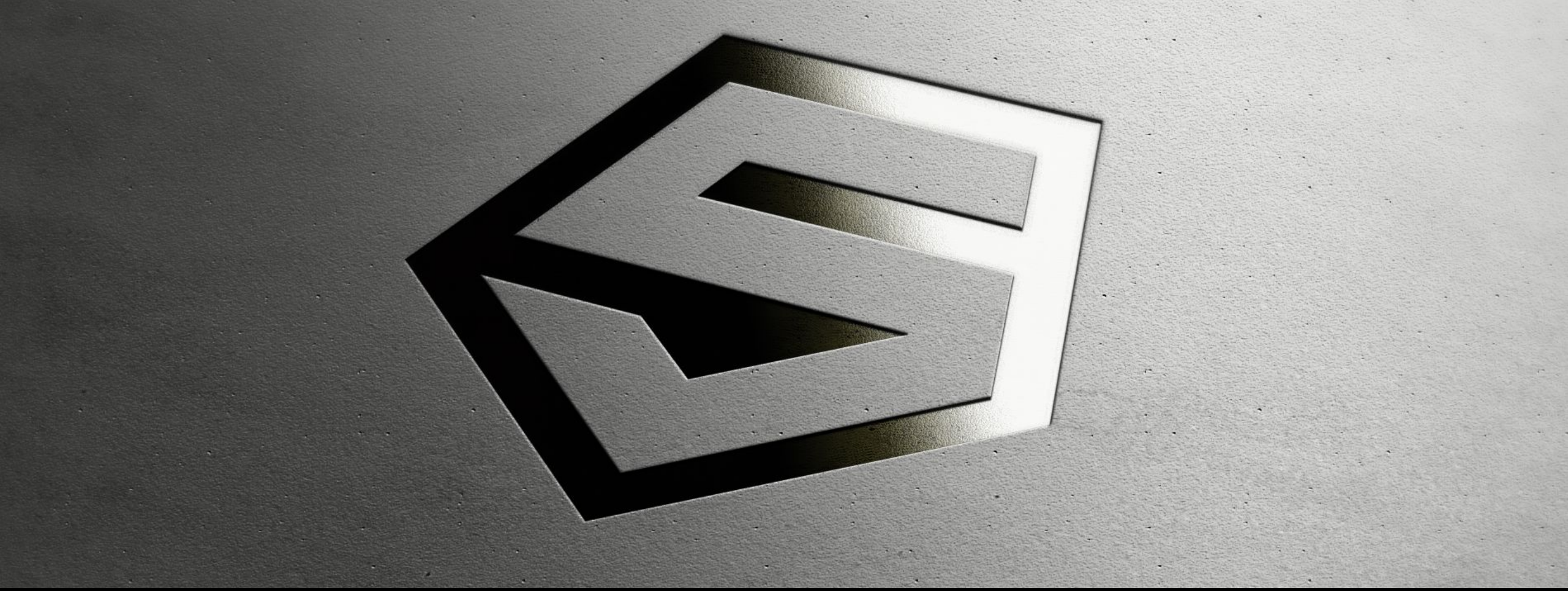


RISC-V Debug Spec Update

6th RISC-V Workshop
Shanghai, China

Megan Wachs

May 9, 2017

Outline

- Debug Working Group Intro
- Why does RISC-V need an External Debug Spec?
- Overview of the Debug Spec v. 013
- Future tasks of Debug Working Group

Debug Working Group Intro

History of the current Specification

- Started out as an email list
- Formed task group with regular meetings at last RISC-V Workshop

- Lots of discussion around two basic mechanisms
- Took a non-binding “opinion poll” of the member companies
- Strong desire for a unified spec
  - To avoid fragmentation of RISC-V ecosystem
  - Working group agreed to pursue that and see where it led
- WG converged on a spec item by item

