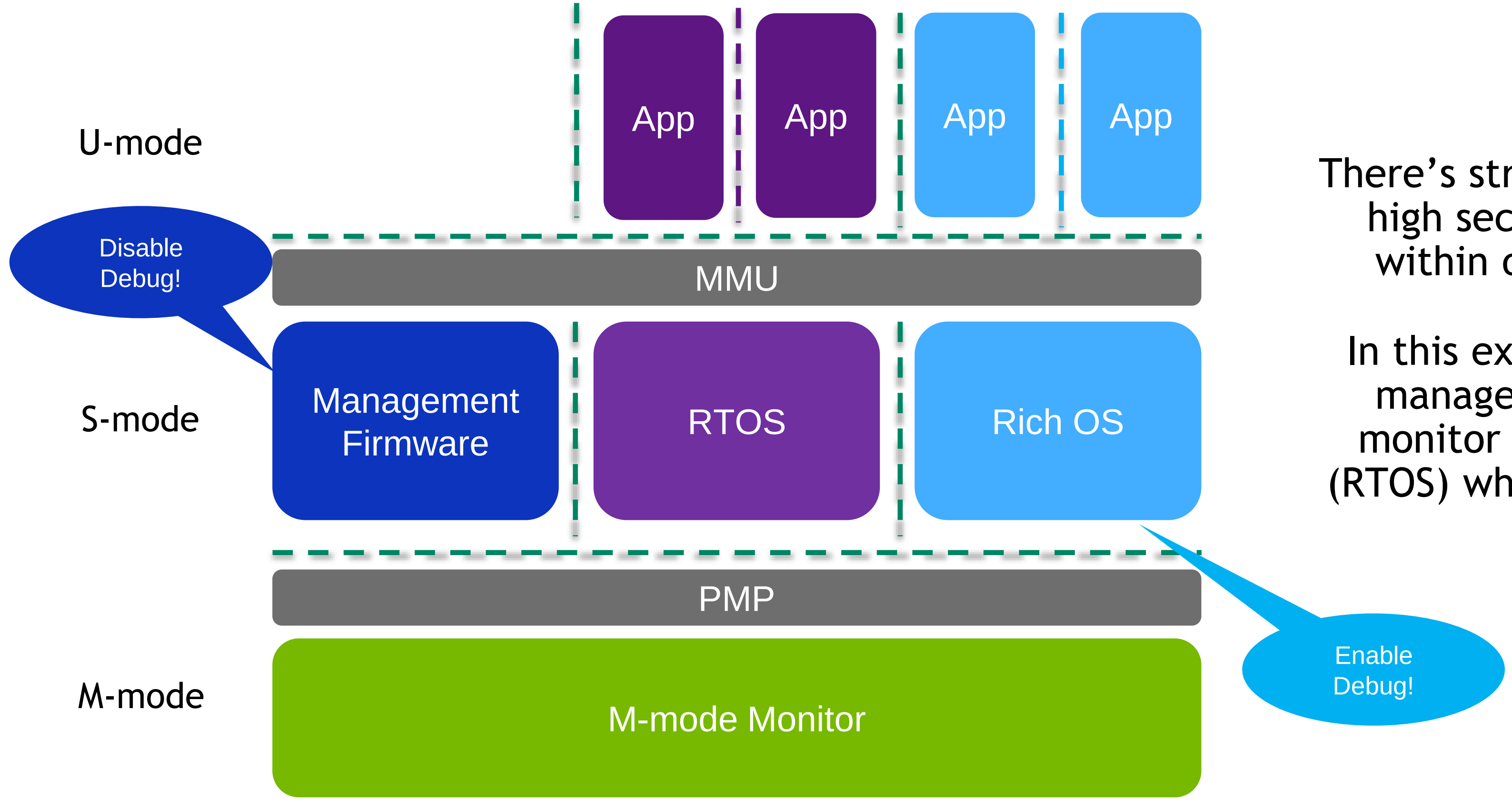
One actor wants to protect its confidential data from another actor (Silicon creator considers Silicon owner as adversary)





Source - Project OpenTitan

WHY THIS IS A PROBLEM?

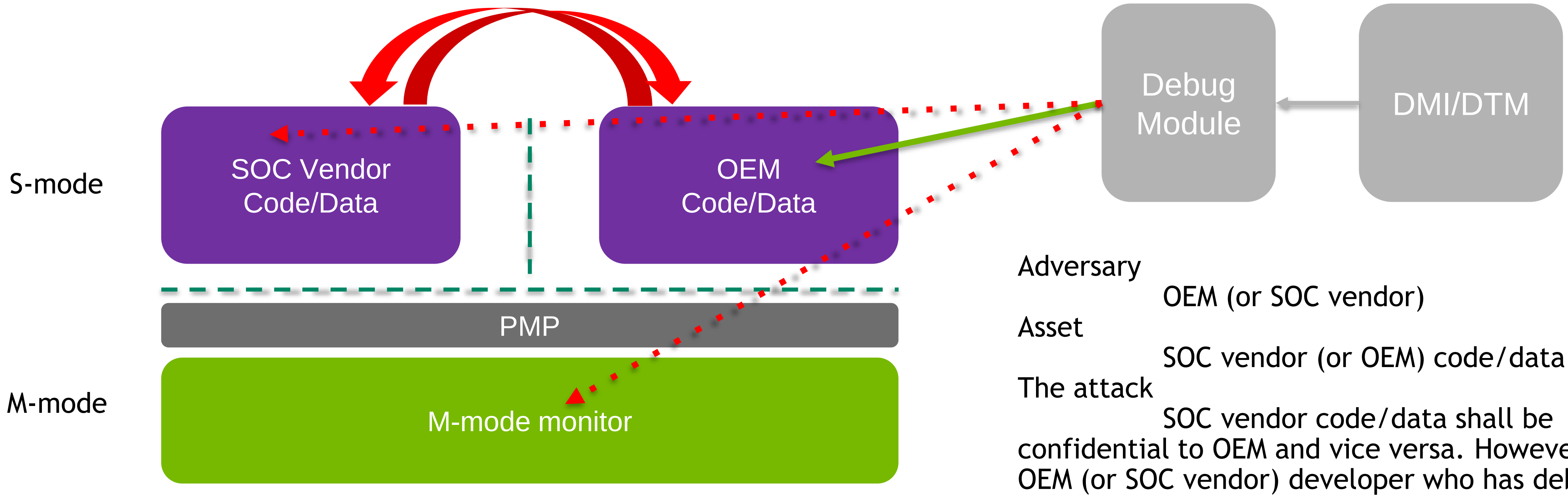


There's strong requirement to protect high security domain's data even within one actor, to reduce TCB

