

RISC-V External Debug Support  
Version 0.13  
ce1b359329ba87c3302df496885ff3781cb4c053

Tim Newsome <tim@sifive.com>

Fri Nov 17 08:26:39 2017 -0800



# Preface

**Warning! This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.**

## Acknowledgments

I would like to thank the following people for their time, feedback, and ideas: Bruce Ableidinger, Krste Asanovic, Mark Beal, Alex Bradbury, Zhong-Ho Chen, Monte Dalrymple, Vyacheslav Dyanchenco, Peter Egold, Richard Herveille, Po-wei Huang, Scott Johnson, Aram Nahidipour, Rishiyur Nikhil, Gajinder Panesar, Klaus Kruse Pedersen, Antony Pavlov, Ken Pettit, Wesley Terpstra, Megan Wachs, Stefan Wallentowitz, Ray Van De Walker, Andrew Waterman, and Andy Wright.



# Contents

|  |          |
|--|----------|
| <b>Preface</b>   | <b>i</b> |
| <b>1 Introduction</b>  | <b>1</b> |
| 1.1 Terminology . . . . .  | 1        |
| 1.1.1 Context . . . . .  | 1        |
| 1.2 About This Document . . . . .                                | 2        |
| 1.2.1 Structure . . . . .  | 2        |
| 1.2.2 Register Definition Format . . . . .                       | 2        |
| 1.2.2.1 Long Name ( <code>shortname</code> , at 0x123) . . . . . | 2        |
| 1.3 Background . . . . .   | 3        |
| 1.4 Supported Features . . . . .                                 | 3        |
| <b>2 System Overview</b>   | <b>5</b> |
| <b>3 Debug Module (DM)</b>                                       | <b>7</b> |
| 3.1 Debug Module Interface (DMI) . . . . .                       | 7        |
| 3.2 Reset Control . . . . .                                      | 8        |
| 3.3 Selecting Harts . . . . .                                    | 8        |
| 3.3.1 Selecting a Single Hart . . . . .                          | 9        |
| 3.3.2 Selecting Multiple Harts . . . . .                         | 9        |
| 3.4 Run Control . . . . .  | 9        |
| 3.5 Abstract Commands . . . . .                                  | 10       |

|         |   |    |
|---------|---|----|
| 3.5.1   | Abstract Command Listing                                      | 10 |
| 3.5.1.1 | Access Register   | 10 |
| 3.5.1.2 | Quick Access  | 12 |
| 3.6     | Program Buffer  | 12 |
| 3.7     | Overview of States  | 13 |
| 3.8     | System Bus Access   | 13 |
| 3.9     | Quick Access  | 15 |
| 3.10    | Security  | 15 |
| 3.11    | Debug Module DMI Registers                                    | 15 |
| 3.11.1  | Debug Module Status ( <b>dmstatus</b> , at 0x11)              | 15 |
| 3.11.2  | Debug Module Control ( <b>dmcontrol</b> , at 0x10)            | 18 |
| 3.11.3  | Hart Info ( <b>hartinfo</b> , at 0x12)                        | 20 |
| 3.11.4  | Halt Summary ( <b>haltsum</b> , at 0x13)                      | 21 |
| 3.11.5  | Hart Array Window Select ( <b>hawindowssel</b> , at 0x14)     | 22 |
| 3.11.6  | Hart Array Window ( <b>hawindow</b> , at 0x15)                | 22 |
| 3.11.7  | Abstract Control and Status ( <b>abstractcs</b> , at 0x16)    | 22 |
| 3.11.8  | Abstract Command ( <b>command</b> , at 0x17)                  | 23 |
| 3.11.9  | Abstract Command Autoexec ( <b>abstractauto</b> , at 0x18)    | 24 |
| 3.11.10 | Device Tree Addr 0 ( <b>devtreeaddr0</b> , at 0x19)           | 24 |
| 3.11.11 | Abstract Data 0 ( <b>data0</b> , at 0x04)                     | 25 |
| 3.11.12 | Program Buffer 0 ( <b>progbuf0</b> , at 0x20)                 | 25 |
| 3.11.13 | Authentication Data ( <b>authdata</b> , at 0x30)              | 25 |
| 3.11.14 | System Bus Access Control and Status ( <b>sbc</b> s, at 0x38) | 25 |
| 3.11.15 | System Bus Address 31:0 ( <b>sbaddress0</b> , at 0x39)        | 27 |
| 3.11.16 | System Bus Address 63:32 ( <b>sbaddress1</b> , at 0x3a)       | 27 |
| 3.11.17 | System Bus Address 95:64 ( <b>sbaddress2</b> , at 0x3b)       | 27 |
| 3.11.18 | System Bus Data 31:0 ( <b>sbdata0</b> , at 0x3c)              | 28 |
| 3.11.19 | System Bus Data 63:32 ( <b>sbdata1</b> , at 0x3d)             | 29 |

|          |   |           |
|----------|---|-----------|
| 3.11.20  | System Bus Data 95:64 ( <b>sbdata2</b> , at 0x3e)       | 29        |
| 3.11.21  | System Bus Data 127:96 ( <b>sbdata3</b> , at 0x3f)      | 29        |
| <b>4</b> | <b>RISC-V Debug</b>                                     | <b>31</b> |
| 4.1      | Debug Mode  | 31        |
| 4.2      | Load-Reserved/Store-Conditional Instructions            | 32        |
| 4.3      | Reset   | 32        |
| 4.3.1    | <b>dret</b> Instruction                                 | 32        |
| 4.4      | Core Debug Registers                                    | 32        |
| 4.4.1    | Debug Control and Status ( <b>dcsr</b> , at 0x7b0)      | 33        |
| 4.4.2    | Debug PC ( <b>dpc</b> , at 0x7b1)                       | 34        |
| 4.4.3    | Debug Scratch Register 0 ( <b>dscratch0</b> , at 0x7b2) | 35        |
| 4.4.4    | Debug Scratch Register 1 ( <b>dscratch1</b> , at 0x7b3) | 35        |
| 4.5      | Virtual Debug Registers                                 | 35        |
| 4.5.1    | Privilege Level ( <b>priv</b> , at virtual)             | 36        |
| <b>5</b> | <b>Trigger Module</b>                                   | <b>37</b> |
| 5.1      | Trigger Registers                                       | 37        |
| 5.1.1    | Trigger Select ( <b>tselect</b> , at 0x7a0)             | 38        |
| 5.1.2    | Trigger Data 1 ( <b>tdata1</b> , at 0x7a1)              | 38        |
| 5.1.3    | Trigger Data 2 ( <b>tdata2</b> , at 0x7a2)              | 39        |
| 5.1.4    | Trigger Data 3 ( <b>tdata3</b> , at 0x7a3)              | 39        |
| 5.1.5    | Match Control ( <b>mcontrol</b> , at 0x7a1)             | 39        |
| 5.1.6    | Instruction Count ( <b>icount</b> , at 0x7a1)           | 43        |
| <b>6</b> | <b>Debug Transport Module (DTM)</b>                     | <b>45</b> |
| 6.1      | JTAG Debug Transport Module                             | 45        |
| 6.1.1    | JTAG Background   | 45        |
| 6.1.2    | JTAG DTM Registers                                      | 46        |

|          |   |           |
|----------|---|-----------|
| 6.1.3    | IDCODE (at 0x01) . . . . .  | 46        |
| 6.1.4    | DTM Control and Status ( <code>dtmcs</code> , at 0x10) . . . . .      | 47        |
| 6.1.5    | Debug Module Interface Access ( <code>dmi</code> , at 0x11) . . . . . | 48        |
| 6.1.6    | BYPASS (at 0x1f) . . . . .  | 49        |
| 6.1.7    | Recommended JTAG Connector . . . . .                                  | 50        |
| <b>A</b> | <b>Hardware Implementations</b>                                       | <b>53</b> |
| A.1      | Abstract Command Based . . . . .                                      | 53        |
| A.2      | Execution Based . . . . .   | 53        |
| <b>B</b> | <b>Debugger Implementation</b>  | <b>55</b> |
| B.1      | Debug Module Interface Access . . . . .                               | 55        |
| B.2      | Main Loop . . . . .   | 56        |
| B.3      | Halting . . . . .   | 56        |
| B.4      | Running . . . . .   | 56        |
| B.5      | Single Step . . . . .   | 56        |
| B.6      | Accessing Registers . . . . .   | 56        |
| B.6.1    | Using Abstract Command . . . . .                                      | 56        |
| B.6.2    | Using Program Buffer . . . . .  | 57        |
| B.7      | Reading Memory . . . . .  | 57        |
| B.7.1    | Using System Bus Access . . . . .                                     | 57        |
| B.7.2    | Using Program Buffer . . . . .  | 58        |
| B.8      | Writing Memory . . . . .  | 59        |
| B.8.1    | Using System Bus Access . . . . .                                     | 59        |
| B.8.2    | Using Program Buffer . . . . .  | 59        |
| B.9      | Handling Exceptions . . . . .   | 60        |
| B.10     | Quick Access . . . . .  | 60        |
| <b>C</b> | <b>Future Ideas</b>   | <b>63</b> |



|       |   |           |
|-------|---|-----------|
| C.1   | Serial Ports  | 64        |
| C.1.1 | Serial Control and Status ( <code>sercs</code> , at 0x34) | 64        |
| C.1.2 | Serial TX Data ( <code>sertx</code> , at 0x35)            | 65        |
| C.1.3 | Serial RX Data ( <code>serrx</code> , at 0x36)            | 65        |
|       | <b>Index</b>  | <b>66</b> |
| D     | <b>Change Log</b>   | <b>69</b> |



# List of Figures

|     |  |    |
|-----|--|----|
| 2.1 | RISC-V Debug System Overview . . . . . | 6  |
| 3.1 | Run/Halt Debug State Machine . . . . . | 14 |



# List of Tables

|     |  |    |
|-----|--|----|
| 1.1 | Register Access Abbreviations . . . . .                | 2  |
| 3.1 | Debug Module Interface Address Space . . . . .         | 8  |
| 3.2 | Use of Data Registers . . . . .                        | 10 |
| 3.3 | Meaning of <code>cmdtype</code> . . . . .              | 10 |
| 3.4 | Abstract Register Numbers . . . . .                    | 12 |
| 3.5 | Debug Module Debug Bus Registers . . . . .             | 16 |
| 4.1 | Core Debug Registers . . . . .                         | 32 |
| 4.2 | Virtual address in DPC upon Debug Mode Entry . . . . . | 35 |
| 4.3 | Virtual Core Debug Registers . . . . .                 | 36 |
| 4.4 | Privilege Level Encoding . . . . .                     | 36 |
| 5.1 | Trigger Registers . . . . .                            | 38 |
| 5.2 | Suggested Breakpoint Timings . . . . .                 | 40 |
| 6.1 | JTAG DTM TAP Registers . . . . .                       | 46 |
| 6.2 | JTAG Connector Diagram . . . . .                       | 50 |
| 6.3 | JTAG Connector Pinout . . . . .                        | 50 |
| B.1 | Memory Read Timeline . . . . .                         | 59 |
| C.1 | Debug Module Debug Bus Registers . . . . .             | 64 |



# Chapter 1

## Introduction

When a design progresses from simulation to hardware implementation, a user’s control and understanding of the system’s current state drops dramatically. To help bring up and debug low level software and hardware, it is critical to have good debugging support built into the hardware. When a robust OS is running on a core, software can handle many debugging tasks. However, in many scenarios, hardware support is essential.

This document outlines a standard architecture for external debug support on RISC-V platforms. This architecture allows a variety of implementations and tradeoffs, which is complementary to the wide range of RISC-V implementations. At the same time, this specification defines common interfaces to allow debugging tools and components to target a variety of platforms based on the RISC-V ISA.

System designers may choose to add additional hardware debug support, but this specification defines a standard interface for common functionality.

### 1.1 Terminology

A *platform* is a single integrated circuit consisting of one or more *components*. Some components may be RISC-V cores, while others may have a different function. Typically they will all be connected to a single system bus. A single RISC-V core contains one or more hardware threads, called *harts*.

#### 1.1.1 Context

This document is written to work with:

1. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2
2. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10

## 1.2 About This Document

### 1.2.1 Structure

This document contains two parts. The main part of the document is the specification, which is given in the numbered sections. The second part of the document is a set of appendices. The information in the appendix is intended to clarify and provide examples, but is not part of the actual specification.

### 1.2.2 Register Definition Format

All register definitions in this document follow the format shown below. A simple graphic shows which fields are in the register. The upper and lower bit indices are shown to the top left and top right of each field. The total number of bits in the field are shown below it.

After the graphic follows a table which for each field lists its name, description, allowed accesses, and reset value. The allowed accesses are listed in Table [1.1](#).

Table 1.1: Register Access Abbreviations

|       |  |
|-------|--|
| R     | Read-only.   |
| R/W   | Read/Write.  |
| R/W0  | Read/Write. Only writing 0 has an effect.  |
| R/W1  | Read/Write. Only writing 1 has an effect.  |
| R/W1C | Read/Write. For each bit in the field, writing 1 clears that bit. Writing 0 has no effect. |
| W     | Write-only. When read this field returns 0.  |
| W1    | Write-only. Only writing 1 has an effect.  |

Names of registers and their bits are hyperlinks to their definition. If you're reading this on paper, there's an index with all their names on page [66](#).

#### 1.2.2.1 Long Name (shortname, at 0x123)



| Field | Description                                 | Access | Reset |
|-------|---|--------|-------|
| field | Description of what this field is used for. | R/W    | 15    |



## 1.3 Background

There are several use cases for dedicated debugging hardware, both internal to a CPU core and with an external connection. This specification addresses the use cases listed below. Implementations can choose not to implement every feature, which means some use cases might not be supported.

- Debugging low-level software in the absence of an OS or other software.
- Debugging issues in the OS itself.
- Bootstrapping a system to test, configure, and program components before there is any executable code path in the system.
- Accessing hardware on a system without a working CPU.

In addition, even without a hardware debugging interface, architectural support in a RISC-V CPU can aid software debugging and performance analysis by allowing hardware triggers and breakpoints. This specification aims to define common resources which can be used for different cases.

When debugging software, this specification distinguishes between two forms of external debugging. The first is *halt mode* debugging, where an external debugger halts some or all components of a platform and inspects their state while they are in stasis. The debugger can read and/or modify state, then direct the hardware to execute a single instruction, or continue to run freely.

The second is *run mode* debugging. In this mode a software debug agent runs on a component (eg. triggered by a timer interrupt or breakpoint on a RISC-V core) which transfers data to or from the debugger without halting the component, only briefly interrupting its program flow. This functionality is essential if the component is controlling some real-time system (like a hard drive) where long timing delays could lead to physical damage. This requires additional software support (both on the system as well as on the debugger), and efficient communication channels between the component and the debugger.

## 1.4 Supported Features

The debug interface described in this specification supports the following features:

1. RV32, RV64, and future RV128 are all supported.
2. Any hart in the platform can be independently debugged.
3. A debugger can discover almost<sup>1</sup> everything it needs to know itself, without user configuration.
4. Each hart can be debugged from the very first instruction executed.

---

<sup>1</sup>Notable exceptions include information about the memory map and peripherals.

5. A RISC-V hart can be halted when a software breakpoint instruction is executed.
6. Hardware single-step can execute one instruction at a time.
7. Debug functionality is independent of the debug transport used.
8. The debugger does not need to know anything about the microarchitecture of the cores it is debugging.
9. Arbitrary subsets of harts can be halted and resumed simultaneously. (Optional)
10. Arbitrary instructions can be executed on a halted hart. That means no new debug functionality is needed when a core has additional or custom instructions or state, as long as there exist programs that can move that state into GPRs. (Optional)
11. Registers can be accessed without halting. (Optional)
12. A running hart can be directed to execute a short sequence of instructions, with little overhead. (Optional)
13. A system bus master allows memory access without involving any hart. (Optional)
14. A RISC-V hart can be halted when a trigger matches the PC, read/write address/data, or an instruction opcode. (Optional)

While both the mechanism to execute arbitrary instructions and the system bus master are optional, at least one of them must be implemented. Otherwise there is no mechanism to access memory.

This document does not suggest a strategy or implementation for hardware test, debugging or error detection techniques. Scan, BIST, etc. are out of scope of this specification, but this specification does not intend to limit their use in RISC-V systems.

## Chapter 2

# System Overview

Figure 2.1 shows the main components of External Debug Support. Blocks shown in dotted lines are optional.

The user interacts with the Debug Host (eg. laptop), which is running a debugger (eg. gdb). The debugger communicates with a Debug Translator (eg. OpenOCD, which may include a hardware driver) to communicate with Debug Transport Hardware (eg. Olimex USB-JTAG adapter). The Debug Transport Hardware connects the Debug Host to the Platform’s Debug Transport Module (DTM). The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

The DM allows the debugger to halt any hart in the platform. Abstract commands provide access to GPRs. Additional registers are accessible through abstract commands or by writing programs to the optional Program Buffer.

The Program Buffer allows the debugger to execute arbitrary instructions on a hart. This mechanism can be used to access memory. An optional system bus access block allows memory accesses without using a RISC-V hart to perform the access.

Each RISC-V hart may implement a Trigger Module. When trigger conditions are met, harts will halt and inform the debug module that they have halted.



Figure 2.1: RISC-V Debug System Overview

## Chapter 3

# Debug Module (DM)

The Debug Module implements a translation interface between abstract debug operations and their specific implementation. It might support the following operations:

1. Give the debugger necessary information about the implementation. (Required)
2. Allow any individual hart to be halted and resumed. (Required)
3. Provide status on which harts are halted. (Required)
4. Provide read and write access to a halted hart's GPRs. (Required)
5. Provide access to a reset signal that allows debugging from the very first instruction after reset. (Required)
6. Provide access to other hart registers. (Optional)
7. Provide a Program Buffer to force the hart to execute arbitrary instructions. (Optional)
8. Allow multiple harts to be halted, resumed, and/or reset at the same time. (Optional)
9. Allow direct System Bus Access. (Optional)

In order to implement memory access, a target must implement either the Program Buffer or System Bus Access.

A single DM can debug up to 1024 harts.