Debug Working Group Intro

History of the current Specification

- Specification source is available on github:

  https://github.com/riscv/riscv-debug-spec

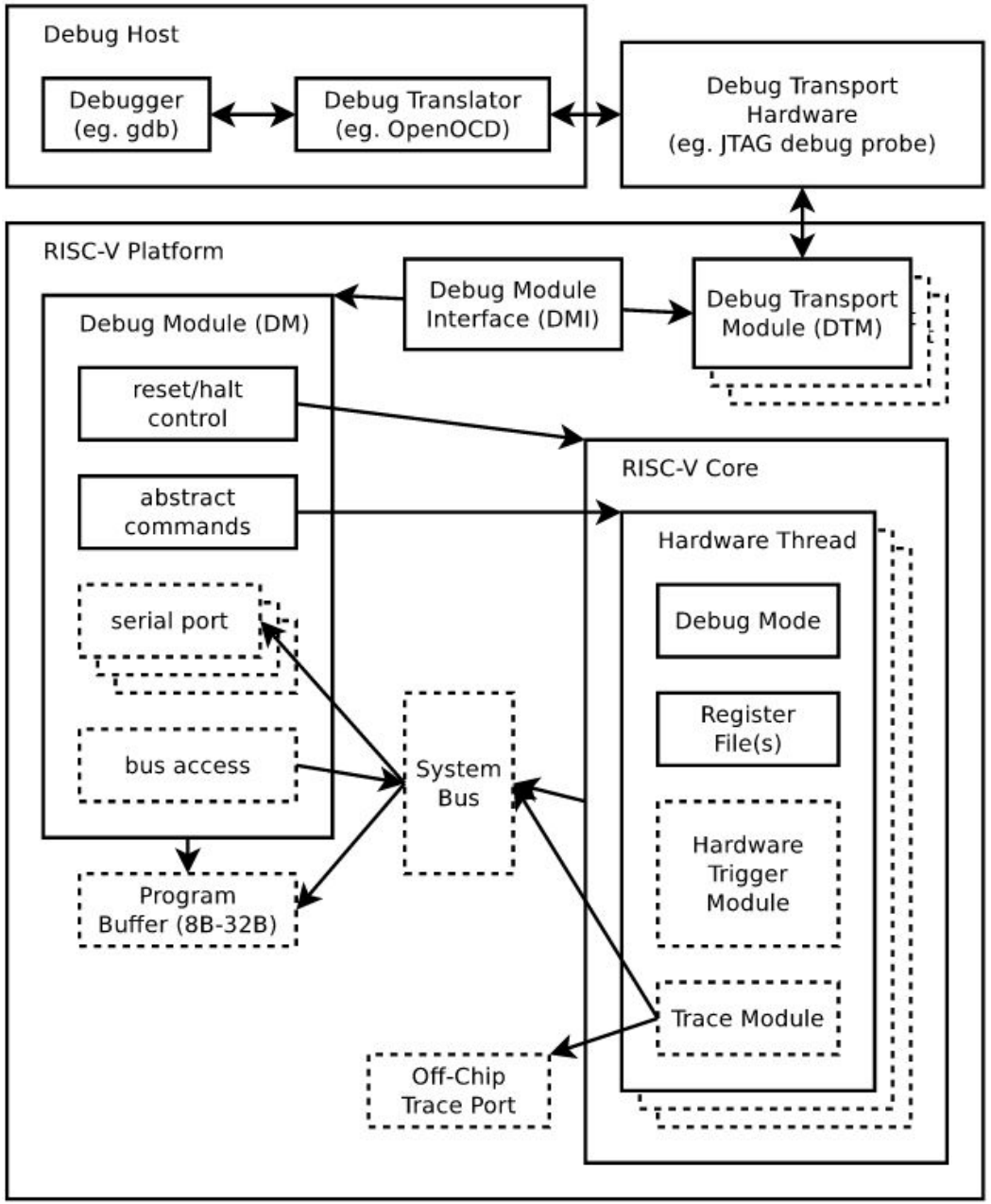
- Pre-compiled PDFs regularly posted on SiFive website:

  https://www.sifive.com/documentation/risc-v/risc-v-external-debug-support/

Debug Working Group Intro

Active members from many member companies

- Alex Bradbury (lowRISC)
- Palmer Dabbelt (SiFive)
- Vyacheslav Dyanchenko (Syntacore)
- Cyril Jean (Microsemi)
- Richard Hereville (ROALogic)
- Po-Wei Huang (Individual)
- Gajinder Panesar (UltraSoC)
- Larry Madar (Google)
- Tim Newsome (SiFive)
- Megan Wachs (SiFive)
- Stefan Wallentowitz (lowRISC)
- ... and many more



External Debug Definition

- Provide visibility into system from external hardware interface
- Access hart registers, memory, devices
- RISC-V Core is only part of a platform, but Spec focuses on the core
- Share debug hardware between multiple harts

Why does RISC-V need a Debug Spec?