In this example we want to protect management domain or M-mode monitor from low security domain (RTOS) when using external debugger to debug RTOS

EXAMPLE

OEM attack SOC vendor



Adversary

OEM (or SOC vendor)

Asset

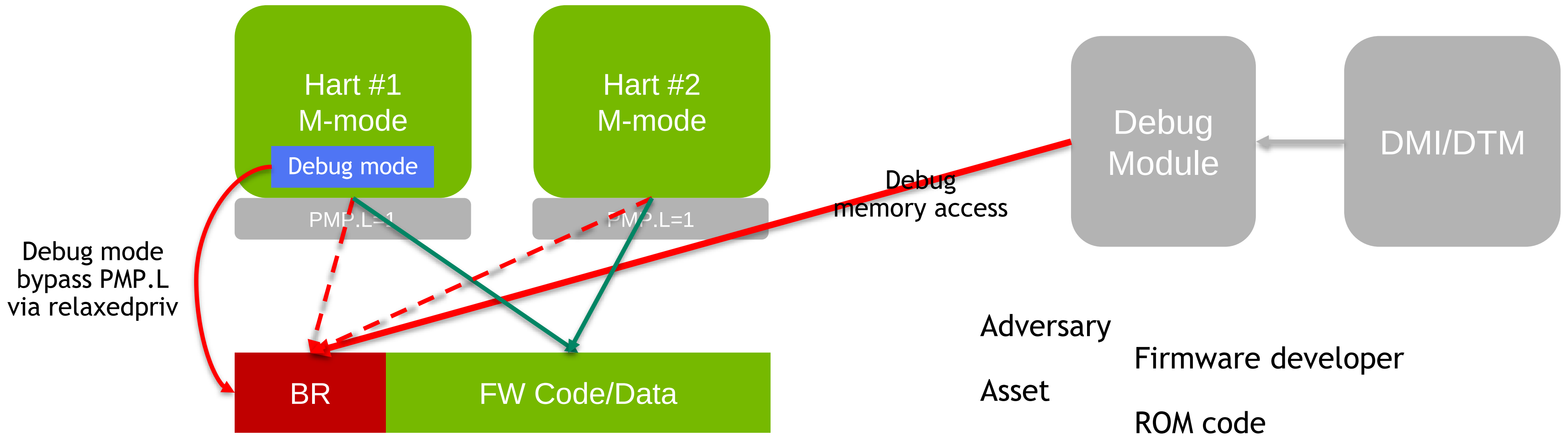
SOC vendor (or OEM) code/data

The attack

SOC vendor code/data shall be confidential to OEM and vice versa. However, an OEM (or SOC vendor) developer who has debug access to OEM code can break PMP protection and steal SOC vendor (or OEM) confidential code/data

EXAMPLE

ROM Dump

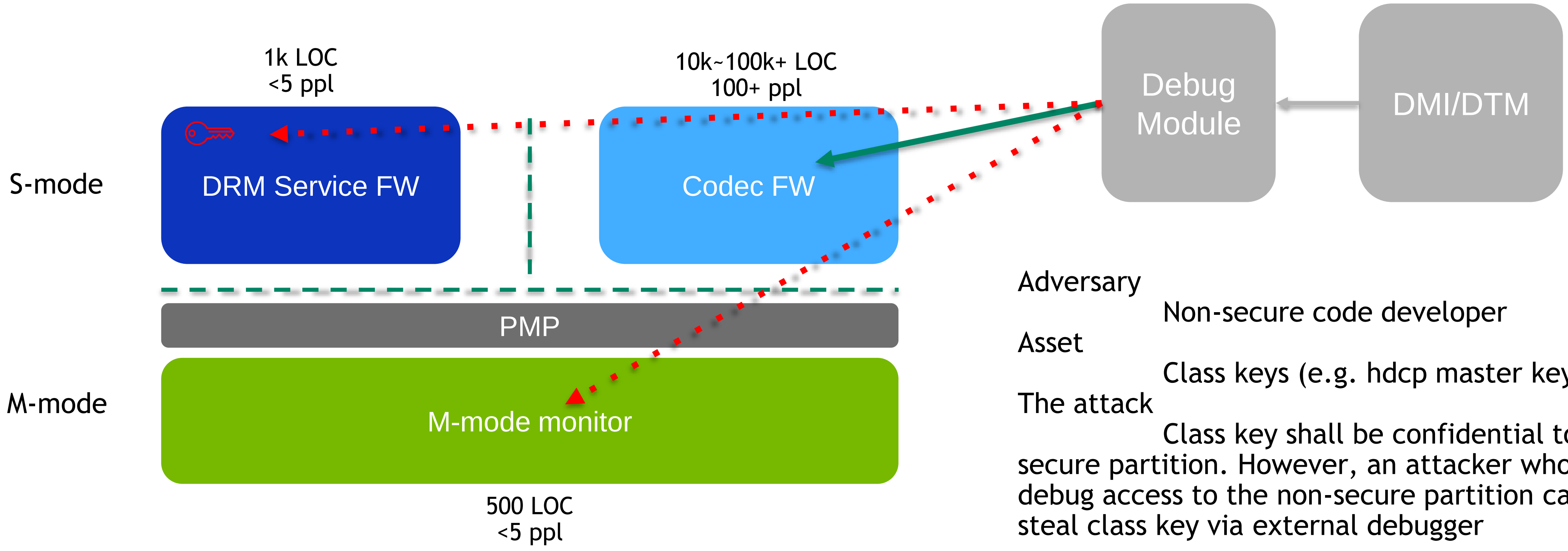


Adversary
Firmware developer

Asset
ROM code

The attack
Boot ROM code shall be confidential to firmware developer. A firmware developer who has external debug access can bypass PMP.L protection to read ROM content, assuming BR protection relies on PMP.L

EXAMPLE



Adversary

Non-secure code developer

Asset

Class keys (e.g. hdcp master key)