### 3.1 Debug Module Interface (DMI)

The Debug Module is a slave to a bus called the Debug Module Interface (DMI). The master of the bus is the Debug Transport Module(s). The Debug Module Interface can be a trivial bus with one master and one slave, or use a more full-featured bus like TileLink or the AMBA Advanced Peripheral Bus. The details are left to the system designer.

The DMI uses between 7 and 32 address bits. It supports read and write operations. The bottom of the address space is used for the DM. Extra space can be used for custom debug devices, other cores, additional DMs, etc.

The Debug Module is controlled via register accesses to its DMI address space.

Table 3.1: Debug Module Interface Address Space

|             |  |
|-------------|--|
| 0x00 – 0x3f | Registers described in Section 3.11.   |
| 0x40 – 0x5f | This is called the ‘halt region’. These 32 addresses for 32-bit words provide access to the halt bit for up to 1024 harts. If the hart is halted, the bit is 1. Otherwise the bit is 0. The bit for hart 0 is the LSB in the 32-bit word at 0x40. The bit for hart 1023 is the MSB in the 32-bit word at 0x5f. |

## 3.2 Reset Control

The Debug Module controls a global reset signal, `ndmreset` (non-debug module reset), which can reset, or hold in reset, every component in the platform, except for the Debug Module and Debug Transport Modules. Exactly what is affected by this reset is implementation dependent, as long as it is possible to debug programs from the first instruction executed. The Debug Module’s own state and registers should only be reset at power-up and while `dmactive` in `dmcontrol` is 0. The halt state of harts should be maintained across system reset provided that `dmactive` is 1, although trigger CSRs may be cleared.

Due to clock and power domain crossing issues, it may not be possible to perform arbitrary DMI accesses across system reset. While `ndmreset` or any external reset is asserted, the only supported DM operation is accessing `dmcontrol`. The behavior of other accesses is undefined.

There is no requirement on the duration of the assertion of `ndmreset`. The implementation must ensure that a write of `ndmreset` to 1 followed by a write of `ndmreset` to 0 triggers system reset. The system may take an arbitrarily long time to come out of reset, as reported by `allunavail`, `anyunavail`, or other implementation specific indicators.

When harts have been reset, they must set a sticky `havereset` state bit. The conceptual `havereset` state bits can be read for selected harts in `anyhavereset` and `allhavereset` in `dmstatus`. These bits must be set regardless of the cause of the reset. The `havereset` bits for the selected harts can be cleared by writing 1 to `ackhavereset` in `dmcontrol`. The `havereset` bits may or may not be cleared when `dmactive` is low.

## 3.3 Selecting Harts

Up to 1024 harts can be connected to a single DM. The debugger selects a hart, and then subsequent halt, resume, reset, and debugging commands are specific to that hart.

A debugger can enumerate all the harts attached to the DM by selecting each hart starting from 0 until `anynonexistent` in `dmstatus` is 1.

The debugger can discover the mapping between hart indices and `mhartid` by using the interface to read `mhartid`, or by reading the system's Device Tree.

### 3.3.1 Selecting a Single Hart

All debug modules must support selecting a single hart. The debugger can select a hart by writing its index to `hartsel`. Hart indexes start at 0 and are contiguous until the final index.

### 3.3.2 Selecting Multiple Harts

Debug Modules may optionally implement a Hart Array Mask register to allow selecting multiple harts at once. The debugger can set bits in the hart array mask register using `hawindowssel` and `hawindow`, then apply actions to all selected harts by setting `hasel`. If this feature is supported, multiple harts can be halted, resumed, and reset simultaneously.

Only the actions initiated by `dmcontrol` can apply to multiple harts at once, Abstract Commands apply only to the hart selected by `hartsel`.

## 3.4 Run Control

For every hart, the Debug Module contains 3 conceptual bits of state: halt request, resume request, and hart reset. (The hart reset bit is optional.) These bits all reset to 0. A debugger can write them for the currently selected harts through `haltreq`, `resumereq`, and `hartreset` in `dmcontrol`. In addition the DM receives halted, running, and resume ack signals from each hart.

When a running hart receives a halt request, it responds by halting and asserting its halted signal. The halted signals of all selected harts are reflected in the `allhalted` and `anyhalted` bits. `haltreq` is ignored by halted harts.

When a halted hart receives a resume request, it responds by resuming, clearing its halted signal, and asserting its running signal and resume ack signals. The resume ack signal is lowered when the resume request is deasserted. These status signals of all selected harts are reflected in `allresumeack`, `anyresumeack`, `allrunning`, and `anyrunning`. `resumereq` is ignored by running harts.

When halt or resume is requested, a hart must respond in less than one second, unless it is unavailable. (How this is implemented is not further specified. A few clock cycles will be a more typical latency).

## 3.5 Abstract Commands

The DM supports a set of abstract commands, most of which are optional. Depending on the implementation, the debugger may be able to perform some abstract commands even when the selected hart is not halted. Debuggers can only determine which abstract commands are supported by a given hart in a given state by attempting them and then looking at `cmderr` in `abstractcs` to see if they were successful.

Debuggers execute abstract commands by writing them to `command`. Debuggers can determine whether an abstract command is complete by reading `busy` in `abstractcs`. If the command takes arguments, the debugger must write them to the `data` registers before writing to `command`. If a command returns results, the Debug Module must ensure they are placed in the `data` registers before `busy` is cleared. Which `data` registers are used for the arguments is described in Table 3.2. In all cases the least-significant word is placed in the lowest-numbered `data` register.

Table 3.2: Use of Data Registers

| XLEN | arg0/return value                       | arg1                                    | arg2                                     |
|------|---|---|--|
| 32   | <code>data0</code>                      | <code>data1</code>                      | <code>data2</code>                       |
| 64   | <code>data0</code> , <code>data1</code> | <code>data2</code> , <code>data3</code> | <code>data4</code> , <code>data5</code>  |
| 128  | <code>data0</code> – <code>data3</code> | <code>data4</code> – <code>data7</code> | <code>data8</code> – <code>data11</code> |

### 3.5.1 Abstract Command Listing

This section describes each of the different abstract commands and how their fields should be interpreted when they are written to `command`.

Each abstract command is a 32-bit value. The top 8 bits contain `cmdtype` which determines the kind of command. Table 3.3 lists all commands.

Table 3.3: Meaning of `cmdtype`

| <code>cmdtype</code> | Command                 | Page |
|----------------------|-------------------------|------|
| 0                    | Access Register Command | 10   |
| 1                    | Quick Access            | 12   |

#### 3.5.1.1 Access Register

This command gives the debugger access to CPU registers and program buffer. It performs the following sequence of operations:

1. Copy data from the register specified by `regno` into the `arg0` region of `data`, if `write` is clear and `transfer` is set.
2. Copy data from the `arg0` region of `data` into the register specified by `regno`, if `write` is set and `transfer` is set.



3. Execute the Program Buffer, if `postexec` is set.

If any of these operations fail, `cmderr` is set and none of the remaining steps are executed. An implementation may detect an upcoming failure early, and fail the overall command before it reaches the step that would cause failure.

Debug Modules must implement this command and must support read and write access to all GPRs when the selected hart is halted. Debug Modules may optionally support accessing other registers, or accessing registers when the hart is running. If this command is supported for a register while the hart is running, it must also be supported for a register while the hart is halted. Each individual register (aside from GPRs) may be supported differently across read, write, and halt status.

---

*The encoding of `size` was chosen to match `sbaccess` in `sbc`.*

|         |    |      |    |          |          |       |       |    |    |   |
|---------|----|------|----|----------|----------|-------|-------|----|----|---|
| 31      | 24 | 23   | 22 | 20       | 19       | 18    | 17    | 16 | 15 | 0 |
| cmdtype | 0  | size | 0  | postexec | transfer | write | regno |    |    |   |
| 8       | 1  | 3    | 1  | 1        | 1        | 1     | 16    |    |    |   |

| Field                  | Description   |
|------------------------|---|
| <code>cmdtype</code>   | This is 0 to indicate Access Register Command.  |
| <code>size</code>      | 2: Access the lowest 32 bits of the register.<br>3: Access the lowest 64 bits of the register.<br>4: Access the lowest 128 bits of the register.<br>If <code>size</code> specifies a size larger than the register's actual size, then the access must fail. If a register is accessible, then reads of <code>size</code> less than or equal to the register's actual size must be supported. |
| <code>postexec</code>  | When 1, execute the program in the Program Buffer exactly once after performing the transfer, if any.   |
| <code>transfer</code>  | 0: Don't do the operation specified by <code>write</code> .<br>1: Do the operation specified by <code>write</code> .<br>This bit can be used to just execute the Program Buffer without having to worry about placing valid values into <code>size</code> or <code>regno</code> .   |
| <code>write</code>     | When <code>transfer</code> is set: 0: Copy data from the specified register into <code>arg0</code> portion of <code>data</code> .<br>1: Copy data from <code>arg0</code> portion of <code>data</code> into the specified register.  |
| Continued on next page |   |

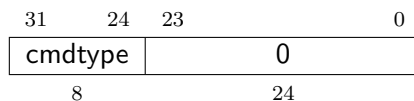
|       |   |
|-------|---|
| regno | Number of the register to access, as described in Table 3.4. <code>dpc</code> may be used as an alias for PC if this command is supported on a non-halted hart. |
|-------|---|

### 3.5.1.2 Quick Access

Perform the following sequence of operations:

1. If the hart is halted, the command sets `cmderr` to `halt/resume` and does not continue.
2. Halt the hart. If the hart halts for some other reason (e.g. breakpoint), the command sets `cmderr` to `halt/resume` and does not continue.
3. Execute the Program Buffer. If an exception occurs, `cmderr` is set to `exception` and the program buffer execution ends, but the quick access command continues.
4. Resume the hart.

Implementing this command is optional.



| Field   | Description                                 |
|---------|---|
| cmdtype | This is 1 to indicate Quick Access command. |

Table 3.4: Abstract Register Numbers

|                 |  |
|-----------------|--|
| 0x0000 – 0x0fff | CSRs. The “PC” can be accessed here through <code>dpc</code> . |
| 0x1000 – 0x101f | GPRs   |
| 0x1020 – 0x103f | Floating point registers                                       |
| 0xc000 – 0xffff | Reserved for non-standard extensions and internal use.         |

## 3.6 Program Buffer

To support executing arbitrary instructions on a halted hart, a Debug Module can include a Program Buffer that a debugger can write small programs to. Systems that support all necessary functionality using abstract commands only may choose to omit the Program Buffer.

A debugger can write a small program to the Program Buffer, and then execute it exactly once with the Access Register Abstract Command, setting the `postexec` bit in `command`. The debugger can write whatever program it likes (including jumps out of the Program Buffer), but the program

must end with `ebreak` or `ebreak.c`. To save hardware, an implementation may support an implied `ebreak` that is executed when a hart runs off the end of the Program Buffer. This is indicated in `impebreak`. With this feature, a Program Buffer of just 2 32-bit words can offer efficient debugging.

If `progbufsize` is 1, the Program Buffer may only hold a single instruction, and `impebreak` must be 1. This instruction can be a 32-bit instruction, or a compressed instruction in the lower 16 bits accompanied by a compressed `nop` in the upper 16 bits.

If the debugger executes a program that does not terminate with an `ebreak` instruction, the hart will remain in Debug Mode until it is reset.

While these programs are executed, the hart does not leave Debug Mode (see Section 4.1). If an exception is encountered during execution of the Program Buffer, no more instructions are executed, the hart remains in Debug Mode, and `cmderr` is set to 3 (`exception error`). If the debugger executes a program that doesn't terminate, then it loses control of the hart.

Executing the Program Buffer may clobber `dpc`. If that is the case, it must be possible to read/write `dpc` using an abstract command with `postexec` not set. The debugger must attempt to save `dpc` between halting and executing a Program Buffer, and then restore `dpc` before leaving Debug Mode.

---

*Allowing Program Buffer execution to clobber `dpc` allows for direct implementations that don't have a separate PC register, and do need to use the PC when executing the Program Buffer.*

---

The Program Buffer may be implemented as RAM which is accessible to the hart as RAM memory. A debugger can determine if this is the case by executing small programs that attempt to write and read back relative to `pc` while executing from the Program Buffer. If so, the debugger has more flexibility in what it can do with the program buffer.

### 3.7 Overview of States

Figure 3.1 shows a conceptual view of the states passed through by a hart during run/halt debugging as influenced by the different fields of `dmcontrol`, `abstractcs`, `abstractauto`, and `command`.

### 3.8 System Bus Access

When a Program Buffer is present, a debugger can access the system bus by having a RISC-V hart perform the accesses it requires. A Debug Module may also include a System Bus Access block to provide memory access without involving a hart, regardless of whether Program Buffer is implemented. The System Bus Access block uses physical addresses.

Depending on the microarchitecture, data accessed through System Bus Access may not always be coherent with that observed by each hart. (For instance, a hart may have caches that don't snoop or aren't write-through.) It is up to the debugger to enforce coherency if the implementation does not. This specification does not define a standard way to do this, as it is implementation/platform specific. Possibilities may include using the System Bus Interface and/or Program Buffer to write to special memory-mapped locations, or executing special instructions via the Program Buffer.



Figure 3.1: Run/Halt Debug State Machine. As only a small amount of state is visible to the debugger, the states and transitions are conceptual.

---

*Implementing a System Bus Access block has several benefits even when a Debug Module also implements a Program Buffer. First, it is possible to access memory in a running system with minimal impact. Second, it may improve performance when accessing memory. Third, it may provide access to devices that a hart does not have access to.*

## 3.9 Quick Access

Depending on the task it is performing, some harts can only be halted very briefly. There are several mechanisms that allow accessing resources in such a running system with a minimal impact on the running hart.

First, an implementation may allow some abstract commands to execute without halting the hart.

Second, the Quick Access abstract command can be used to halt a hart, quickly execute the contents of the Program Buffer, and let the hart run again. Combined with instructions that allow Program Buffer code to access the `data` registers, as described in 3.11.3, this can be used to quickly perform a memory or register access. For some systems this will be too intrusive, but many systems that can't be halted can bear an occasional hiccup of a hundred or less cycles.

Third, if the System Bus Access block is implemented, it can be used while a hart is running to access system memory.

## 3.10 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. A few bits in `dmstatus` and `authdata` can support an arbitrarily complex authentication mechanism. When `authenticated` is clear, the DM must not interact with the rest of the platform in any way.

## 3.11 Debug Module DMI Registers

When read, unimplemented Debug Module DMI Registers return 0. Writing them has no effect.

### 3.11.1 Debug Module Status (`dmstatus`, at 0x11)

The address of this register will not change in the future, because it contains `version`. It has changed from version 0.11 of this spec.

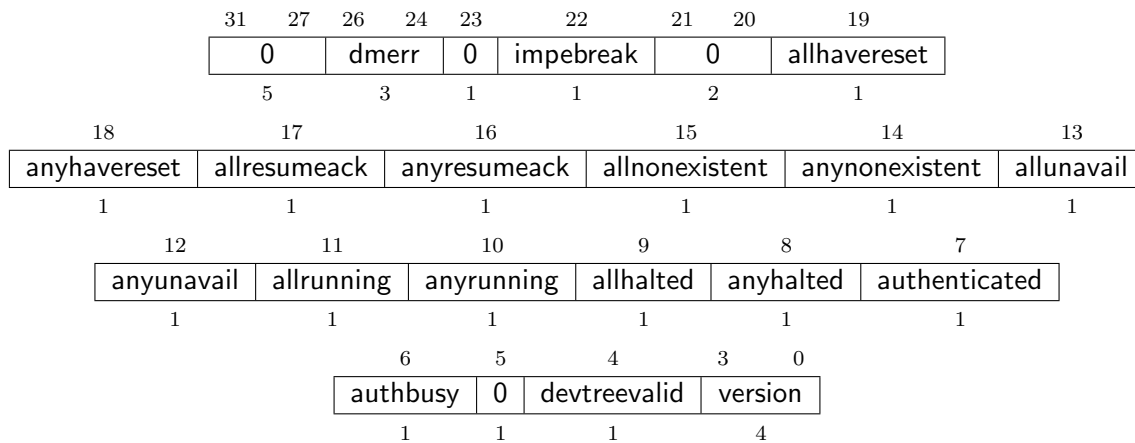
Table 3.5: Debug Module Debug Bus Registers

| Address | Name                                 | Page               |
|---------|--------------------------------------|--------------------|
| 0x04    | Abstract Data 0                      | <a href="#">25</a> |
| 0x0f    | Abstract Data 11                     |                    |
| 0x10    | Debug Module Control                 | <a href="#">18</a> |
| 0x11    | Debug Module Status                  | <a href="#">15</a> |
| 0x12    | Hart Info                            | <a href="#">20</a> |
| 0x13    | Halt Summary                         | <a href="#">21</a> |
| 0x14    | Hart Array Window Select             | <a href="#">22</a> |
| 0x15    | Hart Array Window                    | <a href="#">22</a> |
| 0x16    | Abstract Control and Status          | <a href="#">22</a> |
| 0x17    | Abstract Command                     | <a href="#">23</a> |
| 0x18    | Abstract Command Autoexec            | <a href="#">24</a> |
| 0x19    | Device Tree Addr 0                   | <a href="#">24</a> |
| 0x1a    | Device Tree Addr 1                   |                    |
| 0x1b    | Device Tree Addr 2                   |                    |
| 0x1c    | Device Tree Addr 3                   |                    |
| 0x20    | Program Buffer 0                     | <a href="#">25</a> |
| 0x2f    | Program Buffer 15                    |                    |
| 0x30    | Authentication Data                  | <a href="#">25</a> |
| 0x38    | System Bus Access Control and Status | <a href="#">25</a> |
| 0x39    | System Bus Address 31:0              | <a href="#">27</a> |
| 0x3a    | System Bus Address 63:32             | <a href="#">27</a> |
| 0x3b    | System Bus Address 95:64             | <a href="#">27</a> |
| 0x3c    | System Bus Data 31:0                 | <a href="#">28</a> |
| 0x3d    | System Bus Data 63:32                | <a href="#">29</a> |
| 0x3e    | System Bus Data 95:64                | <a href="#">29</a> |
| 0x3f    | System Bus Data 127:96               | <a href="#">29</a> |

This register reports status for the overall debug module as well as the currently selected harts, as defined in [hasel](#).