- Essential tool for any real hardware
- Debug embedded software on simple systems
- Debug kernel issues on more complex systems
- Perform bringup and test before SW is up and running
- NOT intended to find HW faults/bugs
  - But can be used to narrow them down!

Why does RISC-V need a Debug Spec?

- "Software is King"
- Develop HW and SW debuggers that can work for any RISC-V core
- Not dependant on the quality of a Vendor's debugging toolchain

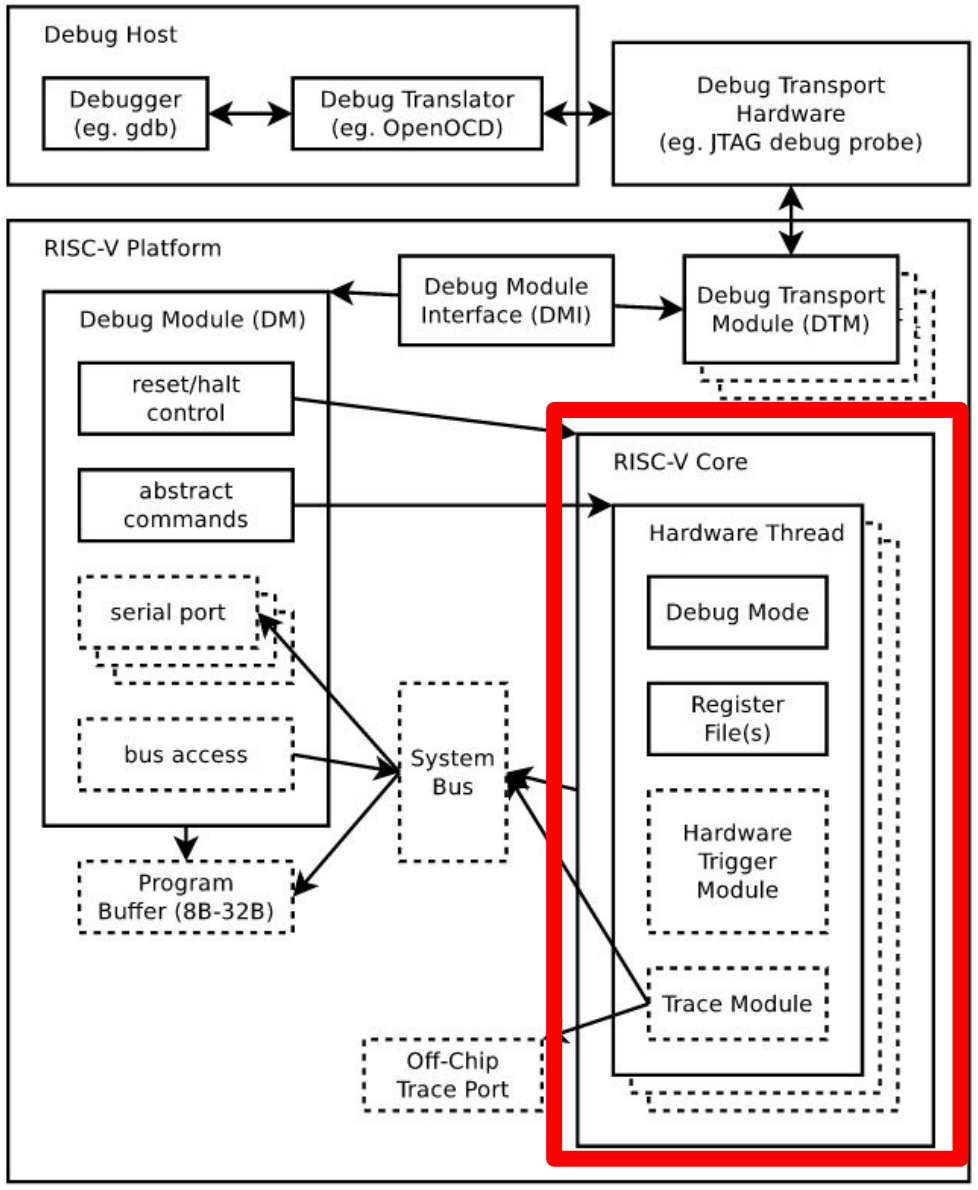
Goal of specification:

A debugger can connect "blind" to any RISC-V platform, and discover everything it needs to know.

Debug Spec v0.13

Basic Features

- Selecting Harts
- Halt - Resume
- Abstract Commands
- Program Buffer
- Single - Stepping
- Debugging across reset / power down
- Triggers

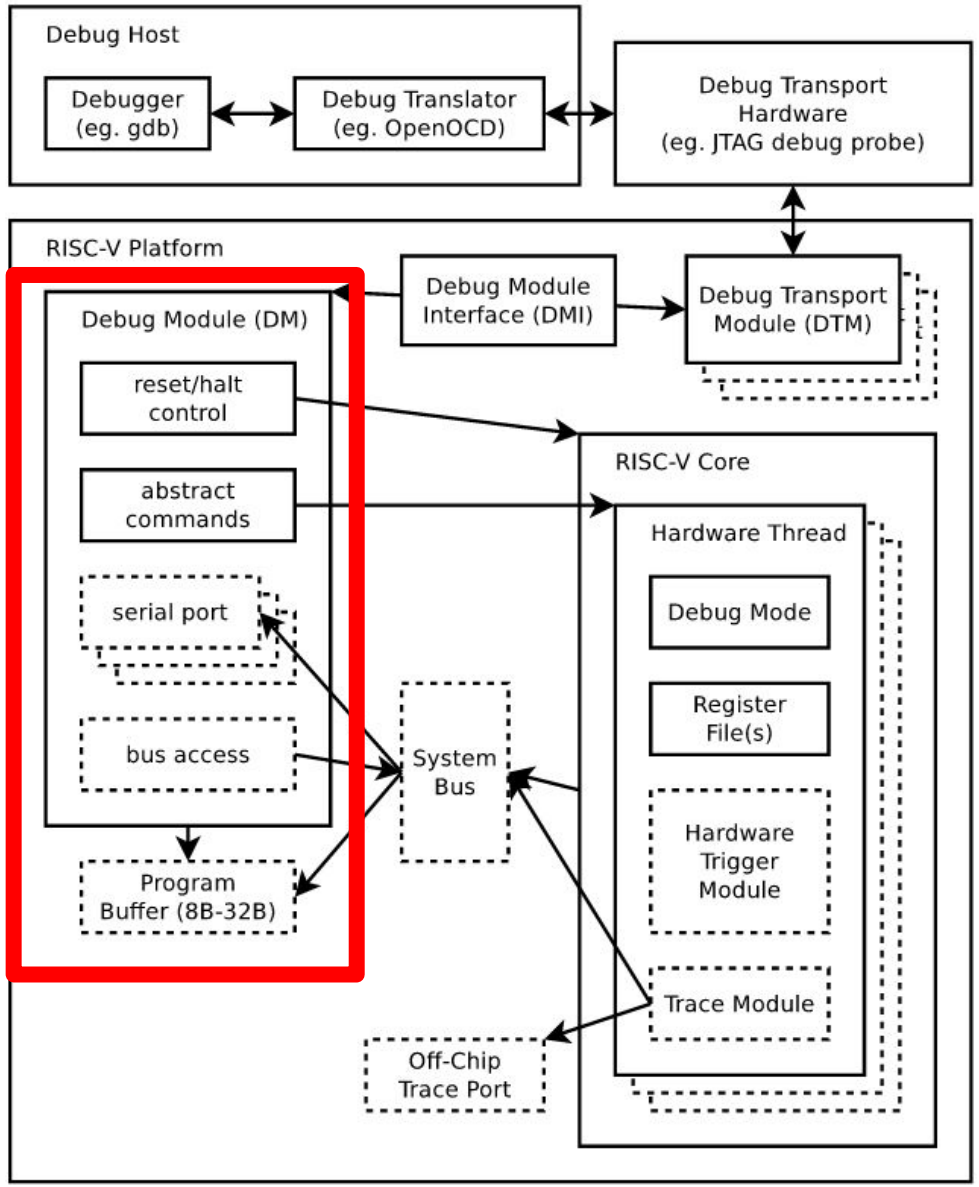


Requirements on RISC-V Harts

- "Debug" execution mode
  - Waiting for instruction from debugger
  - Generally acts like M-Mode
  - Interrupts are disabled
  - Exceptions handled by debugger
  - Can be implemented with simple pipeline stall