The attack

Class key shall be confidential to non-secure partition. However, an attacker who has debug access to the non-secure partition can steal class key via external debugger

COMPETITOR ANALYSIS - ARM

ARMv8 provides rich mechanisms to protect secure data from debugger access

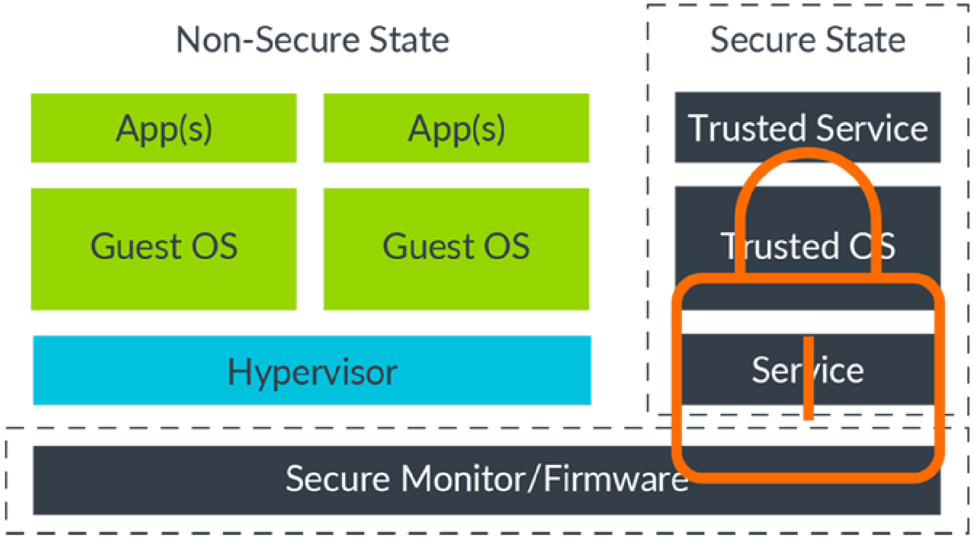
- Controls secure / non-secure / realm separately
- Controls invasive / non-invasive debug separately
- Controls external debug and self-hosted debug separately
- Controls halting debug events sperately
- Consists of both hardware or software

Debug access to memory/register honors target privilege level, hence no privilege escalation

ARM TARGET STATE

An Armv8-A architecture processor or core can have two security states, Secure state and Non-secure state. If the Secure state is implemented, privileged or secret information for the SoC is stored or handled through the Secure state. This means that, after a certain point in the development cycle of an SoC, the hardware or software will lock untrusted users out of the Secure state. This step is taken to prevent access to the protected data in the Secure state.
This diagram shows hardware or software locking a user out of the Secure state:

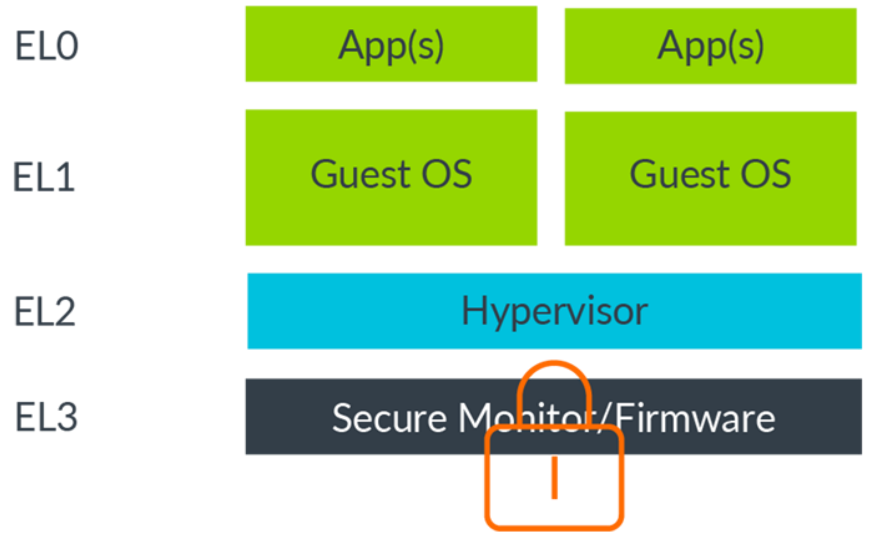
If you try to connect a debugger to an SoC that uses this operational model, then the debugger will only allow you to connect to the Non-secure state.



ARM TARGET STATE

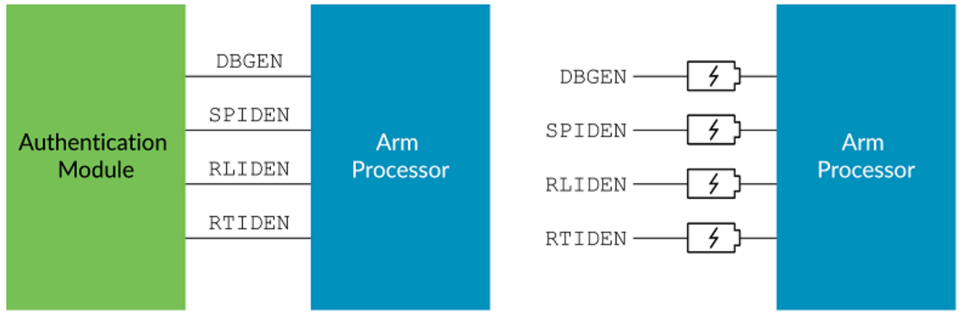
SoC designs can lock users out of certain Exception levels either through hardware or software. This locking usually occurs late in the development cycle of the SoC, so that unprivileged users cannot access aspects of the code or SoC design. This diagram shows an SoC in which EL3, which is usually where the secure monitor and firmware reside, is locked.

If you connected a debugger to an SoC that uses this operational model, then you could debug EL2, EL1 and EL0, but not EL3.