Harts are nonexistent if they will never be part of this system, no matter how long a user waits. Eg. in a simple single-hart system only one hart exists, and all others are nonexistent.

Harts are unavailable if they might exist/become available at a later time. Eg. in a multi-hart system some might temporarily be powered down, or a system might support hot-swapping harts.



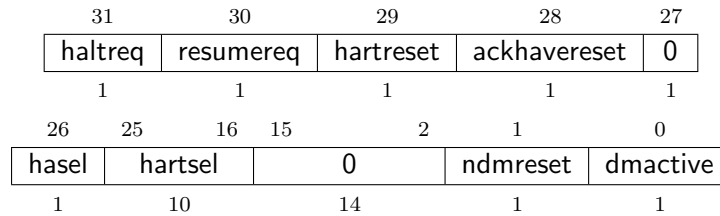
| Field                  | Description   | Access | Reset  |
|------------------------|---|--------|--------|
| dmerr                  | Gets set if the Debug Module was accessed incorrectly.<br>0 (none): No error.<br>1 (badaddr): There was an access to an unimplemented Debug Module address.<br>7 (other): An access failed for another reason.  | R/W1C  | 0      |
| impebreak              | If 1, then there is an implicit <b>ebreak</b> instruction at the non-existent word immediately after the Program Buffer. This saves the debugger from having to write the <b>ebreak</b> itself, and allows the Program Buffer to be one word smaller. This must be 1 when <a href="#">progbufsize</a> is 1. | R      | Preset |
| allhavereset           | This field is 1 when all currently selected harts have been reset but the reset has not been acknowledged.  | R      | -      |
| anyhavereset           | This field is 1 when any currently selected hart has been reset but the reset has not been acknowledged.  | R      | -      |
| Continued on next page |   |        |        |

|                |   |   |        |
|----------------|---|---|--------|
| allresumeack   | This field is 1 when all currently selected harts have acknowledged the previous resume request.  | R | -      |
| anyresumeack   | This field is 1 when any currently selected hart has acknowledged the previous resume request.  | R | -      |
| allnonexistent | This field is 1 when all currently selected harts do not exist in this system.  | R | -      |
| anynonexistent | This field is 1 when any currently selected hart does not exist in this system.   | R | -      |
| allunavail     | This field is 1 when all currently selected harts are unavailable.  | R | -      |
| anyunavail     | This field is 1 when any currently selected hart is unavailable.  | R | -      |
| allrunning     | This field is 1 when all currently selected harts are running.  | R | -      |
| anyrunning     | This field is 1 when any currently selected hart is running.  | R | -      |
| allhalted      | This field is 1 when all currently selected harts are halted.   | R | -      |
| anyhalted      | This field is 1 when any currently selected hart is halted.   | R | -      |
| authenticated  | 0 when authentication is required before using the DM. 1 when the authentication check has passed. On components that don't implement authentication, this bit must be preset as 1.   | R | Preset |
| authbusy       | 0: The authentication module is ready to process the next read/write to <a href="#">authdata</a> .<br>1: The authentication module is busy. Accessing <a href="#">authdata</a> results in unspecified behavior.<br><a href="#">authbusy</a> only becomes set in immediate response to an access to <a href="#">authdata</a> . | R | 0      |
| devtreevalid   | 0: <a href="#">devtreeaddr0</a> – <a href="#">devtreeaddr3</a> hold information which is not relevant to the Device Tree.<br>1: <a href="#">devtreeaddr0</a> – <a href="#">devtreeaddr3</a> registers hold the address of the Device Tree.  | R | Preset |
| version        | 0: There is no Debug Module present.<br>1: There is a Debug Module and it conforms to version 0.11 of this specification.<br>2: There is a Debug Module and it conforms to version 0.13 of this specification.<br>15: There is a Debug Module but it does not conform to any available version of this spec.                  | R | 2      |

### 3.11.2 Debug Module Control ([dmcontrol](#), at 0x10)

This register controls the overall debug module as well as the currently selected harts, as defined in [hasel](#).





| Field        | Description   | Access | Reset |
|--------------|---|--------|-------|
| haltreq      | Writes the halt request bit for all currently selected harts. When set to 1, each selected hart will halt if it is not currently halted.<br>Writing 1 or 0 has no effect on a hart which is already halted, but the bit must be cleared to 0 before the hart is resumed.<br>Writes apply to the new value of <a href="#">hartsel</a> and <a href="#">hasel</a> .  | W      | -     |
| resumereq    | Writes the resume request bit for all currently selected harts. When set to 1, each selected hart will resume if it is currently halted.<br>The resume request bit is ignored while the halt request bit is set.<br>Writes apply to the new value of <a href="#">hartsel</a> and <a href="#">hasel</a> .  | W      | -     |
| hartreset    | This optional field writes the reset bit for all the currently selected harts. To perform a reset the debugger writes 1, and then writes 0 to deassert the reset signal.<br>If this feature is not implemented, the bit always stays 0, so after writing 1 the debugger can read the register back to see if the feature is supported.<br>Writes apply to the new value of <a href="#">hartsel</a> and <a href="#">hasel</a> .  | W      | -     |
| ackhavereset | Writing 1 to this bit clears the <b>havereset</b> bits for any selected harts.<br>Writes apply to the new value of <a href="#">hartsel</a> and <a href="#">hasel</a> .  | W      | -     |
| hasel        | Selects the definition of currently selected harts.<br>0: There is a single currently selected hart, that selected by <a href="#">hartsel</a> .<br>1: There may be multiple currently selected harts – that selected by <a href="#">hartsel</a> , plus those selected by the hart array mask register.<br>An implementation which does not implement the hart array mask register should tie this field to 0. A debugger which wishes to use the hart array mask register feature should set this bit and read back to see if the functionality is supported. | R/W    | 0     |

Continued on next page

|                 |  |     |   |
|-----------------|--|-----|---|
| <b>hartsel</b>  | The DM-specific index of the hart to select. This hart is always part of the currently selected harts.   | R/W | 0 |
| <b>ndmreset</b> | This bit controls the reset signal from the DM to the rest of the system. The signal should reset every part of the system, including every hart, except for the DM and any logic required to access the DM. To perform a system reset the debugger writes 1, and then writes 0 to deassert the reset.   | R/W | 0 |
| <b>dmactive</b> | <p>This bit serves as a reset signal for the Debug Module itself.</p> <p>0: The module's state, including authentication mechanism, takes its reset values (the <b>dmactive</b> bit is the only bit which can be written to something other than its reset value).</p> <p>1: The module functions normally.</p> <p>No other mechanism should exist that may result in resetting the Debug Module after power up, including the platform's system reset or Debug Transport reset signals.</p> <p>A debugger may pulse this bit low to get the debug module into a known state.</p> <p>Implementations may use this bit to aid debugging, for example by preventing the Debug Module from being power gated while debugging is active.</p> | R/W | 0 |

### 3.11.3 Hart Info (hartinfo, at 0x12)

This register gives information about the hart currently selected by **hartsel**.

This register is optional. If it is not present it should read all-zero.

If this register is included, the debugger can do more with the Program Buffer by writing programs which explicitly access the **data** and/or **dscratch** registers.

This entire register is read-only.

|    |          |    |    |    |            |    |          |    |          |   |
|----|----------|----|----|----|------------|----|----------|----|----------|---|
| 31 | 24       | 23 | 20 | 19 | 17         | 16 | 15       | 12 | 11       | 0 |
| 0  | nscratch |    |    | 0  | dataaccess |    | datasize |    | dataaddr |   |
| 8  | 4        |    |    | 3  | 1          |    | 4        |    | 12       |   |

| Field      | Description   | Access | Reset  |
|------------|---|--------|--------|
| nscratch   | Number of <code>dscratch</code> registers available for the debugger to use during program buffer execution, starting from <code>dscratch0</code> . The debugger can make no assumptions about the contents of these registers between commands.  | R      | Preset |
| dataaccess | 0: The <code>data</code> registers are shadowed in the hart by CSR registers. Each CSR register is XLEN bits in size, and corresponds to a single argument, per Table 3.2.<br>1: The <code>data</code> registers are shadowed in the hart's memory map. Each register takes up 4 bytes in the memory map. | R      | Preset |
| datasize   | If <code>dataaccess</code> is 0: Number of CSR registers dedicated to shadowing the <code>data</code> registers.<br>If <code>dataaccess</code> is 1: Number of 32-bit words in the memory map dedicated to shadowing the <code>data</code> registers.   | R      | Preset |
| dataaddr   | If <code>dataaccess</code> is 0: The number of the first CSR dedicated to shadowing the <code>data</code> registers.<br>If <code>dataaccess</code> is 1: Signed address of RAM where the <code>data</code> registers are shadowed, to be used to access relative to <code>zero</code> .                   | R      | Preset |

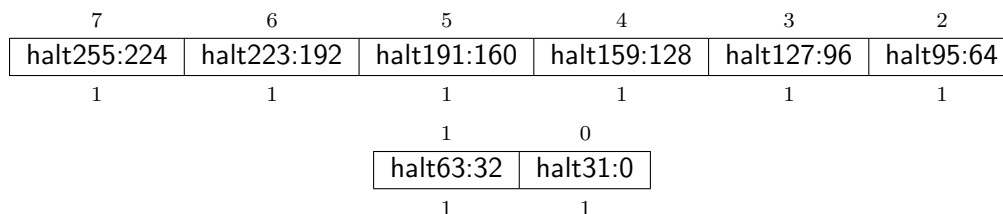
### 3.11.4 Halt Summary (`haltsum`, at 0x13)

This register contains a summary of which harts are halted.

Each bit contains the logical OR of 32 halt bits. When there are a large number of harts in the system, the debugger can first read this register, and then read from the halt region (0x40–0x5f) to determine which hart is the one that is halted.

This entire register is read-only.

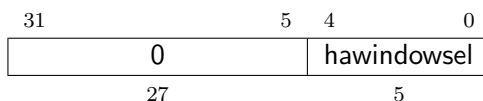
|              |             |             |             |             |             |
|--------------|-------------|-------------|-------------|-------------|-------------|
| 31           | 30          | 29          | 28          | 27          | 26          |
| halt1023:992 | halt991:960 | halt959:928 | halt927:896 | halt895:864 | halt863:832 |
| 1            | 1           | 1           | 1           | 1           | 1           |
| 25           | 24          | 23          | 22          | 21          | 20          |
| halt831:800  | halt799:768 | halt767:736 | halt735:704 | halt703:672 | halt671:640 |
| 1            | 1           | 1           | 1           | 1           | 1           |
| 19           | 18          | 17          | 16          | 15          | 14          |
| halt639:608  | halt607:576 | halt575:544 | halt543:512 | halt511:480 | halt479:448 |
| 1            | 1           | 1           | 1           | 1           | 1           |
| 13           | 12          | 11          | 10          | 9           | 8           |
| halt447:416  | halt415:384 | halt383:352 | halt351:320 | halt319:288 | halt287:256 |
| 1            | 1           | 1           | 1           | 1           | 1           |



### 3.11.5 Hart Array Window Select (hawindowssel, at 0x14)

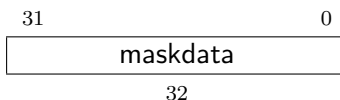
This register selects which of the 32-bit portion of the hart array mask register is accessible in [hawindow](#).

The hart array mask register provides a mask of all harts controlled by the debug module. A hart is part of the currently selected harts if the corresponding bit is set in the hart array mask register and [hasel](#) in [dmcontrol](#) is 1, or if the hart is selected by [hartsel](#).

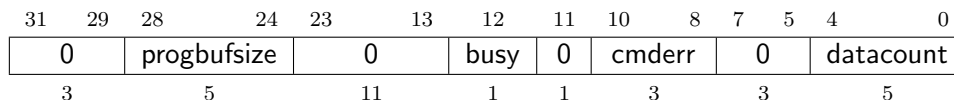


### 3.11.6 Hart Array Window (hawindow, at 0x15)

This register provides R/W access to a 32-bit portion of the hart array mask register. The position of the window is determined by [hawindowssel](#). I.e. bit 0 refers to hart  $\text{hawindowssel} * 32$ , while bit 31 refers to hart  $\text{hawindowssel} * 32 + 31$ .



### 3.11.7 Abstract Control and Status (abstractcs, at 0x16)



| Field       | Description  | Access | Reset  |
|-------------|--|--------|--------|
| progbuFSIZE | Size of the Program Buffer, in 32-bit words. Valid sizes are 0 - 16. | R      | Preset |

Continued on next page

|                        |   |       |   |
|------------------------|---|-------|---|
| busy                   | 1: An abstract command is currently being executed.<br>This bit is set as soon as <code>command</code> is written, and is not cleared until that command has completed.   | R     | 0 |
| cmderr                 | Gets set if an abstract command fails. The bits in this field remain set until they are cleared by writing 1 to them. No abstract command is started until the value is reset to 0.<br>0 (none): No error.<br>1 (busy): An abstract command was executing while <code>command</code> , <code>abstractcs</code> , <code>abstractauto</code> was written, or when one of the <code>data</code> or <code>progbuf</code> registers was read or written.<br>2 (not supported): The requested command is not supported. A command that is not supported while the hart is running may be supported when it is halted.<br>3 (exception): An exception occurred while executing the command (eg. while executing the Program Buffer).<br>4 (halt/resume): An abstract command couldn't execute because the hart wasn't in the expected state (running/halted).<br>7 (other): The command failed for another reason. | R/W1C | 0 |
| Continued on next page |   |       |   |

|           |   |   |        |
|-----------|---|---|--------|
| datacount | Number of <b>data</b> registers that are implemented as part of the abstract command interface. Valid sizes are 0 - 12. | R | Preset |
|-----------|---|---|--------|

### 3.11.8 Abstract Command (command, at 0x17)

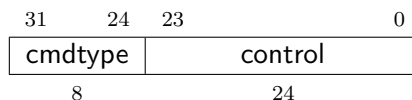
Writes to this register cause the corresponding abstract command to be executed.

Writing while an abstract command is executing causes **cmderr** to be set.

If **cmderr** is non-zero, writes to this register are ignored.

---

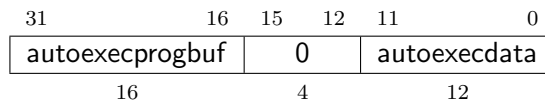
**cmderr** inhibits starting a new command to accommodate debuggers that, for performance reasons, send several commands to be executed in a row without checking **cmderr** in between. They can safely do so and check **cmderr** at the end without worrying that one command failed but then a later command (which might have depended on the previous one succeeding) passed.



| Field   | Description  | Access | Reset |
|---------|--|--------|-------|
| cmdtype | The type determines the overall functionality of this abstract command.                      | W      | 0     |
| control | This field is interpreted in a command-specific manner, described for each abstract command. | W      | 0     |

### 3.11.9 Abstract Command Autoexec (abstractauto, at 0x18)

This register is optional. Including it allows more efficient burst accesses. Debugger can attempt to set bits and read them back to determine if the functionality is supported.



| Field           | Description   | Access | Reset |
|-----------------|---|--------|-------|
| autoexecprogbuf | When a bit in this field is 1, read or write accesses the corresponding <b>progbuf</b> word cause the command in <b>command</b> to be executed again. | R/W    | 0     |
| autoexecdata    | When a bit in this field is 1, read or write accesses the corresponding <b>data</b> word cause the command in <b>command</b> to be executed again.    | R/W    | 0     |

### 3.11.10 Device Tree Addr 0 (devtreeaddr0, at 0x19)

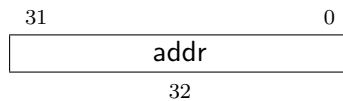
When `devtreevalid` is set, reading this register returns bits 31:0 of the Device Tree address. Reading the other `devtreeaddr` registers returns the upper bits of the address.

When system bus mastering is implemented, this must be an address that can be used with the System Bus Access module. Otherwise, this must be an address that can be used to access the Device Tree from the hart with ID 0.

If `devtreevalid` is 0, then the `devtreeaddr` registers hold identifier information which is not further specified in this document.

The Device Tree itself is described in the RISC-V Privileged Specification.

This entire register is read-only.



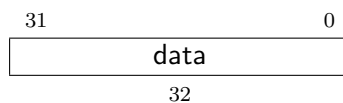
### 3.11.11 Abstract Data 0 (data0, at 0x04)

Basic read/write registers that may be read or changed by abstract commands.

Accessing them while an abstract command is executing causes `cmderr` to be set.

Attempts to write them while `busy` is set does not change their value.

The values in these registers may not be preserved after an abstract command is executed. The only guarantees on their contents are the ones offered by the command in question. If the command fails, no assumptions can be made about the contents of these registers.

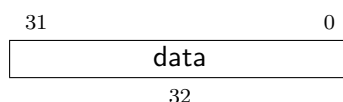


### 3.11.12 Program Buffer 0 (progbuf0, at 0x20)

The `progbuf` registers provide read/write access to the optional program buffer.

Accessing them while an abstract command is executing causes `cmderr` to be set.

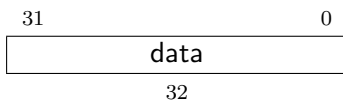
Attempts to write them while `busy` is set does not change their value.



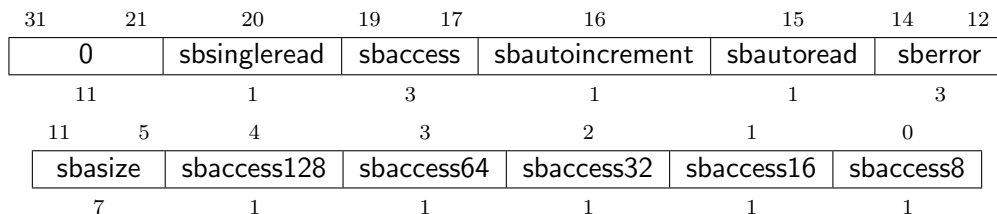
### 3.11.13 Authentication Data (authdata, at 0x30)

This register serves as a 32-bit serial port to the authentication module.

When [authbusy](#) is clear, the debugger can communicate with the authentication module by reading or writing this register. There is no separate mechanism to signal overflow/underflow.