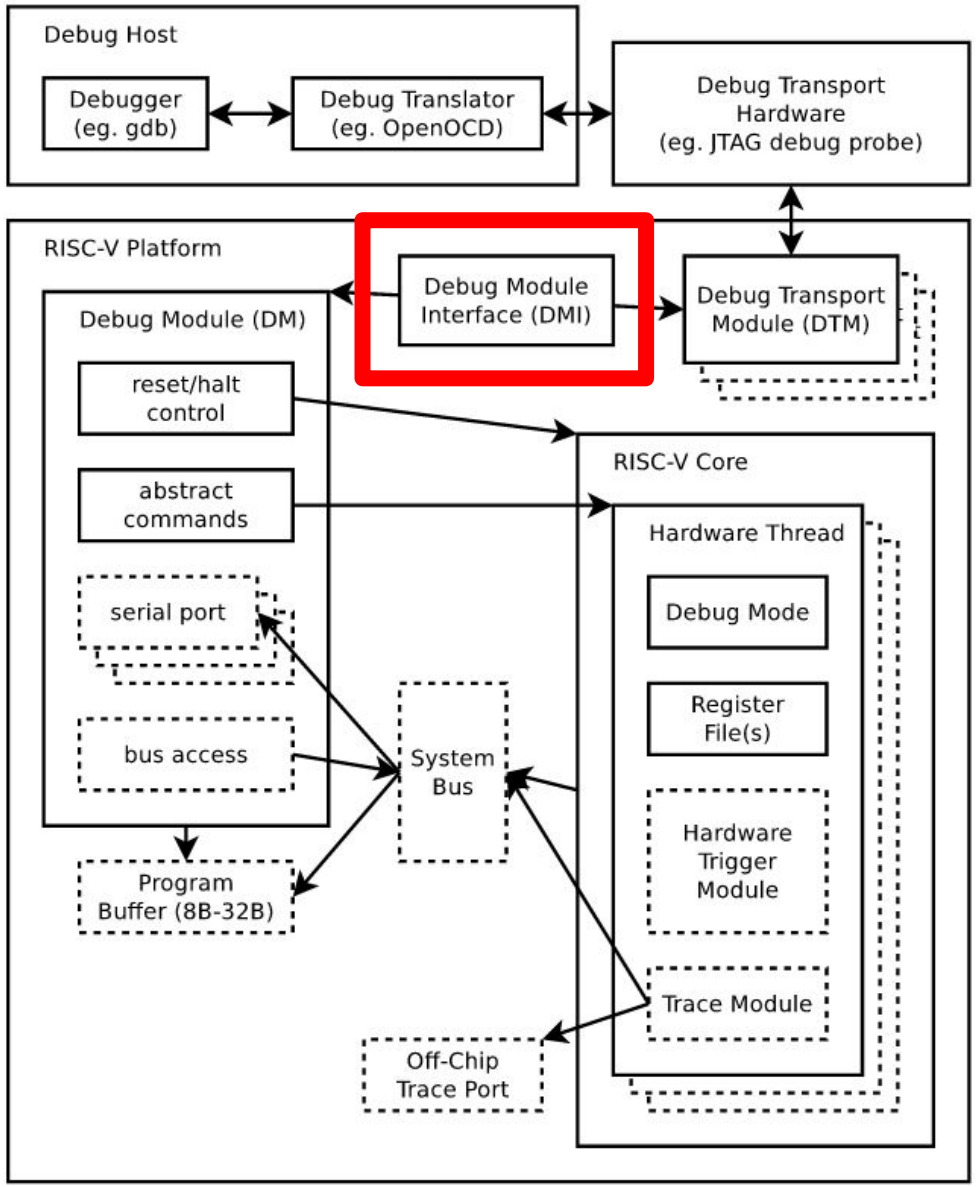
Requirements on RISC-V Harts : CSRs

- $dcsr (~$2^3$ bits)
  - Reports cause of entering debug mode
  - Configures what ‘ebreak’ does
  - Controls Single-step
- $dpc (XLEN bits, optional)
  - Holds a copy of $pc
  - May alias to $pc
- $dscratch(0..N) (XLEN bits, optional)
  - Can be used by debug module/ debugger



Debug Module

- 32-bit register space to control debug functionality
- Specification focuses on this register space
- Details between Debug Module & Core left up to implementation



Debug Module Interface

- Conceptual 32-bit bus
- Connects Debug Transport(s) to Debug Module
- Provides a register-based interface to Debug Module
- Abstraction: could be combined in hardware

Selecting Harts

- Single debug module can support up to 1024 harts
- Debugger writes 10-bit hart selector to DMCONTROL.hartsel
- Hart Selector != $mhartid, but easy to discover mapping
- Optional feature to allow selecting multiple harts

Halt, Resume

To halt, debugger:

1. Selects desired hart(s)
2. Sets DMCONTROL.haltreq
3. Waits for DMSTATUS.allhalted
4. Clears DMCONTROL.haltreq

To resume, debugger:

1. Selects desired hart(s)
2. Sets DMCONTROL.resumereq
3. Waits for DMSTATUS.allresumeack

Abstract Commands

Simple Abstraction for Common Operations

- Read/Write GPRs -- REQUIRED
- Read/Write CSRs -- Optional