EXTERNAL DEBUG CONTROL

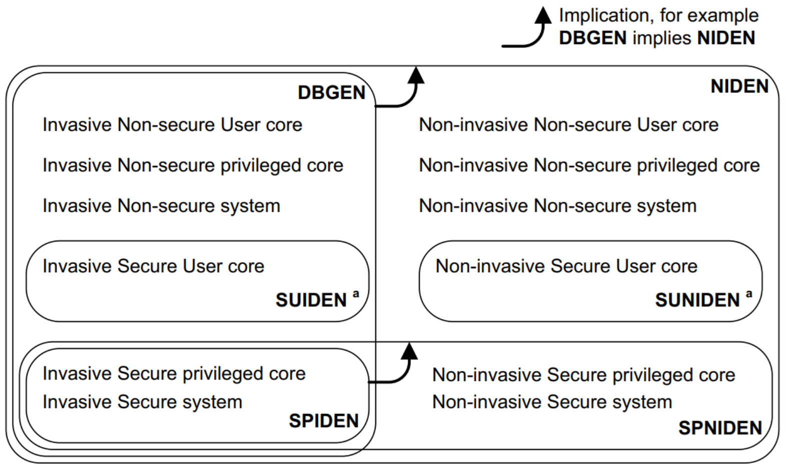
- DBGEN: Top-level invasive debug enable
- NIDEN: Non-secure invasive debug enable
- SPIDEN: Secure invasive debug enable, controls external ability to debug in secure state
- SPNIDEN: Secure non-invasive debug enable, controls external ability to non-invasive debug in secure state
- RLPIDEN: Realm Invasive Debug Enable, controls external ability to debug in realm state
- RTPIDEN: Root Invasive Debug Enable, controls external ability to debug in root state



USER MODE DEBUG

Individual components can offer greater control over the permitted level of debugging. For example, some processors implementing Arm Security Extensions can grant permission to debug-specific Secure processes by permitting debugging of Secure User mode without permitting debugging of Secure privileged modes.

Controlled via SDER (Secure Debug Enable Register)

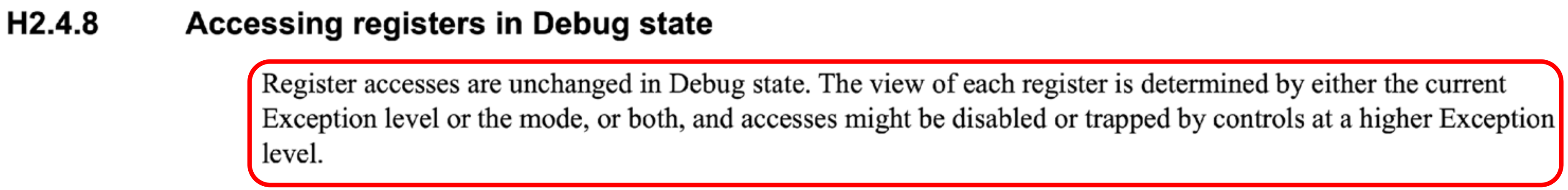


Implication, for example
DBGEN implies NIDEN
DBGEN
Invasive Non-secure User core
Invasive Non-secure privileged core
Invasive Non-secure system
Invasive Secure User core
SUIDEN [a]
Invasive Secure privileged core
Invasive Secure system
SPIDEN
NIDEN
Non-invasive Non-secure User core
Non-invasive Non-secure privileged core
Non-invasive Non-secure system
Non-invasive Secure User core
SUNIDEN [a]
Non-invasive Secure privileged core
Non-invasive Secure system
SPNIDEN

Pin control
[a] Indicates control by the Secure operating system

ACCESSING REGISTER IN DEBUG STATE

ARM DDI 0487J.a ID042523 a-profile_architecture_reference_manual H2-11113

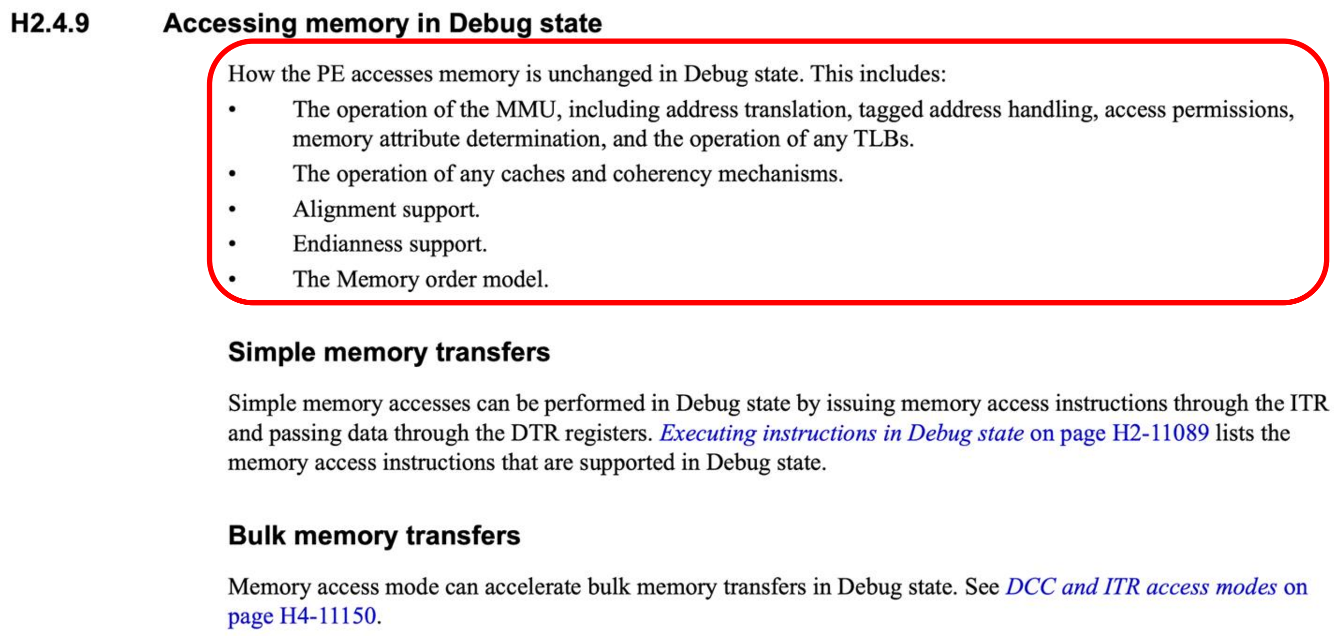


H2.4.8 Accessing registers in Debug state

Register accesses are unchanged in Debug state. The view of each register is determined by either the current Exception level or the mode, or both, and accesses might be disabled or trapped by controls at a higher Exception level.