### 3.11.14 System Bus Access Control and Status (sbcs, at 0x38)



| Field                  | Description   | Access | Reset |
|------------------------|---|--------|-------|
| sbsingleread           | When a 1 is written here, triggers a read at the address in <b>sbaddress</b> using the access size set by <a href="#">sbaccess</a> .  | W1     | 0     |
| sbaccess               | Select the access size to use for system bus accesses triggered by writes to the <b>sbaddress</b> registers or <a href="#">sbddata0</a> .<br>0: 8-bit<br>1: 16-bit<br>2: 32-bit<br>3: 64-bit<br>4: 128-bit<br>If an unsupported system bus access size is written here, the DM does not perform the access and sberror is set to 3. | R/W    | 2     |
| Continued on next page |   |        |       |



|                        |   |       |        |
|------------------------|---|-------|--------|
| <b>sbautoincrement</b> | When 1, <b>sbaddress</b> is incremented by the access size (in bytes) selected in <b>sbaccess</b> after every system bus access.  | R/W   | 0      |
| <b>sbautoread</b>      | When 1, every read from <b>sbddata0</b> automatically triggers a system bus read at the (possibly auto-incremented) address.  | R/W   | 0      |
| <b>sberror</b>         | When the debug module's system bus master causes a bus error, this field gets set. The bits in this field remain set until they are cleared by writing 1 to them. While this field is non-zero, no more system bus accesses can be initiated by the debug module.<br>0: There was no bus error.<br>1: There was a timeout.<br>2: A bad address was accessed.<br>3: There was some other error (eg. alignment).<br>4: The system bus master was busy when one of the <b>sbaddress</b> or <b>sbddata</b> registers was written, or <b>sbddata0</b> was read when it had stale data. | R/W1C | 0      |
| <b>sbsize</b>          | Width of system bus addresses in bits. (0 indicates there is no bus access support.)  | R     | Preset |
| <b>sbaccess128</b>     | 1 when 128-bit system bus accesses are supported.   | R     | Preset |
| <b>sbaccess64</b>      | 1 when 64-bit system bus accesses are supported.  | R     | Preset |
| <b>sbaccess32</b>      | 1 when 32-bit system bus accesses are supported.  | R     | Preset |
| <b>sbaccess16</b>      | 1 when 16-bit system bus accesses are supported.  | R     | Preset |
| <b>sbaccess8</b>       | 1 when 8-bit system bus accesses are supported.   | R     | Preset |

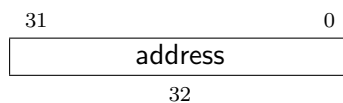
### 3.11.15 System Bus Address 31:0 (**sbaddress0**, at 0x39)

If **sbsize** is 0, then this register is not present.

When the system bus master is busy, writes to this register will set **sberror**.

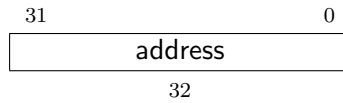
If **sberror** is 0 and **sbautoread** is set then the system bus master will start to read after updating the address from **address**. The access size is controlled by **sbaccess** in **sbc**.

If **sbsingleread** is set, the bit is cleared.



| Field          | Description  | Access | Reset |
|----------------|--|--------|-------|
| <b>address</b> | Accesses bits 31:0 of the physical address in <b>sbaddress</b> . | R/W    | 0     |

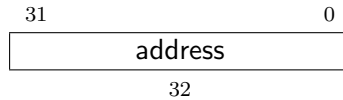
### 3.11.16 System Bus Address 63:32 (**sbaddress1**, at 0x3a)



| Field          | Description   | Access | Reset |
|----------------|---|--------|-------|
| <b>address</b> | Accesses bits 63:32 of the physical address in <b>sbaddress</b> (if the system address bus is that wide). | R/W    | 0     |

### 3.11.17 System Bus Address 95:64 (**sbaddress2**, at 0x3b)

If **sbasize** is less than 65, then this register is not present.



| Field          | Description   | Access | Reset |
|----------------|---|--------|-------|
| <b>address</b> | Accesses bits 95:64 of the physical address in <b>sbaddress</b> (if the system address bus is that wide). | R/W    | 0     |

### 3.11.18 System Bus Data 31:0 (**sbdata0**, at 0x3c)

If all of the **sbaccess** bits in **sbc** are 0, then this register is not present.

Any successful system bus read updates the data in this register, and marks it no longer stale.

If **sberror** isn't 0 then accesses do nothing.

Writes to this register:

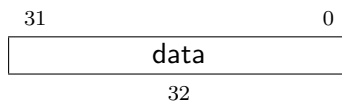
1. If the bus master is busy then accesses set **sberror**, and don't do anything else.
2. Start a bus write of **sbdata** to **sbaddress**.
3. If **sbautoincrement** is set, increment **sbaddress**.

Reads from this register:

1. If the register is marked stale, then set **sberror** and don't do anything else.

2. “Return” the data.
3. Mark the register stale.
4. If `sbautoincrement` is set, increment `sbaddress`.
5. If `sbautoread` is set, start another system bus read.

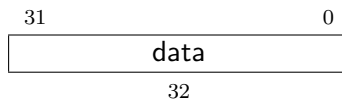
Only `sbddata0` has this behavior. The other `sbddata` registers have no side effects. On systems that have buses wider than 32 bits, a debugger should access `sbddata0` after accessing the other `sbddata` registers.



| Field       | Description                                  | Access | Reset |
|-------------|--|--------|-------|
| <b>data</b> | Accesses bits 31:0 of <code>sbddata</code> . | R/W    | 0     |

### 3.11.19 System Bus Data 63:32 (`sbddata1`, at 0x3d)

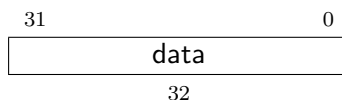
If `sbaccess64` and `sbaccess128` are 0, then this register is not present.



| Field       | Description   | Access | Reset |
|-------------|---|--------|-------|
| <b>data</b> | Accesses bits 63:32 of <code>sbddata</code> (if the system bus is that wide). | R/W    | 0     |

### 3.11.20 System Bus Data 95:64 (`sbddata2`, at 0x3e)

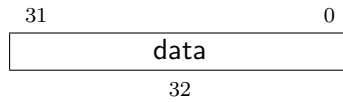
This register only exists if `sbaccess128` is 1.



| Field       | Description  | Access | Reset |
|-------------|--|--------|-------|
| <b>data</b> | Accesses bits 95:64 of <b>sbdata</b> (if the system bus is that wide). | R/W    | 0     |

### 3.11.21 System Bus Data 127:96 (sbdata3, at 0x3f)

This register only exists if [sbaccess128](#) is 1.



| Field       | Description   | Access | Reset |
|-------------|---|--------|-------|
| <b>data</b> | Accesses bits 127:96 of <b>sbdata</b> (if the system bus is that wide). | R/W    | 0     |

## Chapter 4

# RISC-V Debug

Modifications to the RISC-V core to support debug are kept to a minimum. There is a special execution mode (Debug Mode) and a few extra CSRs. The DM takes care of the rest.

### 4.1 Debug Mode

Debug Mode is a special processor mode used only when the core is halted for external debugging. How Debug Mode is implemented is not specified here.

When executing code from the Program Buffer, the processor stays in Debug Mode and the following apply:

1. All operations are executed at machine mode privilege level, except that `mprv` in `mstatus` is ignored.
2. All interrupts are masked.
3. Exceptions don't update any registers. That includes `cause`, `epc`, `badaddr`, `dpc`, and `mstatus`. They do end execution of the Program Buffer.
4. No action is taken if a trigger matches.
5. Trace is disabled.
6. Counters may be stopped, depending on `stopcount` in `dcscr`.
7. Timers may be stopped, depending on `stoptime` in `dcscr`.
8. The `wfi` instruction acts as a `nop`.
9. Almost all instructions that change the privilege level have undefined behavior. This includes `ecall`, `mret`, `hret`, `sret`, and `uret`. (To change the privilege level, the debugger can write `prv` in `dcscr`). The only exception is `ebreak`. When that is executed in Debug Mode, it halts the processor again but without updating `dpc` or `dcscr`.

## 4.2 Load-Reserved/Store-Conditional Instructions

The reservation registered by an `lr` instruction on a memory address may be lost when entering Debug Mode or while in Debug Mode. This means that there may be no forward progress if Debug Mode is entered between `lr` and `sc` pairs.

## 4.3 Reset

If the halt signal (driven by the hart's halt request bit in the Debug Module) is asserted when a hart comes out of reset, the hart must enter Debug Mode before executing any instructions, but after performing any initialization that would usually happen before the first instruction is executed.

### 4.3.1 `dret` Instruction

To return from Debug Mode, a new instruction is defined: `dret`. It has an encoding of 0x7b200073. On harts which support this instruction, executing `dret` in Debug Mode changes `pc` to the value stored in `dpc`. The current privilege level is changed to that specified by `prv` in `dcsr`. The hart is no longer in debug mode.

It is not necessary for the debugger to know whether an implementation supports `dret`, as the Debug Module will ensure that it is executed if necessary. It is defined in this specification only to reserve the opcode and allow for reusable Debug Module implementations.

## 4.4 Core Debug Registers

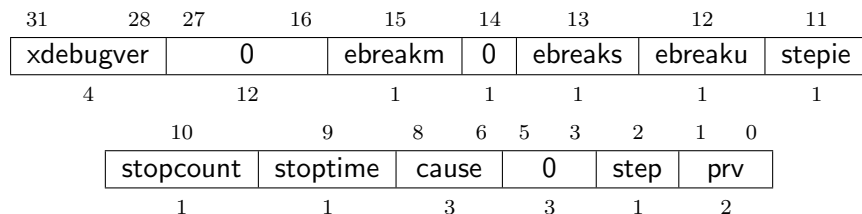
The supported Core Debug Registers must be implemented for each hart that can be debugged.

These registers are only accessible from Debug Mode.

Table 4.1: Core Debug Registers

| Address | Name                     | Page               |
|---------|--------------------------|--------------------|
| 0x7b0   | Debug Control and Status | <a href="#">33</a> |
| 0x7b1   | Debug PC                 | <a href="#">34</a> |
| 0x7b2   | Debug Scratch Register 0 |                    |
| 0x7b3   | Debug Scratch Register 1 |                    |

## 4.4.1 Debug Control and Status (dcsr, at 0x7b0)



| Field     | Description   | Access | Reset  |
|-----------|---|--------|--------|
| xdebugver | 0: There is no external debug support.<br>4: External debug support exists as it is described in this document.<br>15: There is external debug support, but it does not conform to any available version of this spec.  | R      | Preset |
| ebreakm   | When 1, <b>ebreak</b> instructions in Machine Mode enter Debug Mode.  | R/W    | 0      |
| ebreaks   | When 1, <b>ebreak</b> instructions in Supervisor Mode enter Debug Mode.   | R/W    | 0      |
| ebreaku   | When 1, <b>ebreak</b> instructions in User/Application Mode enter Debug Mode.   | R/W    | 0      |
| stepie    | 0: Interrupts are disabled during single stepping.<br>1: Interrupts are enabled during single stepping.<br>Implementations may hard wire this bit to 0. The debugger must read back the value it writes to check whether the feature is supported. If not supported, interrupt behavior can be emulated by the debugger.  | R/W    | 0      |
| stopcount | 0: Increment counters as usual.<br>1: Don't increment any counters while in Debug Mode or on <b>ebreak</b> instructions that cause entry into Debug Mode. These counters include the <b>cycle</b> and <b>instret</b> CSRs. This is preferred for most debugging scenarios.<br>An implementation may choose not to support writing to this bit. The debugger must read back the value it writes to check whether the feature is supported. | R/W    | Preset |

Continued on next page

|          |  |     |        |
|----------|--|-----|--------|
| stoptime | <p>0: Increment timers as usual.</p> <p>1: Don't increment any hart-local timers while in Debug Mode.</p> <p>An implementation may choose not to support writing to this bit. The debugger must read back the value it writes to check whether the feature is supported.</p>   | R/W | Preset |
| cause    | <p>Explains why Debug Mode was entered.</p> <p>When there are multiple reasons to enter Debug Mode in a single cycle, the cause with the highest priority is the one written.</p> <p>1: An <b>ebreak</b> instruction was executed. (priority 3)</p> <p>2: The Trigger Module caused a breakpoint exception. (priority 4)</p> <p>3: The debugger requested entry to Debug Mode. (priority 2)</p> <p>4: The hart single stepped because <b>step</b> was set. (priority 1)</p> <p>Other values are reserved for future use.</p> | R   | 0      |
| step     | <p>When set and not in Debug Mode, the hart will only execute a single instruction and then enter Debug Mode. If the instruction does not complete due to an exception, the hart will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set.</p>  | R/W | 0      |
| prv      | <p>Contains the privilege level the hart was operating in when Debug Mode was entered. The encoding is described in Table 4.4. A debugger can change this value to change the hart's privilege level when exiting Debug Mode.</p> <p>Not all privilege levels are supported on all harts. If the encoding written is not supported or the debugger is not allowed to change to it, the hart may change to any supported privilege level.</p>   | R/W | 0      |

#### 4.4.2 Debug PC (dpc, at 0x7b1)

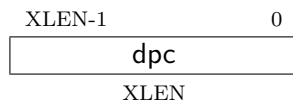
Upon entry to debug mode, **dpc** is updated with the virtual address of the next instruction to be executed. The behavior is described in more detail in Table 4.2.



Table 4.2: Virtual address in DPC upon Debug Mode Entry

| Cause          | Virtual Address in DPC  |
|----------------|---|
| <b>ebreak</b>  | Address of the <b>ebreak</b> instruction  |
| single step    | Address of the instruction that would be executed next if no debugging was going on. Ie. <code>pc + 4</code> for 32-bit instructions that don't change program flow, the destination PC on taken jumps/branches, etc. |
| trigger module | If <b>timing</b> is 0, the address of the instruction which caused the trigger to fire. If <b>timing</b> is 1, the address of the next instruction to be executed at the time that debug mode was entered.            |
| halt request   | Address of the next instruction to be executed at the time that debug mode was entered  |

When resuming, the hart's PC is updated to the virtual address stored in **dpc**. A debugger may write **dpc** to change where the hart resumes.



#### 4.4.3 Debug Scratch Register 0 (dscratch0, at 0x7b2)

Optional scratch register that can be used by implementations that need it. A debugger must not write to this register unless **hartinfo** explicitly mentions it (the Debug Module may use this register internally).

#### 4.4.4 Debug Scratch Register 1 (dscratch1, at 0x7b3)

Optional scratch register that can be used by implementations that need it. A debugger must not write to this register unless **hartinfo** explicitly mentions it (the Debug Module may use this register internally).

### 4.5 Virtual Debug Registers

Virtual debug registers are a requirement on the debugger SW/interface, not on the Core designer.

Users of the debugger shouldn't need to know about the core debug registers, but may want to change things affected by them. A virtual register is one that doesn't exist directly in the hardware, but that the debugger exposes as if it does.

Table 4.3: Virtual Core Debug Registers

| Address | Name            | Page |
|---------|-----------------|------|
| virtual | Privilege Level | 36   |

Table 4.4: Privilege Level Encoding

| Encoding | Privilege Level  |
|----------|------------------|
| 0        | User/Application |
| 1        | Supervisor       |
| 3        | Machine          |

#### 4.5.1 Privilege Level (priv, at virtual)

User can read this register to inspect the privilege level that the hart was running in when the hart halted. User can write this register to change the privilege level that the hart will run in when it resumes.

This register contains `priv` from `dcsr`, but in a place that the user is expected to access. The user should not access `dcsr` directly, because doing so might interfere with the debugger.



| Field             | Description  | Access | Reset |
|-------------------|--|--------|-------|
| <code>priv</code> | Contains the privilege level the hart was operating in when Debug Mode was entered. The encoding is described in Table 4.4, and matches the privilege level encoding from the RISC-V Privileged ISA Specification. A user can write this value to change the hart's privilege level when exiting Debug Mode. | R/W    | 0     |

## Chapter 5

# Trigger Module

Triggers can cause a breakpoint exception, entry into Debug Mode, or a trace action without having to execute a special instruction. This makes them invaluable when debugging code from ROM. They can trigger on execution of instructions at a given memory address, or on the address/data in loads/stores. These are all features that can be useful without having the Debug Module present, so the Trigger Module is broken out as a separate piece that can be implemented separately.

Each trigger may support a variety of features. A debugger can build a list of all triggers and their features as follows:

1. Write 0 to `tselect`.
2. Read back `tselect` to confirm this trigger exists. If not, exit.
3. Read `tdata1`, and possible `tdata2` and `tdata3` depending on the trigger type.
4. If `type` is 0, this trigger doesn't exist. Exit the loop.
5. Repeat, incrementing the value in `tselect`.

---

*There are two ways to check whether a given trigger is the last one to support these implementations:*

1. *When no hardware triggers are implemented at all, all related registers return 0. The algorithm above terminates when checking `type`.*
2. *When 2 triggers are implemented, `tselect` is just a single bit that selects one of the two. When the debugger writes 2, it reads back as 0 which terminates the enumeration.*

### 5.1 Trigger Registers

The trigger registers are only accessible in machine and Debug Mode to prevent untrusted user code from causing entry into Debug Mode without the OS's permission.

Table 5.1: Trigger Registers

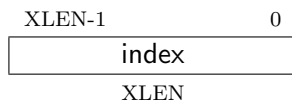
| Address | Name              | Page               |
|---------|-------------------|--------------------|
| 0x7a0   | Trigger Select    | <a href="#">38</a> |
| 0x7a1   | Trigger Data 1    | <a href="#">38</a> |
| 0x7a1   | Match Control     | <a href="#">39</a> |
| 0x7a1   | Instruction Count | <a href="#">43</a> |
| 0x7a2   | Trigger Data 2    | <a href="#">39</a> |
| 0x7a3   | Trigger Data 3    | <a href="#">39</a> |

### 5.1.1 Trigger Select (tselect, at 0x7a0)

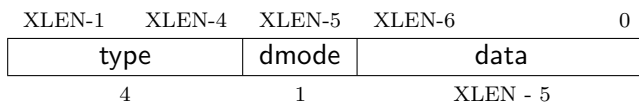
This register determines which trigger is accessible through the other trigger registers. The set of accessible triggers must start at 0, and be contiguous.

Writes of values greater than or equal to the number of supported triggers may result in a different value in this register than what was written. Debuggers should read back the value to confirm that what they wrote was a valid index.