- Read/Write FPRs -- Optional
- Can be supported on running harts -- Optional

To perform an abstract command:

1. Debugger writes argument(s) into DATA registers
2. Debugger writes COMMAND register
3. Debugger waits for ABSTRACTCS.busy = 0
4. Debugger reads results from DATA registers

Program Buffer

Optional Extension of Abstract Command

- Allows arbitrary RISC-V Code to be executed by hart
- Flexible way to implement any desired behavior, including future extensions

To execute Program Buffer:

1. Write desired code into PROGBUF registers (ending with ‘ebreak’)
2. Write COMMAND register with ‘postexec’ bit set
3. Wait for ABSTRACTCS.busy = 0

Program Buffer

Optional Extension of Abstract Command

- Program Buffer may be addressable RAM
- Debugger can determine this by executing programs
- If it is, more flexibility (can use Program Buffer to pass more data)
- DATA registers may be mapped as RAM or CSRs, reported in HARTINFO reg.

Batching Commands

Supported for Program Buffer and Abstract Commands

- Debugger can write bursts of commands without checking busy
  - Check DMSTATUS.cmderr at the end, replay if necessary
- AUTOEXEC functionality replays commands with different data
- Low-overhead burst data transfers

Single Stepping

- Enabled by ‘step’ bit in DCSR
- Once hart is resumed, hart executes a single instruction before returning to debug mode