ACCESSING MEMORY IN DEBUG STATE

ARM DDI 0487J.a ID042523 a-profile_architecture_reference_manual H2-11115



H2.4.9 Accessing memory in Debug state

How the PE accesses memory is unchanged in Debug state. This includes:

- The operation of the MMU, including address translation, tagged address handling, access permissions, memory attribute determination, and the operation of any TLBs.
- The operation of any caches and coherency mechanisms.
- Alignment support.
- Endianness support.
- The Memory order model.

Simple memory transfers

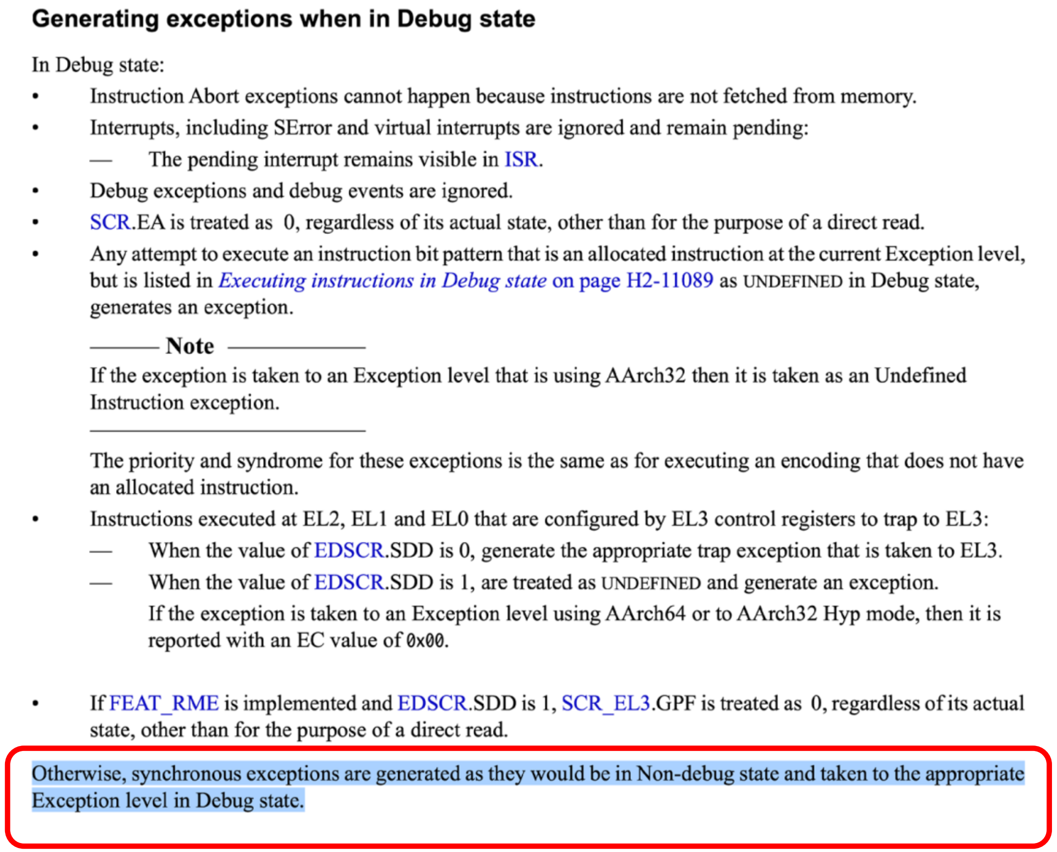
Simple memory accesses can be performed in Debug state by issuing memory access instructions through the ITR and passing data through the DTR registers. *Executing instructions in Debug state* on page H2-11089 lists the memory access instructions that are supported in Debug state.

Bulk memory transfers

Memory access mode can accelerate bulk memory transfers in Debug state. See *DCC and ITR access modes* on page H4-11150.

EXCEPTION IN DEBUG STATE

ARM DDI 0487J.a ID042523 a-profile_architecture_reference_manual H2-11110



Generating exceptions when in Debug state

In Debug state:

- Instruction Abort exceptions cannot happen because instructions are not fetched from memory.
- Interrupts, including SError and virtual interrupts are ignored and remain pending:
  - — The pending interrupt remains visible in ISR.
- Debug exceptions and debug events are ignored.
- SCR.EA is treated as 0, regardless of its actual state, other than for the purpose of a direct read.
- Any attempt to execute an instruction bit pattern that is an allocated instruction at the current Exception level, but is listed in *Executing instructions in Debug state* on page H2-11089 as UNDEFINED in Debug state, generates an exception.

  Note

  If the exception is taken to an Exception level that is using AArch32 then it is taken as an Undefined Instruction exception.

  The priority and syndrome for these exceptions is the same as for executing an encoding that does not have an allocated instruction.
- Instructions executed at EL2, EL1 and EL0 that are configured by EL3 control registers to trap to EL3:
  - — When the value of EDSCR.SDD is 0, generate the appropriate trap exception that is taken to EL3.
  - — When the value of EDSCR.SDD is 1, are treated as UNDEFINED and generate an exception.

    If the exception is taken to an Exception level using AArch64 or to AArch32 Hyp mode, then it is reported with an EC value of `0x00`.
- If FEAT_RME is implemented and EDSCR.SDD is 1, SCR_EL3.GPF is treated as 0, regardless of its actual state, other than for the purpose of a direct read.

Otherwise, synchronous exceptions are generated as they would be in Non-debug state and taken to the appropriate Exception level in Debug state.

The Proposal

Secure Debug Extension Proposal

Requirements

The debug accesses shall be regulated according to the its privilege level (assigning a privilege level to debug access).

Less privileged debug accesses cannot peep/interrupt the hart when it runs in higher privilege level (e.g., S mode debug privilege cannot read/halt the trap handler or context switch in M mode).