Since triggers can be used both by Debug Mode and M Mode, the debugger must restore this register if it modifies it.



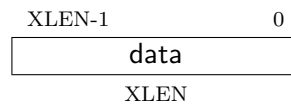
### 5.1.2 Trigger Data 1 (tdata1, at 0x7a1)



| Field | Description   | Access | Reset  |
|-------|---|--------|--------|
| type  | 0: There is no trigger at this <a href="#">tselect</a> .<br>1: The trigger is a legacy SiFive address match trigger. These should not be implemented and aren't further documented here.<br>2: The trigger is an address/data match trigger. The remaining bits in this register act as described in <a href="#">mcontrol</a> .<br>3: The trigger is an instruction count trigger. The remaining bits in this register act as described in <a href="#">icount</a> .<br>15: This trigger exists (so enumeration shouldn't terminate), but is not currently available.<br>Other values are reserved for future use. | R      | Preset |
| dmode | 0: Both Debug and M Mode can write the <code>tdata</code> registers at the selected <a href="#">tselect</a> .<br>1: Only Debug Mode can write the <code>tdata</code> registers at the selected <a href="#">tselect</a> . Writes from other modes are ignored.<br>This bit is only writable from Debug Mode.   | R/W    | 0      |
| data  | Trigger-specific data.  | R/W    | Preset |

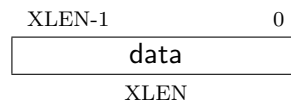
### 5.1.3 Trigger Data 2 (`tdata2`, at `0x7a2`)

Trigger-specific data.



### 5.1.4 Trigger Data 3 (`tdata3`, at `0x7a3`)

Trigger-specific data.



### 5.1.5 Match Control (`mcontrol`, at `0x7a1`)

This register is accessible as [tdata1](#) when [type](#) is 2.

Writing unsupported values to any field in this register results in the reset value being written

instead. When a debugger wants to use a feature, it must write the appropriate value and then read back the register to determine whether it is supported.

Address and data trigger implementation are heavily dependent on how the processor core is implemented. To accommodate various implementations, execute, load, and store address/data triggers may fire at whatever point in time is most convenient for the implementation. The debugger may request specific timings as described in [timing](#). Table 5.2 suggests timings for the best user experience.

Table 5.2: Suggested Breakpoint Timings

| Match Type                  | Suggested Trigger Timing |
|-----------------------------|--------------------------|
| Execute Address             | Before                   |
| Execute Instruction         | Before                   |
| Execute Address+Instruction | Before                   |
| Load Address                | Before                   |
| Load Data                   | After                    |
| Load Address+Data           | After                    |
| Store Address               | Before                   |
| Store Data                  | Before                   |
| Store Address+Data          | Before                   |

|        |        |         |           |         |         |    |         |        |      |   |   |
|--------|--------|---------|-----------|---------|---------|----|---------|--------|------|---|---|
| XLEN-1 | XLEN-4 | XLEN-5  | XLEN-6    | XLEN-11 | XLEN-12 | 20 | 19      | 18     |      |   |   |
| type   | dmode  | maskmax | 0         |         |         |    | select  | timing |      |   |   |
| 4      | 1      | 6       | XLEN - 31 |         |         |    | 1       | 1      |      |   |   |
| 17     | 12     | 11      | 10        | 7       | 6       | 5  | 4       | 3      | 2    | 1 | 0 |
| action | chain  | match   | m         | h       | s       | u  | execute | store  | load |   |   |
| 6      | 1      | 4       | 1         | 1       | 1       | 1  | 1       | 1      | 1    | 1 |   |

| Field                  | Description  | Access | Reset  |
|------------------------|--|--------|--------|
| <b>maskmax</b>         | Specifies the largest naturally aligned powers-of-two (NAPOT) range supported by the hardware. The value is the logarithm base 2 of the number of bytes in that range. A value of 0 indicates that only exact value matches are supported (one byte range). A value of 63 corresponds to the maximum NAPOT range, which is $2^{63}$ bytes in size. | R      | Preset |
| Continued on next page |  |        |        |

|                        |  |     |   |
|------------------------|--|-----|---|
| select                 | 0: Perform a match on the virtual address.<br>1: Perform a match on the data value loaded/stored, or the instruction executed.   | R/W | 0 |
| timing                 | 0: The action for this trigger will be taken just before the instruction that triggered it is executed, but after all preceding instructions are committed.<br>1: The action for this trigger will be taken after the instruction that triggered it is executed. It should be taken before the next instruction is executed, but it is better to implement triggers and not implement that suggestion than to not implement them at all.<br>Most hardware will only implement one timing or the other, possibly dependent on <a href="#">select</a> , <a href="#">execute</a> , <a href="#">load</a> , and <a href="#">store</a> . This bit primarily exists for the hardware to communicate to the debugger what will happen. Hardware may implement the bit fully writable, in which case the debugger has a little more control.<br>Data load triggers with <a href="#">timing</a> of 0 will result in the same load happening again when the debugger lets the core run. For data load triggers, debuggers must first attempt to set the breakpoint with <a href="#">timing</a> of 1.<br>A chain of triggers that don't all have the same <a href="#">timing</a> value will never fire (unless consecutive instructions match the appropriate triggers). | R/W | 0 |
| Continued on next page |  |     |   |

|        |  |     |   |
|--------|--|-----|---|
| action | <p>Determines what happens when this trigger matches.</p> <p>0: Raise a breakpoint exception. (Used when software wants to use the trigger module without an external debugger attached.)</p> <p>1: Enter Debug Mode. (Only supported when <code>dmode</code> is 1.)</p> <p>2: Start tracing.</p> <p>3: Stop tracing.</p> <p>4: Emit trace data for this match. If it is a data access match, emit appropriate Load/Store Address/Data. If it is an instruction execution, emit its PC.</p> <p>Other values are reserved for future use.</p>   | R/W | 0 |
| chain  | <p>0: When this trigger matches, the configured action is taken.</p> <p>1: While this trigger does not match, it prevents the trigger with the next index from matching.</p>   | R/W | 0 |
| match  | <p>0: Matches when the value equals <code>tdata2</code>.</p> <p>1: Matches when the top M bits of the value match the top M bits of <code>tdata2</code>. M is XLEN-1 minus the index of the least-significant bit containing 0 in <code>tdata2</code>.</p> <p>2: Matches when the value is greater than (unsigned) or equal to <code>tdata2</code>.</p> <p>3: Matches when the value is less than (unsigned) <code>tdata2</code>.</p> <p>4: Matches when the lower half of the value equals the lower half of <code>tdata2</code> after the lower half of the value is ANDed with the upper half of <code>tdata2</code>.</p> <p>5: Matches when the upper half of the value equals the lower half of <code>tdata2</code> after the upper half of the value is ANDed with the upper half of <code>tdata2</code>.</p> <p>Other values are reserved for future use.</p> | R/W | 0 |

Continued on next page



|         |  |     |   |
|---------|--|-----|---|
| m       | When set, enable this trigger in M mode.   | R/W | 0 |
| h       | When set, enable this trigger in H mode.   | R/W | 0 |
| s       | When set, enable this trigger in S mode.   | R/W | 0 |
| u       | When set, enable this trigger in U mode.   | R/W | 0 |
| execute | When set, the trigger fires on the virtual address or opcode of an instruction that is executed. | R/W | 0 |
| store   | When set, the trigger fires on the virtual address or data of a store.                           | R/W | 0 |
| load    | When set, the trigger fires on the virtual address or data of a load.                            | R/W | 0 |

### 5.1.6 Instruction Count (icount, at 0x7a1)

This register is accessible as `tdata1` when `type` is 3.

Writing unsupported values to any field in this register results in the reset value being written instead. When a debugger wants to use a feature, it must write the appropriate value and then read back the register to determine whether it is supported.

---

*This trigger type is intended to be used as a single step that's useful both for external debuggers and for software monitor programs. For that case it is not necessary to support `count` greater than 1. The only two combinations of the mode bits that are useful in those scenarios are `u` by itself, or `m`, `h`, `s`, and `u` all set.*

*If the hardware limits `count` to 1, and changes mode bits instead of decrementing `count`, this register can be implemented with just 2 bits. One for `u`, and one for `m`, `h`, and `s` tied together. If only the external debugger or only a software monitor needs to be supported, a single bit is enough.*

|        |        |           |        |    |       |    |   |   |   |        |   |   |
|--------|--------|-----------|--------|----|-------|----|---|---|---|--------|---|---|
| XLEN-1 | XLEN-4 | XLEN-5    | XLEN-6 | 24 | 23    | 10 | 9 | 8 | 7 | 6      | 5 | 0 |
| type   | dmode  | 0         |        |    | count | m  | h | s | u | action |   |   |
| 4      | 1      | XLEN - 29 |        |    | 14    | 1  | 1 | 1 | 1 | 6      |   |   |

| Field                  | Description   | Access | Reset |
|------------------------|---|--------|-------|
| count                  | When count is decremented to 0, the trigger fires. Instead of changing <code>count</code> from 1 to 0, it is also acceptable for hardware to clear <code>m</code> , <code>h</code> , <code>s</code> , and <code>u</code> . This allows <code>count</code> to be hard-wired to 1 if this register just exists for single step. | R/W    | 1     |
| Continued on next page |   |        |       |

|        |   |     |   |
|--------|---|-----|---|
| m      | When set, every instruction completed or exception taken in M mode decrements <a href="#">count</a> by 1.   | R/W | 0 |
| h      | When set, every instruction completed or exception taken in in H mode decrements <a href="#">count</a> by 1.  | R/W | 0 |
| s      | When set, every instruction completed or exception taken in S mode decrements <a href="#">count</a> by 1.   | R/W | 0 |
| u      | When set, every instruction completed or exception taken in U mode decrements <a href="#">count</a> by 1.   | R/W | 0 |
| action | <p>Determines what happens when this trigger matches.</p> <p>0: Raise a breakpoint exception. (Used when software wants to use the trigger module without an external debugger attached.)</p> <p>1: Enter Debug Mode. (Only supported when <a href="#">dmode</a> is 1.)</p> <p>2: Start tracing.</p> <p>3: Stop tracing.</p> <p>4: Emit trace data for this match. If it is a data access match, emit appropriate Load/Store Address/Data. If it is an instruction execution, emit its PC.</p> <p>Other values are reserved for future use.</p> | R/W | 0 |

## Chapter 6

# Debug Transport Module (DTM)

Debug Transport Modules provide access to the DM over one or more transports (eg. JTAG or USB).

There may be multiple DTMs in a single platform. Ideally every component that communicates with the outside world includes a DTM, allowing a platform to be debugged through every transport it supports. For instance a USB component could include a DTM. This would trivially allow any platform to be debugged over USB. All that is required is that the USB module already in use also has access to the Debug Module Interface.

Using multiple DTMs at the same time is not supported. It is left to the user to ensure this does not happen.

This specification defines a JTAG DTM in Section 6.1. Additional DTMs may be added in future versions of this specification.

## 6.1 JTAG Debug Transport Module

This Debug Transport Module is based around a normal JTAG Test Access Port (TAP). The JTAG TAP allows access to arbitrary JTAG registers by first selecting one using the JTAG instruction register (IR), and then accessing it through the JTAG data register (DR).

### 6.1.1 JTAG Background

JTAG refers to IEEE Std 1149.1-2013. It is a standard that defines test logic that can be included in an integrated circuit to test the interconnections between integrated circuits, test the integrated circuit itself, and observe or modify circuit activity during the components normal operation. This specification uses the latter functionality. The JTAG standard defines a Test Access Port (TAP) that can be used to read and write a few custom registers, which can be used to communicate with debug hardware in a component.

### 6.1.2 JTAG DTM Registers

JTAG TAPs used as a DTM must have an IR of at least 5 bits. When the TAP is reset, IR must default to 00001, selecting the IDCODE instruction. A full list of JTAG registers along with their encoding is in Table 6.1. If the IR actually has more than 5 bits, then the encodings in Table 6.1 should be extended with 0's in their most significant bits. The only regular JTAG registers a debugger might use are BYPASS and IDCODE, but this specification leaves IR space for many other standard JTAG instructions. Unimplemented instructions must select the BYPASS register.

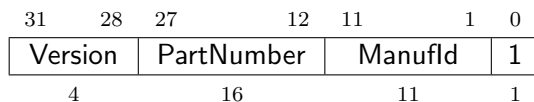
Table 6.1: JTAG DTM TAP Registers

| Address | Name                          | Description                          | Page     |
|---------|-------------------------------|--------------------------------------|----------|
| 0x00    | BYPASS                        | JTAG recommends this encoding        | 47<br>48 |
| 0x01    | IDCODE                        | JTAG recommends this encoding        |          |
| 0x10    | DTM Control and Status        | For Debugging                        |          |
| 0x11    | Debug Module Interface Access | For Debugging                        |          |
| 0x12    | Reserved (BYPASS)             | Reserved for future RISC-V debugging |          |
| 0x13    | Reserved (BYPASS)             | Reserved for future RISC-V debugging |          |
| 0x14    | Reserved (BYPASS)             | Reserved for future RISC-V debugging |          |
| 0x15    | Reserved (BYPASS)             | Reserved for future RISC-V standards |          |
| 0x16    | Reserved (BYPASS)             | Reserved for future RISC-V standards |          |
| 0x17    | Reserved (BYPASS)             | Reserved for future RISC-V standards |          |
| 0x1f    | BYPASS                        | JTAG requires this encoding          |          |

### 6.1.3 IDCODE (at 0x01)

This register is selected (in IR) when the TAP state machine is reset. Its definition is exactly as defined in IEEE Std 1149.1-2013.

This entire register is read-only.



| Field      | Description   | Access | Reset  |
|------------|---|--------|--------|
| Version    | Identifies the release version of this part.  | R      | Preset |
| PartNumber | Identifies the designer's part number of this part.   | R      | Preset |
| Manufld    | Identifies the designer/manufacture of this part. Bits 6:0 must be bits 6:0 of the designer/manufacture's Identification Code as assigned by JEDEC Standard JEP106. Bits 10:7 contain the modulo-16 count of the number of continuation characters (0x7f) in that same Identification Code. | R      | Preset |

### 6.1.4 DTM Control and Status (dtmcs, at 0x10)

The size of this register will remain constant in future versions so that a debugger can always determine the version of the DTM.

|    |              |          |    |      |         |       |         |    |   |   |   |   |
|----|--------------|----------|----|------|---------|-------|---------|----|---|---|---|---|
| 31 | 18           | 17       | 16 | 15   | 14      | 12    | 11      | 10 | 9 | 4 | 3 | 0 |
| 0  | dmihardreset | dmireset | 0  | idle | dmistat | abits | version |    |   |   |   |   |
| 14 | 1            | 1        | 1  | 3    | 2       | 6     | 4       |    |   |   |   |   |

| Field        | Description   | Access | Reset  |
|--------------|---|--------|--------|
| dmihardreset | Writing 1 to this bit does a hard reset of the DTM, causing the DTM to forget about any outstanding DMI transactions. In general this should only be used when the Debugger has reason to expect that the outstanding DMI transaction will never complete (e.g. a reset condition caused an inflight DMI transaction to be cancelled).  | W1     | 0      |
| dmireset     | Writing 1 to this bit clears the sticky error state and allows the DTM to retry or complete the previous transaction.   | W1     | 0      |
| idle         | This is a hint to the debugger of the minimum number of cycles a debugger should spend in Run-Test/Idle after every DMI scan to avoid a ‘busy’ return code ( <a href="#">dmistat</a> of 3). A debugger must still check <a href="#">dmistat</a> when necessary.<br>0: It is not necessary to enter Run-Test/Idle at all.<br>1: Enter Run-Test/Idle and leave it immediately.<br>2: Enter Run-Test/Idle and stay there for 1 cycle before leaving.<br>And so on. | R      | Preset |

Continued on next page

|                |  |   |        |
|----------------|--|---|--------|
| <b>dmistat</b> | 0: No error.<br>1: Reserved. Interpret the same as 2.<br>2: An operation failed (resulted in <b>op</b> of 2).<br>3: An operation was attempted while a DMI access was still in progress (resulted in <b>op</b> of 3).  | R | 0      |
| <b>abits</b>   | The size of <b>address</b> in <b>dmi</b> .   | R | Preset |
| <b>version</b> | 0: Version described in spec version 0.11.<br>1: Version described in spec version 0.13 (and later?), which reduces the DMI data width to 32 bits.<br>15: Version not described in any available version of this spec. | R | 1      |

### 6.1.5 Debug Module Interface Access (dmi, at 0x11)

This register allows access to the Debug Module Interface (DMI).

In Update-DR, the DTM starts the operation specified in **op** unless the current status reported in **op** is sticky.

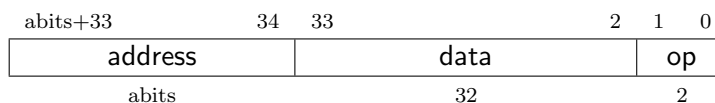
In Capture-DR, the DTM updates **data** with the result from that operation, updating **op** if the current **op** isn't sticky.

See Section B.1 and Table B.1 for examples of how this is used.