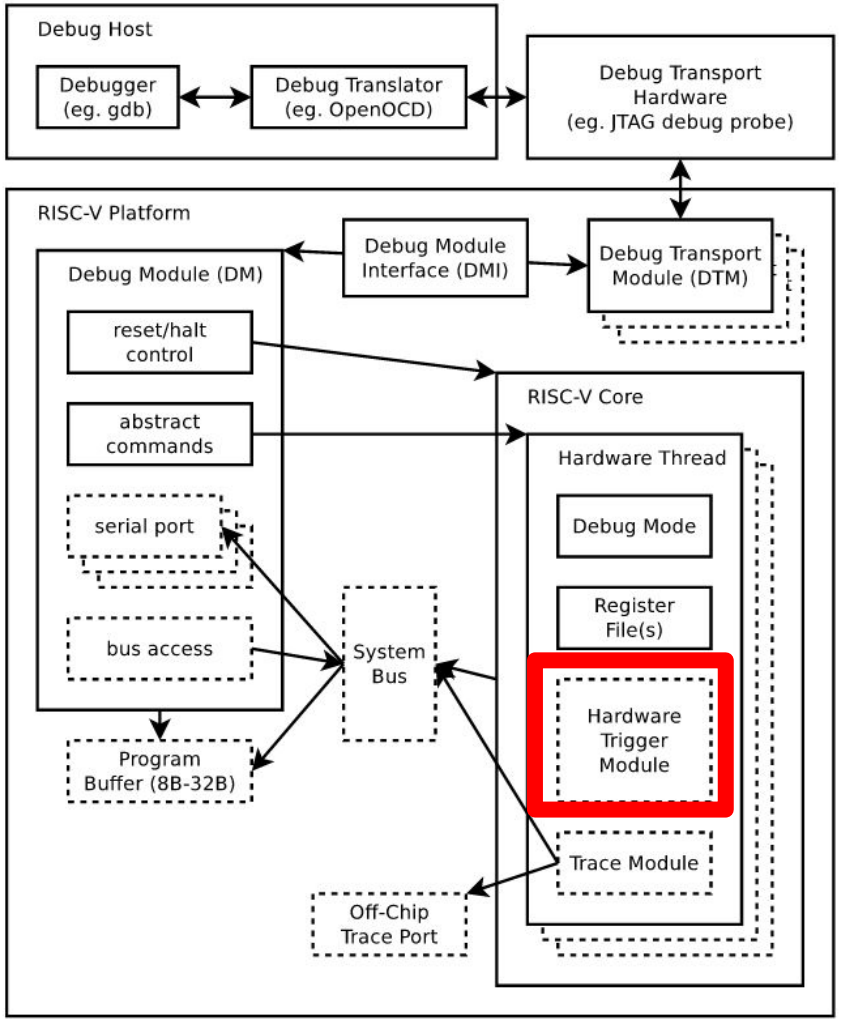
Debugging Across Reset/Powerdown

- Debug logic reset behavior clearly specified
- Debugger can set DMCONTROL.ndmreset
  - What parts of system are reset is implementation-specific
  - Debug logic is not affected by external reset

- Debugger must set DMCONTROL.dmactive
  - Implementations can use this indicator to prevent power gating, etc

Triggers (not part of Debug Module)

- CSRs in the core, shared with M-Mode
- Support up to 2^{XLEN} triggers
- Trigger select, trigger type: WARL fields
- 2 types of triggers currently defined:
  - Virtual address and/or data match
  - Instruction count
- Debugger polls DMSTATUS.haltsum to see if harts have halted



SiFive

Additional Optional Features

See Spec for Details

- System Bus Mastering
- Serial Port Interface
- Quick Access

Public Implementations of Debug v0.13

Would love to hear about more

- Debugger Software (OpenOCD)
  - https://github.com/riscv/riscv-openocd
- Simulator (Spike)
  - https://github.com/riscv/riscv-isa-sim
  - debug-0.13 branch
- RTL (rocket-chip)
  - https://github.com/ucb-bar/rocket-chip
- SiFive E31/E51 Coreplex

Current Focus of Debug Working Group

- Lots of eyes on the spec
  - Clarifications
  - Style edits -- want style of RISC-V Specs to match
- Focus is on ratifying what was presented today
- Working on Profiles to reduce optionality for SW writers
  - Soliciting suggestions, target implementations
- Future work will focus on additional features
  - Non-JTAG Transports
  - Trace

How to Get Involved:

- Read the spec:
  - https://github.com/riscv/riscv-debug-spec
  - https://www.sifive.com/documentation/risc-v/risc-v-external-debug-support/
- Join the mailing list:
  - https://workspace.riscv.org
  - debug@workspace.riscv.org
- Email me:
  - megan@sifive.com