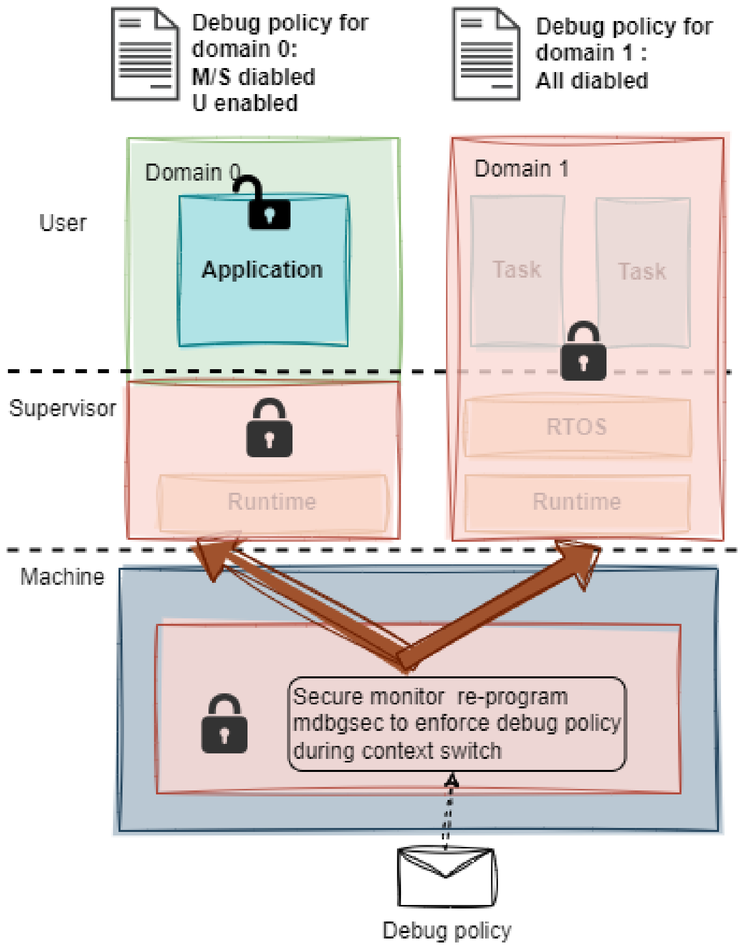
Less privileged debug accesses cannot tamper resources of higher privileged level (e.g., S mode debug privilege level to access M mode CSR or memory granted to M mode by PMP).

The debug access can be conditionally enabled. (e.g., both ROM and Non-ROM can live in M mode, but the debuggability should be granted differently).

Memory accesses from System Bus Block shall be regulated by IOPMP or something equivalent.

Secure Debug Extension

Overview



Debug policy for domain 0:
M/S diabled
U enabled
Debug policy for domain 1 :
All diabled
User
Domain 0
Application
Domain 1
Task
Task
Supervisor
RTOS
Runtime
Runtime
Machine
Secure monitor re-program mdbgsec to enforce debug policy during context switch
Debug policy

- Debug policy defines whether external debugger is enabled or disabled for each domain
  - A piece of manifest as input for M-mode firmware
  - Specifies mdbgsec configuration for domains
- M-mode firmware to enable/disable external debugger access for each domain via mdbgsec
- External Debugger (debug module) access regulated by privilege levels
- Implementations can extend the control to permanently disable external debug for high secure domains

Secure Debug Extension

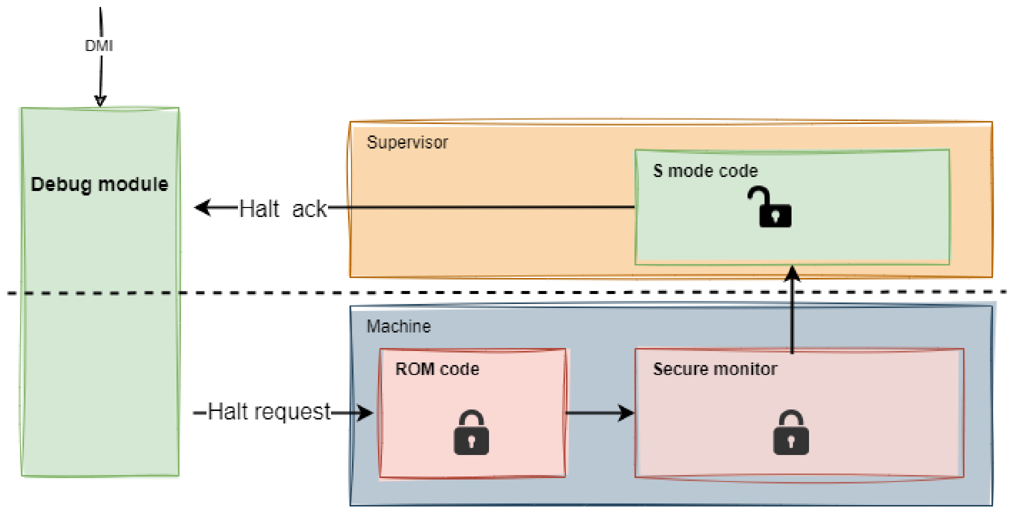
MDBGSEC CSR (fields for debug access control)

- Add Machine Debug Security Control Register (mdbgsec)
  - Global debug enable knob
  - Controls the maximum privilege level

H ext. supported	dbgen (bit3)	dbgv (bit2)	dbgprv (bit0-1)	Max debug prv mode
No	0	0	don't-care	External debug disabled
No	1	0	0	U mode external debug enabled
No	1	0	1	U/S mode external debug enabled
No	1	0	3	U/S/M mode external debug enabled
Yes	0	dont' care	don't care	External debug disabled
Yes	1	0	0	U mode external debug enabled
Yes	1	0	1	U/VU/VS/HS mode external debug enabled
Yes	1	0	3	U/VU/VS/HS/M mode external debug enabled
Yes	1	1	0	VU mode external debug enabled
Yes	1	1	1	VS/VU mode external debug enabled