---

*The still-in-progress status is sticky to accommodate debuggers that batch together a number of scans, which must all be executed or stop as soon as there's a problem.*

*For instance a series of scans may write a Debug Program and execute it. If one of the writes fails but the execution continues, then the Debug Program may hang or have other unexpected side effects.*



| Field                  | Description  | Access | Reset |
|------------------------|--|--------|-------|
| <b>address</b>         | Address used for DMI access. In Update-DR this value is used to access the DM over the DMI.  | R/W    | 0     |
| <b>data</b>            | The data to send to the DM over the DMI during Update-DR, and the data returned from the DM as a result of the previous operation. | R/W    | 0     |
| Continued on next page |  |        |       |

|    |   |     |   |
|----|---|-----|---|
| op | <p>When the debugger writes this field, it has the following meaning:</p> <p>0: Ignore <b>data</b> and <b>address</b>. (nop)</p> <p>Don't send anything over the DMI during Update-DR. This operation should never result in a busy or error response. The address and data reported in the following Capture-DR are undefined.</p> <p>1: Read from <b>address</b>. (read)</p> <p>2: Write <b>data</b> to <b>address</b>. (write)</p> <p>3: Reserved.</p> <p>When the debugger reads this field, it means the following:</p> <p>0: The previous operation completed successfully.</p> <p>1: Reserved.</p> <p>2: A previous operation failed. The data scanned into <b>dmi</b> in this access will be ignored. This status is sticky and can be cleared by writing <b>dmireset</b> in <b>dtmcs</b>.</p> <p>This indicates that the DM itself responded with an error. Note: there are no specified cases in which the DM would respond with an error, and DMI is not required to support returning errors.</p> <p>3: An operation was attempted while a DMI request is still in progress. The data scanned into <b>dmi</b> in this access will be ignored. This status is sticky and can be cleared by writing <b>dmireset</b> in <b>dtmcs</b>. If a debugger sees this status, it needs to give the target more TCK edges between Update-DR and Capture-DR. The simplest way to do that is to add extra transitions in Run-Test/Idle. (The DTM, DM, and/or component may be in different clock domains, so synchronization may be required. Some relatively fixed number of TCK ticks may be needed for the request to reach the DM, complete, and for the response to be synchronized back into the TCK domain.)</p> | R/W | 2 |
|----|---|-----|---|

### 6.1.6 BYPASS (at 0x1f)

1-bit register that has no effect. It is used when a debugger does not want to communicate with this TAP.

This entire register is read-only.



### 6.1.7 Recommended JTAG Connector

To make it easy to acquire debug hardware, this spec recommends a connector that is compatible with the Atmel AVR JTAG Connector, as described below.

The connector is a .05”-spaced, gold-plated male header with .016” thick hardened copper or beryllium bronze square posts (SAMTEC FTSH-105 or equivalent). Female connectors are compatible 20 $\mu$ m gold connectors.

Viewing the male header from above (the pins pointing at your eye), a target’s connector looks as it does in Table 6.2. The function of each pin is described in Table 6.3.

Table 6.2: JTAG Connector Diagram

|      |   |    |         |
|------|---|----|---------|
| TCK  | 1 | 2  | GND     |
| TDO  | 3 | 4  | VCC     |
| TMS  | 5 | 6  | (SRSTn) |
| (NC) | 7 | 8  | (TRSTn) |
| TDI  | 9 | 10 | GND     |

Table 6.3: JTAG Connector Pinout

|       |       |   |
|-------|-------|---|
| 1     | TCK   | JTAG TCK signal, driven by the debug adapter. This pin must be clearly marked in both male and female headers.  |
| 5     | TMS   | JTAG TMS signal, driven by debug adapter.   |
| 9     | TDI   | JTAG TDI signal, driven by the debug adapter.   |
| 3     | TDO   | JTAG TDO signal, driven by the target.  |
| 8     | TRSTn | Test Reset (optional, only used by some devices. Used to reset the JTAG TAP Controller).  |
| 4     | VCC   | Reference voltage for logic high. A debug adapter may attempt to draw up to 20mA from this pin to power itself, but a target is not obligated to provide that power.  |
| 2, 10 | GND   | Target ground.  |
| 6     | SRSTn | Active-low reset signal, driven by the debug adapter. Asserting reset should reset any RISC-V cores as well as any other peripherals on the PCB. It should not reset the debug logic. Although connecting this pin is optional, it is recommended as it allows the debugger to hold the target device in a reset state, which may be essential to debug some scenarios. If not implemented in a target, this pin must not be connected. |



Target connectors may be shrouded. In that case the key slot should be next to pin 5. Female headers should have a matching key.

Debug adapters should be tagged or marked with their isolation voltage threshold (i.e. unisolated, 250V, etc.).

All debug adapter pins other than GND should be current-limited to 20mA.



# Appendix A

## Hardware Implementations

Below are two possible implementations. A designer could choose one, mix and match, or come up with their own design.

### A.1 Abstract Command Based

Halting happens by stalling the processor execution pipeline.

Muxes on the register file(s) allow for accessing GPRs and CSRs using the Access Register abstract command.

System Bus Access allows main memory access.

### A.2 Execution Based

This implementation only implements the Access Register abstract command for GPRs on a halted hart, and relies on the Program Buffer for all other operations.

This method uses the processor’s existing pipeline and ability to execute from arbitrary memory locations to avoid modifications to a processor’s datapath. When the halt request bit is set, the Debug Module raises a special interrupt to the selected hart(s). This interrupt causes each hart to enter Debug Mode and jump to a defined memory region that is serviced by the DM. When taking this exception, `pc` is saved to `dpc` and `cause` is updated in `dcsr`.

The code in the Debug Module causes the hart to execute a “park loop”. In the park loop the hart writes its `mhartid` to a memory location within the Debug Module to indicate that it is halted. To allow the DM to individually control one out of several halted harts, each hart polls for flags in a DM-controlled memory location to determine whether the debugger wants it to execute the Program Buffer or perform a resume.

To execute an abstract command, the DM first populates some internal words of program buffer

according to `command`. When `transfer` is set, the debugger populates these words with `lw <gpr>, 0x400(zero)` or `sw 0x400(zero), <gpr>`. 64- and 128-bit accesses use `ld/sd` and `lq/sq` respectively. If `transfer` is not set, these instructions are populated as `nops`. If `execute` is set, execution continues to the debugger-controlled Program Buffer, otherwise the debug module causes a `ebreak` to execute immediately.

When `ebreak` is executed (indicating the end of the Program Buffer code) the hart returns to its park loop. If an exception is encountered, the hart jumps to a defined debug exception address within the Debug Module. The code at that address causes the hart to write to an address in the Debug Module which indicates exception. Then the hart jumps back to the park loop. The DM infers from the write that there was an exception, and sets `cmderr` appropriately.

To resume execution, the debug module sets a flag which causes the core to execute a `dret`. When `dret` is executed, `pc` is restored from `dpc` and normal execution resumes at the privilege set by `prv`.

`data0` etc. are mapped into regular memory at an address relative to `zero` with only a 12-bit `imm`. The exact address is an implementation detail that a debugger must not rely on. For example, the `data` registers might be mapped to `0x400`.

For additional flexibility, `progbuf0`, etc. are mapped into regular memory immediately preceding `data0`, in order to form a contiguous region of memory which can be used for either program execution or data transfer.

## Appendix B

# Debugger Implementation

This section details how an external debugger might use the described debug interface to perform some common operations on RISC-V cores using the JTAG DTM described in Appendix ???. All these examples assume a 32-bit core but it should be easy to adapt the examples to 64- or 128-bit cores.

To keep the examples readable, they all assume that everything succeeds, and that they complete faster than the debugger can perform the next access. This will be the case in a typical JTAG setup. However, the debugger must always check the sticky error status bits after performing a sequence of actions. If it sees any that are set, then it should attempt the same actions again, possibly while adding in some delay, or explicit checks for status bits.

### B.1 Debug Module Interface Access

To read an arbitrary Debug Module register, select `dmi`, and scan in a value with `op` set to 1, and `address` set to the desired register address. In Update-DR the operation will start, and in Capture-DR its results will be captured into `data`. If the operation didn't complete in time, `op` will be 3 and the value in `data` must be ignored. The busy condition must be cleared by writing `dmireset` in `dtmcs`, and then the second scan must be performed again. This process must be repeated until `op` returns 0. In later operations the debugger should allow for more time between Capture-DR and Update-DR.

To write an arbitrary Debug Bus register, select `dmi`, and scan in a value with `op` set to 2, and `address` and `data` set to the desired register address and data respectively. From then on everything happens exactly as with a read, except that a write is performed instead of the read.

It should almost never be necessary to scan IR, avoiding a big part of the inefficiency in typical JTAG use.

## B.2 Main Loop

A debugger continuously monitors `haltsum` to see if any harts have spontaneously halted.

## B.3 Halting

To halt one or more harts, the debugger selects them, sets `haltreq`, and then waits for `allhalted` to indicate the harts are halted before clearing `haltreq` to 0.

## B.4 Running

First, the debugger should restore any registers that it has clobbered. Once that's done, it can let the selected harts run by setting `resumereq`. Once `allresumeack` is set, the debugger knows the hart has resumed, and it can clear `resumereq`. Note that harts might halt very quickly after resuming (eg. by hitting a software breakpoint) so the debugger cannot use `allhalted/anyhalted` to check whether the hart resumed.

## B.5 Single Step

Using the hardware single step feature is almost the same as regular running. The debugger just sets `step` in `dcsr` before letting the core run. The core behaves exactly as in the running case, except that interrupts may be disabled (depending on `stepie`) and it only fetches and executes a single instruction before re-entering Debug Mode.

## B.6 Accessing Registers

### B.6.1 Using Abstract Command

Read `s0` using abstract command:

| Op    | Address              | Value                                   | Comment                                   |
|-------|----------------------|---|---|
| Write | <code>command</code> | <code>size = 2, transfer, 0x1008</code> | Read <code>s0</code>                      |
| Read  | <code>data0</code>   | -                                       | Returns value that was in <code>s0</code> |

Write `mstatus` using abstract command:

| Op    | Address              | Value   | Comment                    |
|-------|----------------------|---|----------------------------|
| Write | <code>data0</code>   | new value                                     |                            |
| Write | <code>command</code> | <code>size = 2, transfer, write, 0x300</code> | Write <code>mstatus</code> |

### B.6.2 Using Program Buffer

Abstract commands are used to exchange data with GPRs. Using this mechanism, other registers can be accessed by moving their value into/out of GPRs.

Write `mstatus` using program buffer:

| Op    | Address               | Value  | Comment   |
|-------|-----------------------|--|---|
| Write | <code>progbuf0</code> | <code>csrw s0, MSTATUS</code>                            |   |
| Write | <code>progbuf1</code> | <code>ebreak</code>                                      |   |
| Write | <code>data0</code>    | new value  |   |
| Write | <code>command</code>  | <code>size = 2, postexec, transfer, write, 0x1008</code> | Write <code>s0</code> , then execute program buffer |

Read `f1` using program buffer:

| Op    | Address               | Value                        | Comment                                       |
|-------|-----------------------|------------------------------|---|
| Write | <code>progbuf0</code> | <code>fmv.x.s s0, f1</code>  |   |
| Write | <code>progbuf1</code> | <code>ebreak</code>          |   |
| Write | <code>command</code>  | <code>postexec</code>        | Execute program buffer                        |
| Write | <code>command</code>  | <code>transfer 0x1008</code> | read <code>s0</code>                          |
| Read  | <code>data0</code>    | -                            | Returns the value that was in <code>f1</code> |

## B.7 Reading Memory

### B.7.1 Using System Bus Access

Read a word from memory using system bus access:

| Op    | Address                 | Value                                  | Comment                |
|-------|-------------------------|--|------------------------|
| Write | <code>sbaddress0</code> | address                                |                        |
| Write | <code>sbc</code>        | <code>sbaccess = 2, sbingleread</code> | Perform a read         |
| Read  | <code>sbdatab0</code>   | -                                      | Value read from memory |

Read block of memory using system bus access:

| Op    | Address                    | Value  | Comment  |
|-------|----------------------------|--|--|
| Write | <a href="#">sbaddress0</a> | address  |  |
| Write | <a href="#">sbcs</a>       | <a href="#">sbaccess</a> = 2,<br><a href="#">sbsingleread</a> ,<br><a href="#">sbautoread</a> ,<br><a href="#">sbautoincrement</a> | Turn on autoread and autoincrement, and perform a read |
| Read  | <a href="#">sbddata0</a>   | -  | Value read from memory                                 |
| Read  | <a href="#">sbddata0</a>   | -  | Next value read from memory                            |
| ...   | ...                        | ...  | ...  |
| Write | <a href="#">sbcs</a>       | 0  | Clear <a href="#">sbautoread</a>                       |
| Read  | <a href="#">sbddata0</a>   | -  | Get last value read from memory.                       |

### B.7.2 Using Program Buffer

Read a word from memory using program buffer:

| Op    | Address                  | Value                                | Comment   |
|-------|--------------------------|--------------------------------------|---|
| Write | <a href="#">progbuf0</a> | <code>lw s0, 0(s0)</code>            |   |
| Write | <a href="#">progbuf1</a> | <code>ebreak</code>                  |   |
| Write | <a href="#">data0</a>    | address                              |   |
| Write | <a href="#">command</a>  | <code>write, postexec, 0x1008</code> | Write <code>s0</code> , then execute program buffer |
| Write | <a href="#">command</a>  | <code>0x1008</code>                  | Read <code>s0</code>                                |
| Read  | <a href="#">data0</a>    | -                                    | Value read from memory                              |

Read block of memory using program buffer:

| Op    | Address                      | Value                                | Comment  |
|-------|------------------------------|--------------------------------------|--|
| Write | <a href="#">progbuf0</a>     | <code>lw s1, 0(s0)</code>            |  |
| Write | <a href="#">progbuf1</a>     | <code>addi s0, s0, 4</code>          |  |
| Write | <a href="#">progbuf2</a>     | <code>ebreak</code>                  |  |
| Write | <a href="#">data0</a>        | address                              |  |
| Write | <a href="#">command</a>      | <code>write, postexec, 0x1008</code> | Write <code>s0</code> , then execute program buffer          |
| Write | <a href="#">command</a>      | <code>postexec, 0x1009</code>        | Read <code>s1</code> , then execute program buffer           |
| Write | <a href="#">abstractauto</a> | <a href="#">autoexecdata</a> [0]     | Set <a href="#">autoexecdata</a> [0]                         |
| Read  | <a href="#">data0</a>        | -                                    | Get value read from memory, then execute program buffer      |
| Read  | <a href="#">data0</a>        | -                                    | Get next value read from memory, then execute program buffer |
| ...   | ...                          | ...                                  | ...  |
| Write | <a href="#">abstractauto</a> | 0                                    | Clear <a href="#">autoexecdata</a> [0]                       |
| Read  | <a href="#">data0</a>        | -                                    | Get last value read from memory.                             |

TODO: Table [B.1](#) shows the scans involved in reading a single word using this method.



Table B.1: Memory Read Timeline

|      |            |          |
|------|------------|----------|
|      | JTAG State | Activity |
| TODO | TODO       | TODO     |

## B.8 Writing Memory

### B.8.1 Using System Bus Access

Write a word to memory using system bus access:

| Op    | Address                 | Value   | Comment |
|-------|-------------------------|---------|---------|
| Write | <code>sbaddress0</code> | address |         |
| Write | <code>sbddata0</code>   | value   |         |

Write block of memory using system bus access:

| Op    | Address                 | Value  | Comment               |
|-------|-------------------------|--|-----------------------|
| Write | <code>sbaddress0</code> | address  |                       |
| Write | <code>sbcsc</code>      | <code>sbaccess = 2,</code><br><code>sbautoincrement</code> | Turn on autoincrement |
| Write | <code>sbddata0</code>   | value0   |                       |
| Write | <code>sbddata0</code>   | value1   |                       |
| ...   | ...                     | ...  | ...                   |
| Write | <code>sbddata0</code>   | valueN   |                       |

### B.8.2 Using Program Buffer

Write a word to memory using program buffer:

| Op    | Address               | Value                                | Comment   |
|-------|-----------------------|--------------------------------------|---|
| Write | <code>progbuf0</code> | <code>sw s1, 0(s0)</code>            |   |
| Write | <code>progbuf1</code> | <code>ebreak</code>                  |   |
| Write | <code>data0</code>    | value                                |   |
| Write | <code>command</code>  | <code>write, 0x1008</code>           | Write <code>s0</code>                               |
| Write | <code>data0</code>    | address                              |   |
| Write | <code>command</code>  | <code>write, postexec, 0x1009</code> | Write <code>s1</code> , then execute program buffer |

Write block of memory using program buffer:

| Op    | Address                      | Value                                | Comment   |
|-------|------------------------------|--------------------------------------|---|
| Write | <a href="#">progbuf0</a>     | <code>sw s1, 0(s0)</code>            |   |
| Write | <a href="#">progbuf1</a>     | <code>addi s0, s0, 4</code>          |   |
| Write | <a href="#">progbuf2</a>     | <code>ebreak</code>                  |   |
| Write | <a href="#">data0</a>        | address                              |   |
| Write | <a href="#">command</a>      | <code>write, 0x1008</code>           | Write <code>s0</code>                               |
| Write | <a href="#">data0</a>        | value0                               |   |
| Write | <a href="#">command</a>      | <code>write, postexec, 0x1009</code> | Write <code>s1</code> , then execute program buffer |
| Write | <a href="#">abstractauto</a> | <code>autoexecdata [0]</code>        | Set <code>autoexecdata [0]</code>                   |
| Write | <a href="#">data0</a>        | value1                               |   |
| ...   | ...                          | ...                                  | ...   |
| Write | <a href="#">data0</a>        | valueN                               |   |
| Write | <a href="#">abstractauto</a> | 0                                    | Clear <code>autoexecdata [0]</code>                 |

## B.9 Handling Exceptions

Generally the debugger can avoid exceptions by being careful with the programs it writes. Sometimes they are unavoidable though, eg. if the user asks to access memory or a CSR that is not implemented. A typical debugger will not know enough about the platform to know what's going to happen, and must attempt the access to determine the outcome.

When an exception occurs while executing the Program Buffer, `cmderr` becomes set. The debugger can check this field to see whether a program encountered an exception. If there was an exception, it's left to the debugger to know what must have caused it.

## B.10 Quick Access

Halt the hart for a minimum amount of time to perform a single memory write.

There are a variety of instructions to transfer data between GPRs and the `data` registers. They are either loads/stores or CSR reads/writes. The specific addresses also vary. This is all specified in [hartinfo](#). The example here uses the pseudo-op `transfer dest, src` to represent all these options.

| Op    | Address                  | Value             | Comment                       |
|-------|--------------------------|-------------------|-------------------------------|
| Write | <a href="#">progbuf0</a> | transfer arg2, s0 | Save <b>s0</b>                |
| Write | <a href="#">progbuf1</a> | transfer s0, arg0 | Read first argument (address) |
| Write | <a href="#">progbuf2</a> | transfer arg0, s1 | Save <b>s1</b>                |
| Write | <a href="#">progbuf3</a> | transfer s1, arg1 | Read second argument (data)   |
| Write | <a href="#">progbuf4</a> | sw s1, 0(s0)      |                               |
| Write | <a href="#">progbuf5</a> | transfer s1, arg0 | Restore <b>s1</b>             |
| Write | <a href="#">progbuf6</a> | transfer s0, arg2 | Restore <b>s0</b>             |
| Write | <a href="#">progbuf7</a> | ebreak            |                               |
| Write | <a href="#">data0</a>    | address           |                               |
| Write | <a href="#">data1</a>    | data              |                               |
| Write | <a href="#">command</a>  | 0x10000000        | Perform quick access          |



# Appendix C

## Future Ideas

**All items in this section are future ideas and should not be considered part of the specification.**

Some future version of this spec may implement some of the following features.