Secure Debug Extension

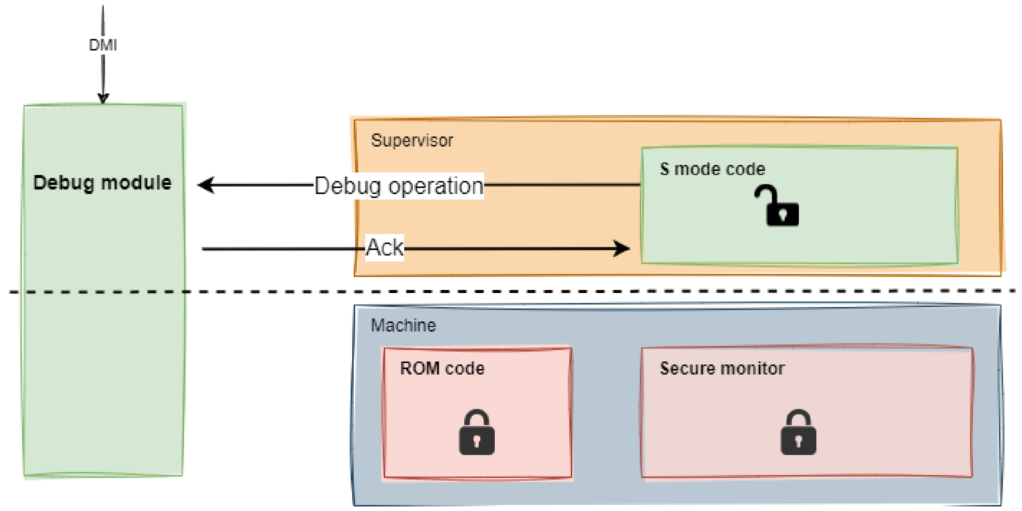
Debug module is regulated by privilege level



- Query the allsecure/anysecure to discover whether the hart is debug-able
- Halt request behavior changes as the following
  - if debug is enabled in any mode, halt request will be pending till the hart can eventually be halted
  - Dropped silently if not debug-able

Secure Debug Extension

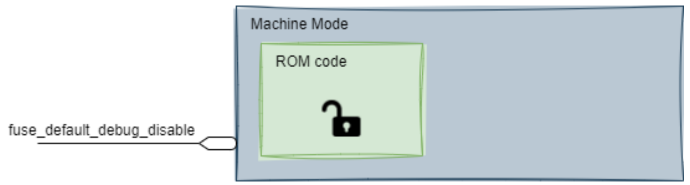
Debug module access regulated by privilege level



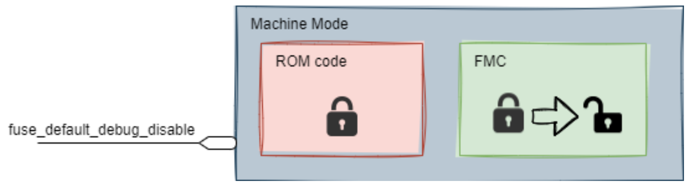
- Abstract commands
  - memory and registers access will be checked as if they are at privilege level specified in dcsr.prv/dcsr.v.
- Program buffer
  - Instructions will work as if the hart is running at privilege level specified in dcsr.prv/dcsr.v (except ecall/ebreak/xret).
- Writing dcsr.prv/dcsr.v with a value whose corresponding privilege level is disabled for debug will get security fault error

Secure Debug Extension

Extending the debug control management with hardware fusing and sticky knob



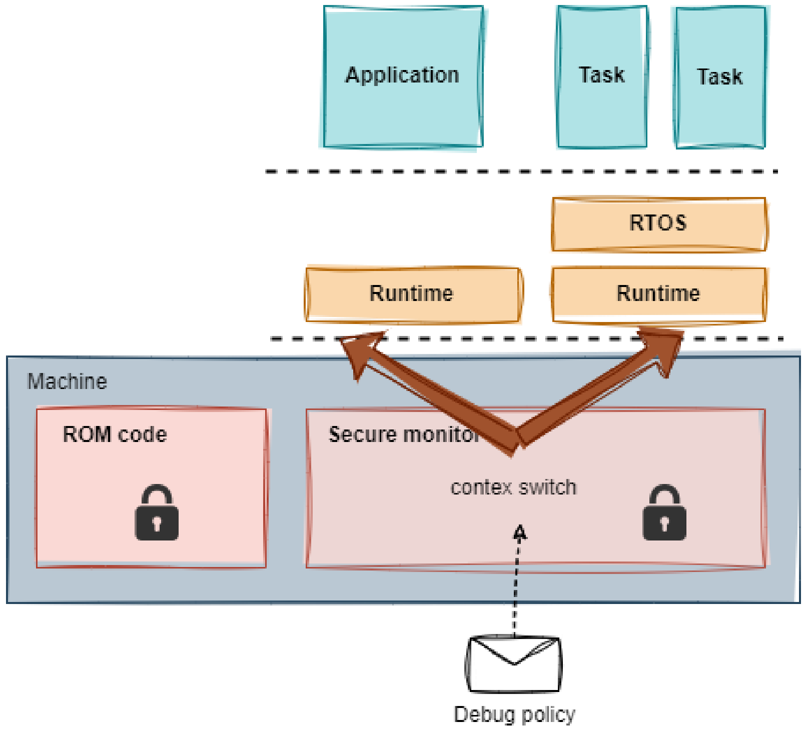
- Phase 1: Debug ROM
  - A life-cycle fuse to determine default value of mdbgsec
  - All privilege levels are allowed to debug if the fuse is not burnt



- Phase 2: Lockdown ROM and debug FMC
  - Burn the fuse to disable debug by default
  - FMC is responsible to enable debug by configuring mdbgsec.
  - FMC waits the external debugger to hook up after enabling debug.

Secure Debug Extension

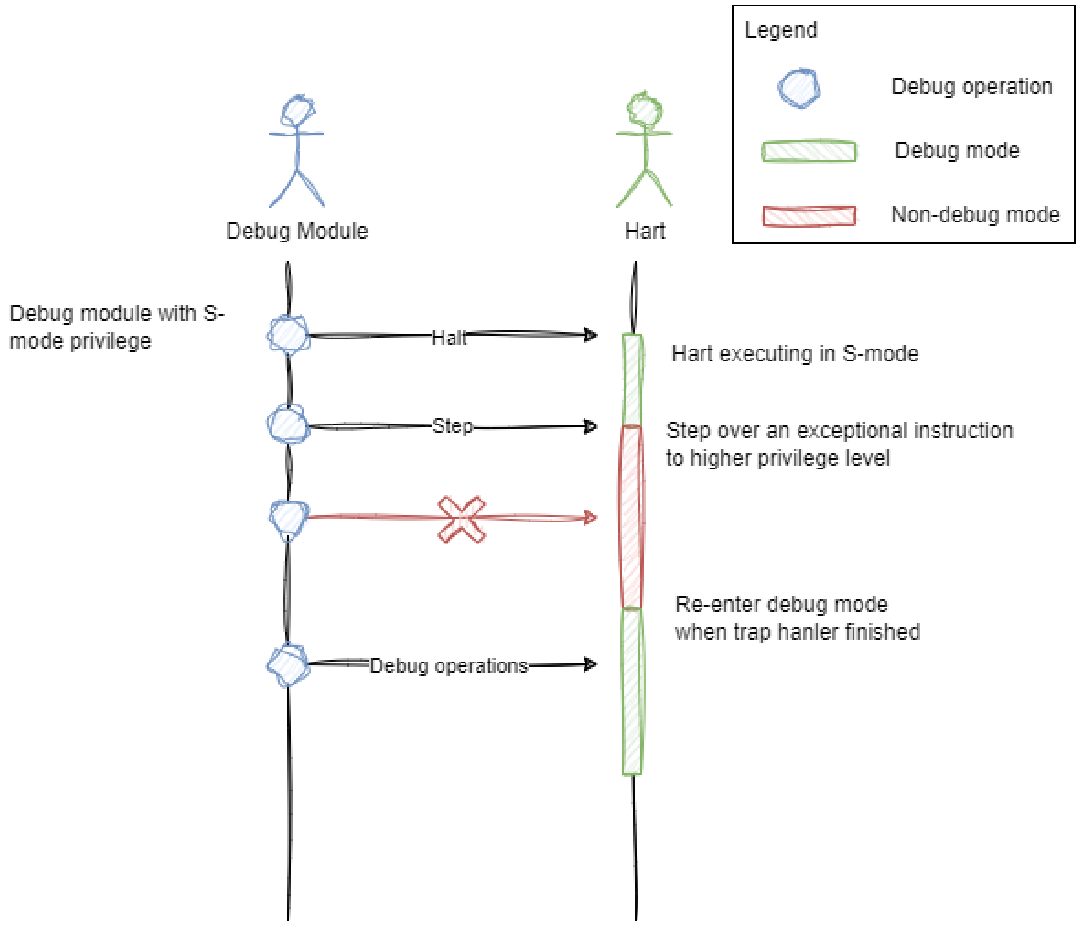
Manage the debug with hardware fusing and sticky knob



- Phase 3: Disable machine mode debug
  - Program the sticky knob in mdbgsec to disable machine mode debug
  - The debug could only be granted to non-machine mode
  - The knob is sticky and cannot be cleared until reset.

Exceptions and Interrupts

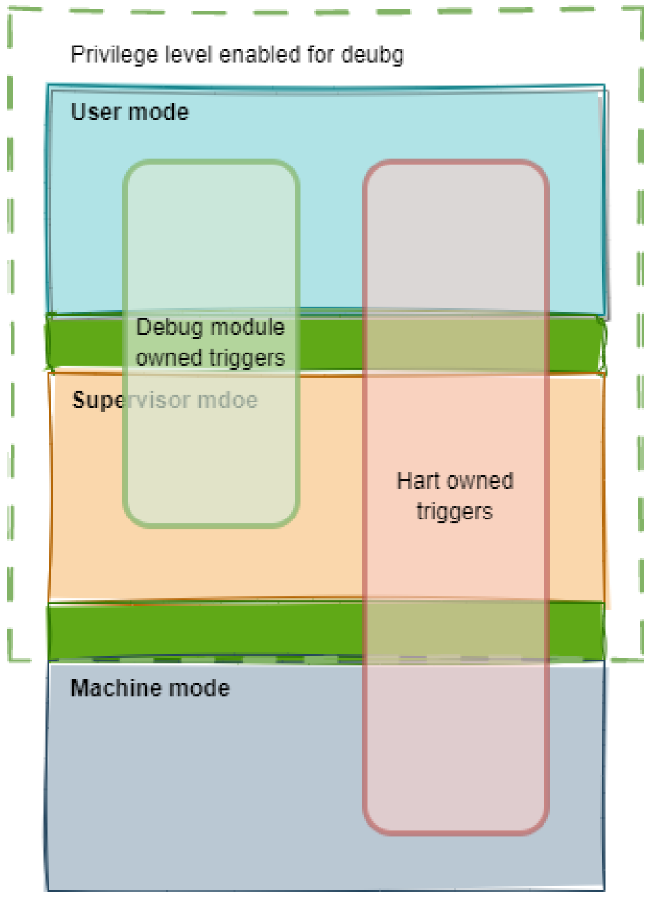
A case study of stepping over traps


Legend
Debug operation
Debug mode
Non-debug mode
Debug Module
Hart
Debug module with S-mode privilege
Halt
Hart executing in S-mode
Step
Step over an exceptional instruction to higher privilege level
Re-enter debug mode when trap hanler finished
Debug operations

- The hart will exit debug mode during step
- If exception or interrupt occurs and it lands at higher privilege level
- The step cannot be completed
- Hart continues execution in high privilege level
- Re-enter debug mode immediately after returning to debug-able privilege level.

Triggers

Debug module is regulated by privilege level


Privilege level enabled for deubg
User mode
Debug module owned triggers
Supervisor mdoe
Hart owned triggers
Machine mode

- Triggers match/fire to enter debug mode if debug is allowed in mdbgsec
- Triggers cannot be enabled for unallowed privilege by external debugger.
- External debugger cannot modify hart owner triggers.

Reset And Power/clock gating

- Per hart operation:
  - hartreset/resethaltreq will be served only if the external debugger has M-mode debug privilege, otherwise they will be dropped silently

- Systemwide operation.:
  - It is recommended that SOC vendors do not implement ndmreset, or use a life-cycle fuse to disable ndmreset.

Conclusion

We believed the use case and security requirements are general, all RISC-V SOC vendors will face the same problem sooner or later

It turns out that the security issues are addressed by competitor (ARM)

NVidia team has a draft proposal (here), the proposal is in its early phase and actively discussed in multiple groups (RTI, Debug)

We call for RISC-V security community to acknowledge the issues and work together on an extension to fix the issues

NVIDIA®

REFERENCE

ARM Architecture Reference Manual Security Extensions Supplement

https://developer.arm.com/documentation/ddi0601/2023-03/External-Registers?lang=en

https://developer.arm.com/documentation/ddi0487/latest/

https://developer.arm.com/documentation/102120/0101