1. The spec defines several additions to the Device Tree which enable a debugger to discover hart IDs and supported triggers for all the cores in the system.
2. DTMs can function as general bus slaves, so they would look like regular RAM to bus masters.
3. Harts can be divided into groups. All the harts in the same group can be halted/run/stepped simultaneously. When a hart hits a breakpoint, all the other harts in the same group also halt within a few clock cycles.
4. DTMs are specified for protocols like USB, I2C, SPI, and SWD.
5. Core registers can be read without halting the processor.
6. The debugger can communicate with the power manager to power cores up or down, and to query their status.
7. Serial ports can raise an interrupt when a send/receive queue becomes full/empty.
8. The debug interrupt can be masked by running code. If the interrupt is asserted, then deasserted, and then asserted again the debug interrupt happens anyway. This mechanism can be used to eg. read/write memory with minimal interruption, making sure never to interrupt during a critical piece of code.
9. The debugger can non-intrusively sample a recent PC value from any running hart.
10. The Debug Module can include a serial interface for re-using the DTM interface as a generic communication interface.

## C.1 Serial Ports

The Debug Module may implement up to 8 serial ports. They support basic flow control and full duplex data transfer between a component and the debugger, essentially allowing the Debug Transport to be used to communicate with a debug monitor running on a hart, or more generally emulate devices which aren't present. All these uses require software support, and are not further specified here. Only the DMI side of the Debug Module serial registers are defined in this specification as the core side interface should look like a peripheral device.

Table C.1: Debug Module Debug Bus Registers

| Address | Name                      | Page               |
|---------|---------------------------|--------------------|
| 0x34    | Serial Control and Status | <a href="#">64</a> |
| 0x35    | Serial TX Data            | <a href="#">65</a> |
| 0x36    | Serial RX Data            | <a href="#">65</a> |

### C.1.1 Serial Control and Status (`sercs`, at 0x34)

If [serialcount](#) is 0, this register is not present.

|             |        |        |        |        |       |        |        |       |    |    |
|-------------|--------|--------|--------|--------|-------|--------|--------|-------|----|----|
| 31          | 28     | 27     | 26     | 24     | 23    | 22     | 21     | 20    | 19 | 18 |
| serialcount | 0      | serial | error7 | valid7 | full7 | error6 | valid6 | full6 |    |    |
| 4           | 1      | 3      | 1      | 1      | 1     | 1      | 1      | 1     | 1  |    |
| 17          | 16     | 15     | 14     | 13     | 12    | 11     | 10     | 9     |    |    |
| error5      | valid5 | full5  | error4 | valid4 | full4 | error3 | valid3 | full3 |    |    |
| 1           | 1      | 1      | 1      | 1      | 1     | 1      | 1      | 1     |    |    |
| 8           | 7      | 6      | 5      | 4      | 3     | 2      | 1      | 0     |    |    |
| error2      | valid2 | full2  | error1 | valid1 | full1 | error0 | valid0 | full0 |    |    |
| 1           | 1      | 1      | 1      | 1      | 1     | 1      | 1      | 1     |    |    |

| Field                    | Description   | Access | Reset  |
|--------------------------|---|--------|--------|
| <code>serialcount</code> | Number of supported serial ports.   | R      | Preset |
| <code>serial</code>      | Select which serial port is accessed by <code>serrx</code> and <code>sertx</code> .   | R/W    | 0      |
| <code>error0</code>      | 1 when the debugger-to-core queue for serial port 0 has over or underflowed. This bit will remain set until it is reset by writing 1 to this bit. | R/W1C  | 0      |
| <code>valid0</code>      | 1 when the core-to-debugger queue for serial port 0 is not empty.   | R      | 0      |
| Continued on next page   |   |        |        |

|       |  |   |   |
|-------|--|---|---|
| full0 | 1 when the debugger-to-core queue for serial port 0 is full. | R | 0 |
|-------|--|---|---|

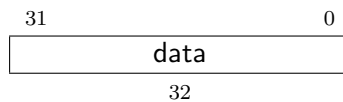
### C.1.2 Serial TX Data (sertx, at 0x35)

If [serialcount](#) is 0, this register is not present.

This register provides access to the write data queue of the serial port selected by [serial](#) in [sercs](#).

If the **error** bit is not set and the queue is not full, a write to this register adds the written data to the core-to-debugger queue. Otherwise the **error** bit is set and the write returns error.

A read to this register returns the last data written.



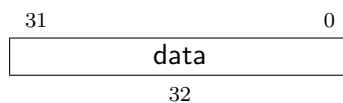
### C.1.3 Serial RX Data (serrx, at 0x36)

If [serialcount](#) is 0, this register is not present.

This register provides access to the read data queues of the serial port selected by [serial](#) in [sercs](#).

If the **error** bit is not set and the queue is not empty, a read from this register reads the oldest entry in the debugger-to-core queue, and removes that entry from the queue. Otherwise the **error** bit is set and the read returns error.

This entire register is read-only.



# Index

abits, [48](#)  
abstractauto, [24](#)  
abstractcs, [22](#)  
Access Register, [10](#)  
ackhavereset, [19](#)  
action, [42](#), [44](#)  
address, [27](#), [28](#), [48](#)  
allhalted, [18](#)  
allhavereset, [17](#)  
allnonexistent, [18](#)  
allresumeack, [18](#)  
allrunning, [18](#)  
allunavail, [18](#)  
anyhalted, [18](#)  
anyhavereset, [17](#)  
anynonexistent, [18](#)  
anyresumeack, [18](#)  
anyrunning, [18](#)  
anyunavail, [18](#)  
authbusy, [18](#)  
authdata, [25](#)  
authenticated, [18](#)  
autoexecdata, [24](#)  
autoexecprogbuf, [24](#)  
  
busy, [23](#)  
BYPASS, [49](#)  
  
cause, [34](#)  
chain, [42](#)  
cmderr, [23](#)  
cmdtype, [11](#), [12](#), [24](#)  
command, [23](#)  
control, [24](#)  
count, [43](#)  
  
data, [29](#), [39](#), [48](#)  
data0, [25](#)  
dataaccess, [21](#)  
dataaddr, [21](#)  
datacount, [23](#)  
  
datasize, [21](#)  
dcsr, [33](#)  
devtreeaddr0, [24](#)  
devtreevalid, [18](#)  
dmactive, [20](#)  
dmcontrol, [18](#)  
dmerr, [17](#)  
dmi, [48](#)  
dmihardreset, [47](#)  
dmireset, [47](#)  
dmistat, [48](#)  
dmode, [39](#)  
dmstatus, [15](#)  
dpc, [34](#)  
dscratch0, [35](#)  
dscratch1, [35](#)  
dtmcs, [47](#)  
  
ebreakm, [33](#)  
ebreaks, [33](#)  
ebreaku, [33](#)  
error0, [64](#)  
execute, [43](#)  
  
field, [2](#)  
full0, [65](#)  
  
h, [43](#), [44](#)  
haltreq, [19](#)  
haltsum, [21](#)  
hartinfo, [20](#)  
hartreset, [19](#)  
hartsel, [20](#)  
hasel, [19](#)  
hawindow, [22](#)  
hawindowssel, [22](#)  
  
icount, [43](#)  
IDCODE, [46](#)  
idle, [47](#)  
impebreak, [17](#)



load, [43](#)

m, [43](#), [44](#)

ManufId, [46](#)

maskmax, [40](#)

match, [42](#)

mcontrol, [39](#)

ndmreset, [20](#)

nscratch, [21](#)

op, [49](#)

PartNumber, [46](#)

postexec, [11](#)

priv, [36](#)

progbuf0, [25](#)

progbufsize, [22](#)

prv, [34](#), [36](#)

Quick Access, [12](#)

regno, [12](#)

resumereq, [19](#)

s, [43](#), [44](#)

sbaccess, [26](#)

sbaccess128, [27](#)

sbaccess16, [27](#)

sbaccess32, [27](#)

sbaccess64, [27](#)

sbaccess8, [27](#)

sbaddress0, [27](#)

sbaddress1, [27](#)

sbaddress2, [27](#)

sbasize, [27](#)

sbautoincrement, [26](#)

sbautoread, [26](#)

sbc, [25](#)

sbdata0, [28](#)

sbdata1, [29](#)

sbdata2, [29](#)

sbdata3, [29](#)

sberror, [26](#)

sbsingleread, [26](#)

select, [41](#)

sercs, [64](#)

serial, [64](#)

serialcount, [64](#)

serrx, [65](#)

sertx, [65](#)

shortname, [2](#)

size, [11](#)

step, [34](#)

stepie, [33](#)

stopcount, [33](#)

stoptime, [34](#)

store, [43](#)

tdata1, [38](#)

tdata2, [39](#)

tdata3, [39](#)

timing, [41](#)

transfer, [11](#)

tselect, [38](#)

type, [39](#)

u, [43](#), [44](#)

valid0, [64](#)

Version, [46](#)

version, [18](#), [48](#)

write, [11](#)

xdebugver, [33](#)



# Appendix D

## Change Log

| Revision | Date       | Author(s)   | Description  |
|----------|------------|-------------|--|
| ce1b359  | 2017-11-17 | Megan Wachs | Reset clarifications (#172)  |
| f49bf1d  | 2017-11-16 | Tim Newsome | Merge pull request #174 from riscv/context   |
| bac9a94  | 2017-11-16 | mwachs5     | Merge remote-tracking branch 'origin/0.13' into context                              |
| 852a70d  | 2017-11-16 | Megan Wachs | icount: remove warning (#173)  |
| 363348f  | 2017-11-16 | Tim Newsome | Explain cache coherency wrt to system bus access (#171)                              |
| 26ea898  | 2017-11-15 | Tim Newsome | Refer to ISA and priv docs.  |
| e803d67  | 2017-11-03 | Tim Newsome | Merge pull request #170 from riscv/index   |
| ffc8c62  | 2017-11-03 | Tim Newsome | Mention the index in "about this doc"  |
| a4257ef  | 2017-11-02 | Tim Newsome | Add an index to the document.  |
| f5f45a5  | 2017-10-30 | Megan Wachs | Add 'has reset' status and control (#168)  |
| e87fd2d  | 2017-10-26 | Tim Newsome | Merge pull request #158 from riscv/bits_not_signals                                  |
| dfe6a49  | 2017-10-25 | Megan Wachs | Merge branch '0.13' into bits_not_signals  |
| 46f3f54  | 2017-10-25 | Tim Newsome | Incorporate review feedback.   |
| 104247f  | 2017-10-24 | Megan Wachs | Update README.md   |
| 6dd5c80  | 2017-10-24 | Megan Wachs | Update README.md   |
| cb1a847  | 2017-10-24 | Megan Wachs | Add a note to the README about the built PDF   |
| 9e1fe79  | 2017-10-19 | Tim Newsome | Merge pull request #167 from riscv/include_pdf                                       |
| e00625f  | 2017-10-18 | Tim Newsome | Include pdf.   |
| 3ea45f1  | 2017-10-18 | Megan Wachs | Merge branch '0.13' into bits_not_signals  |
| c23e729  | 2017-10-18 | Tim Newsome | Clarify more.  |
| 6cab794  | 2017-10-12 | Tim Newsome | Merge pull request #162 from riscv/impebreak   |
| ecdaf0a  | 2017-10-11 | Megan Wachs | Merge branch '0.13' into impebreak   |
| 83f9faf  | 2017-10-11 | Tim Newsome | Clarify what <a href="#">impebreak</a> does.   |
| 58fdc0e  | 2017-10-11 | Megan Wachs | Merge pull request #164 from riscv/legend_on_fig                                     |
| 5f3933f  | 2017-10-11 | Tim Newsome | Merge branch 'impebreak' of github.com:riscv/riscv-debug-spec into impebreak         |
| 78082b5  | 2017-10-11 | Tim Newsome | Mention <a href="#">impebreak</a> in Program Buffer description.                     |
| 0378324  | 2017-10-11 | mwachs5     | Add legend and update some transitions on the Abstract Command State Machine diagram |

|         |            |             |  |
|---------|------------|-------------|--|
| a418a25 | 2017-10-11 | Megan Wachs | Merge branch '0.13' into impebreak   |
| d1c895a | 2017-10-11 | Megan Wachs | Merge pull request #161 from riscv/no_h_mode_fig   |
| f9981e8 | 2017-10-11 | Megan Wachs | Merge branch '0.13' into no_h_mode_fig   |
| ae30fc1 | 2017-10-11 | Megan Wachs | Merge pull request #163 from riscv/fix_build   |
| fa2b600 | 2017-10-11 | Megan Wachs | add missing period   |
| 0610630 | 2017-10-11 | Megan Wachs | Just do simple hmode -i dmode replacement  |
| 16e11f3 | 2017-10-11 | Tim Newsome | Remove hmode reference, to fix build.  |
| 84b9a6a | 2017-10-11 | Tim Newsome | Add <a href="#">impebreak</a> , to support of implicit ebreak.                                   |
| cc90b77 | 2017-10-11 | mwachs5     | Remove reference to 'H' mode from the figure   |
| cc6a9de | 2017-10-11 | Megan Wachs | Change old reference to 'hmode' to 'dmode'   |
| 4a9e017 | 2017-10-10 | Tim Newsome | Merge branch 'bits_not_signals' of<br>github.com:riscv/riscv-debug-spec into<br>bits_not_signals |
| ea2877d | 2017-10-10 | Tim Newsome | Move how-to-debug into the relevant section.   |
| d7b9f4c | 2017-10-06 | Megan Wachs | Merge branch '0.13' into bits_not_signals  |
| 48f437b | 2017-10-06 | Megan Wachs | Merge pull request #159 from riscv/unsup-<br>ported_access_size                                  |
| 24f0494 | 2017-10-06 | Tim Newsome | Merge branch '0.13' into unsupported_access_size   |
| 812686d | 2017-10-06 | Tim Newsome | Merge pull request #157 from riscv/reset   |
| 486ecc6 | 2017-10-05 | Tim Newsome | Refuse unsupported bus accesses.   |
| 6ca221d | 2017-10-05 | Tim Newsome | haltreq, resumereq, hartreset are per-hart bits  |
| c9cdf9e | 2017-10-05 | Tim Newsome | Merge branch '0.13' into reset   |
| 2be57af | 2017-10-04 | Tim Newsome | Merge pull request #128 from riscv/connector   |
| d16b390 | 2017-10-04 | Tim Newsome | Merge branch '0.13' into connector   |
| d4118ab | 2017-09-30 | Tim Newsome | ndmreset can't reset logic required to access DM.  |
| d569387 | 2017-09-29 | Tim Newsome | Merge pull request #154 from riscv/nikhil  |
| 2a47bd1 | 2017-09-29 | Tim Newsome | Merge branch '0.13' into nikhil  |
| 6269d9d | 2017-09-29 | Tim Newsome | Merge pull request #132 from riscv/progbufsize   |
| c6bd8d1 | 2017-09-29 | Tim Newsome | and -i or  |
| 58c2441 | 2017-09-29 | Tim Newsome | Mention <a href="#">stepie</a> in Single Step  |
| 94c5f78 | 2017-09-29 | Tim Newsome | Clarify ndmreset.  |
| 12810b4 | 2017-09-29 | Tim Newsome | Clarify that sbaddress is physical.  |
| 5862fdf | 2017-09-29 | Tim Newsome | Unify M mode and mprv comment.   |
| aea1bd5 | 2017-09-29 | Tim Newsome | Define behavior when haltreq and resumereq are set   |
| fe76d39 | 2017-09-28 | Megan Wachs | Merge branch '0.13' into progbufsize   |
| 146b348 | 2017-09-28 | Megan Wachs | remove superflous 'an'   |
| a5d16c4 | 2017-09-28 | Megan Wachs | remove superfluous 'a'   |
| 052a8ab | 2017-09-28 | Tim Newsome | Clarify that a debugger can lose hart control.   |
| cc52cff | 2017-09-28 | Tim Newsome | Add <a href="#">dmerr</a> .  |
| 25685eb | 2017-09-28 | Tim Newsome | Explain that bus master or progbuf is required.  |
| f75ee7d | 2017-09-28 | Tim Newsome | Clarify debugger can discover "almost" everything  |
| 71e6788 | 2017-09-27 | Tim Newsome | Remove description of manual stepping.   |
| 9aea347 | 2017-09-27 | Tim Newsome | Move Running/Single Step near Halting.   |
| 2090d9b | 2017-09-27 | Tim Newsome | data0 should be sbdata0 in this table.   |
| 5858cfe | 2017-09-27 | Tim Newsome | Clarify why <a href="#">priv</a> exists.   |
| bc3c2aa | 2017-09-27 | Tim Newsome | Mention where priv encoding comes from.  |
| ef77cc4 | 2017-09-27 | Tim Newsome | One more attempt to clarify DPC after single step.   |
| 80a288e | 2017-09-27 | Tim Newsome | Clarify instret not incrementing on ebreak.  |

|         |            |             |  |
|---------|------------|-------------|--|
| 4359b78 | 2017-09-27 | Tim Newsome | Merge pull request #152 from riscv/nikhil                          |
| c163d22 | 2017-09-20 | Tim Newsome | Remove ebreakh.  |
| 9971075 | 2017-09-20 | Tim Newsome | Clarify we're talking about privilege                              |
| 3f8e495 | 2017-09-20 | Tim Newsome | Clarify that we're talking about *implementation*                  |
| 3684854 | 2017-09-20 | Tim Newsome | Use steps environment in sbdata0.                                  |
| d4eda18 | 2017-09-20 | Tim Newsome | Explain that only sbdata0 has side effects.                        |
| ae781c6 | 2017-09-20 | Tim Newsome | Don't refer to internal system bus registers.                      |
| 875922e | 2017-09-20 | Tim Newsome | Explain sbdata0 being stale a bit more.                            |
| cd44fd5 | 2017-09-20 | Tim Newsome | Clarify autoread   |
| 194484b | 2017-09-20 | Tim Newsome | Clarify hawindow.  |
| 02f1aac | 2017-09-20 | Tim Newsome | Clarify that <a href="#">dataaddr</a> is relative to <b>zero</b> . |
| 0e9b6ae | 2017-09-20 | Tim Newsome | Clarify nonexistent vs unavailable.                                |
| b55ff41 | 2017-09-20 | Tim Newsome | Fix devtreevalid.  |
| 4325ef8 | 2017-09-20 | Megan Wachs | Merge branch '0.13' into progbufsize                               |
| 2eccb86 | 2017-09-20 | Tim Newsome | Explicitly state which registers are read-only.                    |
| 4af505c | 2017-09-20 | Tim Newsome | Show section numbers for registers.                                |
| cbd5573 | 2017-09-20 | Tim Newsome | Thank Nikhil   |
| 19c206f | 2017-09-20 | Tim Newsome | Clarify how to determine whether progbuf is RAM                    |
| 0651f7d | 2017-09-20 | Tim Newsome | Explain what happens if ebreak is missing.                         |
| e889dae | 2017-09-20 | Tim Newsome | Move figure of states into its own section.                        |
| cff7b80 | 2017-09-20 | Tim Newsome | Explain when <a href="#">transfer</a> might be used.               |
| 6b2ee61 | 2017-09-20 | Tim Newsome | Explain where <a href="#">size</a> encoding came from.             |
| 900d8ab | 2017-09-20 | Tim Newsome | Merge pull request #145 from riscv/nikhil                          |
| c9f3b73 | 2017-09-14 | Tim Newsome | Fix typo.  |
| 4b25400 | 2017-09-13 | Tim Newsome | Mention dpc in CSRs abstract register numbers.                     |
| c3ee426 | 2017-09-13 | Tim Newsome | Move abstract regno table closer to its reference.                 |
| 111b9a3 | 2017-09-13 | Tim Newsome | cycle -i operation   |
| 994afdc | 2017-09-13 | Tim Newsome | Account for multiple selected harts.                               |
| aa4a297 | 2017-09-13 | Tim Newsome | Halt Control -i Run Control  |
| e97c821 | 2017-09-13 | Tim Newsome | continuous -i contiguous   |
| 97f73ff | 2017-09-13 | Tim Newsome | Clarify ndmreset behavior.   |
| 6078220 | 2017-09-13 | Tim Newsome | Explain ndmreset   |
| a3d4f30 | 2017-09-13 | Tim Newsome | Describe 'halt region'   |
| 272b3d9 | 2017-09-13 | Tim Newsome | Clarify accessing unimplemented DM DMI regs                        |
| 3e91f1b | 2017-09-13 | Tim Newsome | Clarify either Prog Buf or Sys Bus Acc is required                 |
| e8a6145 | 2017-09-13 | Tim Newsome | Clarify CSR access; remove serial port                             |
| ce20766 | 2017-09-13 | Tim Newsome | Remove section referencing itself.                                 |
| 1195a61 | 2017-09-18 | Tim Newsome | Generate constants to be unsigned for clang.                       |
| ba200ab | 2017-08-18 | Megan Wachs | Merge branch '0.13' into progbufsize                               |
| 8967b0a | 2017-08-16 | Megan Wachs | Compressed instructions are c.foo, not foo.c                       |
| b5698a9 | 2017-08-16 | Megan Wachs | clarify progbufsize description                                    |
| d221bab | 2017-08-16 | Megan Wachs | Remove progbufsize enums from register description                 |
| d232d64 | 2017-08-16 | Megan Wachs | Merge pull request #134 from riscv/sw-examples-cleanup             |
| 0498102 | 2017-08-16 | Megan Wachs | appendix: Use standard assembly format for sw                      |
| 6e20373 | 2017-08-15 | Megan Wachs | Merge pull request #131 from riscv/devtree                         |
| 50ea40c | 2017-08-15 | Megan Wachs | Merge branch '0.13' into devtree                                   |
| 4e51a25 | 2017-08-10 | Tim Newsome | Merge pull request #130 from riscv/trigsign                        |

|         |            |             |   |
|---------|------------|-------------|---|
| 4456d99 | 2017-08-09 | Tim Newsome | Rename progsiz to progbufsize.  |
| 55d5b66 | 2017-08-09 | Tim Newsome | Clarify that trigger comparisons are unsigned.                                |
| 21e35ef | 2017-08-09 | Tim Newsome | Configuration String -i Device Tree   |
| dc52f28 | 2017-08-03 | Megan Wachs | Merge pull request #127 from riscv/cmdtype                                    |
| f044f45 | 2017-08-02 | Tim Newsome | Don't require a target to provide 25mA on VCC.                                |
| c883943 | 2017-08-02 | Tim Newsome | Add table of Abstract Command Types   |
| d6b8148 | 2017-08-02 | Tim Newsome | Merge pull request #123 from riscv/lists                                      |
| 71f5cb2 | 2017-08-02 | Tim Newsome | Merge branch '0.13' into lists  |
| b83af70 | 2017-08-02 | Megan Wachs | Merge pull request #125 from riscv/no_dmi_error                               |
| 2a41bd8 | 2017-08-02 | Megan Wachs | Merge branch '0.13' into no_dmi_error   |
| 9c73ce8 | 2017-08-02 | Megan Wachs | Merge pull request #111 from riscv/dpc  |
| a814400 | 2017-08-02 | Tim Newsome | Merge branch '0.13' into dpc  |
| 8bdc5cd | 2017-08-02 | Tim Newsome | Merge pull request #126 from riscv/build                                      |
| 985a3df | 2017-08-02 | Tim Newsome | Fix and speed up build.   |
| 95b9108 | 2017-08-02 | mwachs5     | DTM: Clarify that there are no cases when DMI would actually return an error. |
| 9c9e0c0 | 2017-08-02 | mwachs5     | SystemBus: No longer returns error. So DMI has no 'error' return code.        |
| ae1e9e4 | 2017-07-28 | Tim Newsome | Merge branch '0.13' into dpc  |
| 5ba18f9 | 2017-07-27 | Tim Newsome | Fix more typos.   |
| c6fef98 | 2017-07-26 | Tim Newsome | Merge pull request #122 from riscv/version                                    |
| dbc65bf | 2017-07-26 | Tim Newsome | Fix typos.  |
| bba0ad9 | 2017-07-26 | Tim Newsome | Tighten up introduction lists.  |
| e22d5eb | 2017-07-26 | Tim Newsome | Add version constants for "not compatible".                                   |
| c79038e | 2017-07-26 | Tim Newsome | Small clarification.  |
| 9df0411 | 2017-07-21 | Tim Newsome | Incorporate review feedback.  |
| d67419c | 2017-07-21 | Tim Newsome | Clarify dpc contents.   |
| c562898 | 2017-07-11 | Tim Newsome | Merge pull request #109 from riscv/ll   |
| 498cdf4 | 2017-07-11 | Megan Wachs | Merge branch '0.13' into ll   |
| 0e707f1 | 2017-07-11 | Megan Wachs | Merge pull request #105 from riscv/quick_access_errors                        |
| 2d34f65 | 2017-07-11 | Tim Newsome | Merge branch '0.13' into ll   |
| a56831c | 2017-07-11 | Megan Wachs | Merge branch '0.13' into quick_access_errors                                  |
| 65d596e | 2017-07-11 | Megan Wachs | Merge pull request #106 from riscv/error_halt_resume                          |
| 9f50c05 | 2017-07-11 | Tim Newsome | Use LL instead of L for 64-bit constant suffix.                               |
| 23fd24a | 2017-07-10 | Megan Wachs | Cleaning up whitespaces   |
| 102ba67 | 2017-07-10 | Megan Wachs | Merge branch '0.13' into error_halt_resume                                    |
| 1720505 | 2017-07-10 | Megan Wachs | Merge pull request #107 from riscv/csr_individuality                          |
| d67f6ef | 2017-07-10 | Megan Wachs | Merge branch '0.13' into csr_individuality                                    |
| c1e61b0 | 2017-07-10 | Megan Wachs | Merge pull request #108 from riscv/dcsr_causes                                |
| c5ab04c | 2017-07-10 | Megan Wachs | Update abstract_commands.xml  |
| 6e8cdf1 | 2017-07-10 | Megan Wachs | Update abstract_commands.xml  |
| cf6e3f2 | 2017-07-10 | Megan Wachs | clarify DCSR.cause  |
| 79ffbb9 | 2017-07-10 | Megan Wachs | Clarify implications of CSR read, write, halt                                 |
| 013e191 | 2017-07-10 | Megan Wachs | Clarify when you would get error halt/resume                                  |
| 231e457 | 2017-07-10 | Megan Wachs | Quick Access error clarification  |

|         |            |             |   |
|---------|------------|-------------|---|
| 7c760b0 | 2017-07-03 | Megan Wachs | Merge pull request #104 from riscv/serial.to_appendix                                 |
| c54c2f2 | 2017-07-03 | mwachs5     | serial: add the XML file, not the TEX file  |
| ac77477 | 2017-07-03 | mwachs5     | serial: Fix compile errors after moving serial port to appendix                       |
| 6defcb8 | 2017-07-03 | mwachs5     | serial: Move serial ports out of main spec and into Future Work appendix              |
| 3541152 | 2017-07-03 | Megan Wachs | Merge pull request #102 from riscv/remove_trace                                       |
| a28f639 | 2017-06-30 | mwachs5     | remove trace dependencies from Makefile   |
| 52a122b | 2017-06-30 | mwachs5     | remove trace section  |
| d9e166b | 2017-06-30 | mwachs5     | remove trace registers  |
| 7caf4e5 | 2017-06-30 | mwachs5     | remove trace appendix   |
| aff0c16 | 2017-06-30 | Megan Wachs | Merge pull request #82 from riscv/intdisable  |
| 4688988 | 2017-06-29 | mwachs5     | DCSR: define a 'stepie' bit which may be hard-wired to 0.                             |
| 443b3fe | 2017-06-29 | mwachs5     | Merge remote-tracking branch 'origin/0.13' into int-disable                           |
| 497ed95 | 2017-06-29 | Megan Wachs | Merge pull request #96 from riscv/jtagdtm_non_appendix                                |
| f1488c4 | 2017-06-29 | Megan Wachs | Merge branch '0.13' into jtagdtm_non_appendix   |
| 920ec9a | 2017-06-13 | Megan Wachs | Merge pull request #95 from riscv/remove_spontaneous                                  |
| 67fa7b0 | 2017-06-13 | Megan Wachs | Merge branch '0.13' into remove_spontaneous   |
| 78eb65e | 2017-06-13 | Megan Wachs | Merge pull request #94 from riscv/anynonexistent                                      |
| d97b296 | 2017-06-13 | Megan Wachs | Merge branch '0.13' into anynonexistent   |
| 61c6d30 | 2017-06-13 | Megan Wachs | Merge pull request #93 from riscv/define-dret-again                                   |
| 389ee69 | 2017-06-13 | Megan Wachs | Merge branch '0.13' into define-dret-again  |
| 421dcf2 | 2017-06-13 | Megan Wachs | Merge pull request #97 from riscv/implementation_deets                                |
| 9a0492c | 2017-06-13 | Megan Wachs | Add missing period and some other small text edits                                    |
| 13ccdbf | 2017-06-13 | Megan Wachs | fix typo in ProgBuf register macro  |
| b01f989 | 2017-06-13 | mwachs5     | implementations: be a bit more concrete about the one example implementation we have. |
| a7b5f83 | 2017-06-13 | mwachs5     | jtagdtm: Move it out of the appendix as it is really part of the specification        |
| 87aceb0 | 2017-06-13 | Megan Wachs | remove "spontaneous"  |
| 50b9950 | 2017-06-13 | Megan Wachs | Forward reference for anynonexistent  |
| adea3e2 | 2017-06-13 | Megan Wachs | More clarifications on dret   |
| 1b8dd0e | 2017-06-13 | Megan Wachs | Define DRET instruction   |
| b4f1f43 | 2017-06-08 | Tim Newsome | Merge pull request #79 from riscv/cleanups  |
| 09c7f6e | 2017-06-08 | mwachs5     | Merge remote-tracking branch 'origin/0.13' into cleanups                              |
| 617da4c | 2017-06-08 | Megan Wachs | Update description of R/W1C   |
| de2c56b | 2017-06-08 | Megan Wachs | Clarify that DCSR is also not updated on ebreak                                       |
| efa615d | 2017-06-07 | Tim Newsome | Increase xdebugver field size to 4 bits. (#92)  |
| a0e147a | 2017-06-07 | Tim Newsome | Address some review comments.   |
| c1b3e54 | 2017-06-07 | Megan Wachs | Merge pull request #91 from riscv/ndmreset  |
| 5c7c1bb | 2017-06-07 | Tim Newsome | Merge branch '0.13' into cleanups   |

|         |            |                                  |  |
|---------|------------|----------------------------------|--|
| 72bb874 | 2017-06-06 | Megan Wachs                      | Merge branch '0.13' into ndmreset  |
| 1fbbe6e | 2017-06-06 | Megan Wachs                      | Merge pull request #90 from riscv/dpc_clarifications   |
| 89ffe50 | 2017-06-06 | mwachs5                          | NDMRESET: Clarify what it may and may not do   |
| 1932da0 | 2017-06-06 | mwachs5                          | DPC: Clarifications on its meaning   |
| 03bcafe | 2017-06-06 | Megan Wachs                      | Merge pull request #89 from riscv/datacount  |
| 6470fdb | 2017-06-06 | mwachs5                          | ABSTRACTCS: Correct inconsistency on the number of data words.                                     |
| 1f4a1fe | 2017-06-06 | Megan Wachs                      | Merge pull request #88 from riscv/W0_corrections   |
| 3ca82b4 | 2017-06-06 | Megan Wachs                      | More corrections for R vs R/W1C on SERCS   |
| 9705fb8 | 2017-06-06 | Megan Wachs                      | Correct a bunch of W0 registers  |
| 1058690 | 2017-06-05 | Megan Wachs                      | Merge branch '0.13' into intdisable  |
| 7531c41 | 2017-06-05 | Megan Wachs                      | Merge pull request #80 from riscv/issue76  |
| 1347371 | 2017-06-05 | Tim Newsome                      | Add intdisable to dcsr.  |
| 850bd87 | 2017-06-05 | Megan Wachs                      | Merge branch '0.13' into issue76   |
| 43307eb | 2017-06-05 | Megan Wachs                      | Merge pull request #81 from riscv/issue63  |
| 989c60d | 2017-06-05 | Tim Newsome                      | Fix language. We can only halt harts, not cores.   |
| 517a08b | 2017-06-05 | Tim Newsome                      | Incorporate review feedback.   |
| 802be28 | 2017-06-05 | Tim Newsome                      | Clarify/fix Quick Access example.  |
| dbcaec8 | 2017-06-02 | Tim Newsome                      | Merge branch '0.13' into cleanups  |
| b8cc523 | 2017-06-02 | Tim Newsome                      | Add included tex files as dependencies. (#78)  |
| d0a5959 | 2017-06-02 | Tim Newsome                      | Merge pull request #77 from riscv/pageno   |
| 15f864a | 2017-06-01 | Tim Newsome                      | Language cleanups, consistency and typo fixes.   |
| 4ecae86 | 2017-06-01 | Tim Newsome                      | Add page numbers to list-of-register tables.   |
| 59b3e4a | 2017-05-19 | Megan Wachs                      | Setting up a Travis regression to check for build errors (#72)                                     |
| 124bf44 | 2017-05-17 | mwachs5                          | Debug Module: CMDERR is Write-1-to clear, not R/W0   |
| bb6c7f0 | 2017-05-17 | mwachs5                          | SW Registers file should be XML, not TEX   |
| d360358 | 2017-05-10 | Megan Wachs<br>(Temporary Acct.) | Remove virtual register from core_registers.xml  |
| bfc64fb | 2017-05-10 | Megan Wachs<br>(Temporary Acct.) | Add missing sw_registers.tex file  |
| 0512f5d | 2017-05-06 | mwachs5                          | Move virtual 'prv' register to a separate section to make it more clear it is not a real register. |
| 6b3c9d7 | 2017-05-06 | mwachs5                          | Clarify haltreq/resumereq/resumack   |
| 0a487eb | 2017-04-26 | mwachs5                          | jtag: Change specified JTAG pinout from Coretex to AVR, to provide for TRSTn option.               |
| 93cdfaf | 2017-04-26 | mwachs5                          | DM : Clarify that DATA/PROGBUF can't be written while busy.  |
| ef98f23 | 2017-04-19 | mwachs5                          | jtag: Make it clear that a NOP is really a NOP.  |
| a6f8efa | 2017-04-17 | mwachs5                          | single_step: Exceptions count as the 'step' completion.  |
| bf11e9e | 2017-04-17 | mwachs5                          | resumeack: fix some LaTeX cross references   |
| 4afa081 | 2017-04-11 | mwachs5                          | halt/resumereq: Clarify what setting them to 0 or 1 does   |
| 297a39b | 2017-04-06 | mwachs5                          | fix chisel build   |



|         |            |         |  |
|---------|------------|---------|--|
| 082c499 | 2017-04-06 | mwachs5 | Rename resumed to resumeack, and add more text about what these bits mean.   |
| 909d617 | 2017-04-06 | mwachs5 | Correct some cross references after removing all the multiply listed registers   |
| dd09914 | 2017-04-06 | mwachs5 | Add 'resumedall' and 'resumedany' bits to avoid race condition on about to resume and just halted  |
| feb88fc | 2017-04-05 | mwachs5 | JTAG DTM: Clarify that leading bits are 0 for more than 5-bit IR   |
| 75b96ea | 2017-04-04 | mwachs5 | use renamed dm_registers file  |
| 9f3ec7e | 2017-04-04 | mwachs5 | debugger_implementation: remove some old TODO and commentary.  |
| 45dd5b5 | 2017-04-04 | mwachs5 | Don't list out every single DM register for those that are just indexed versions   |
| b8b3aa2 | 2017-04-04 | mwachs5 | remove core-side register definitions from Debug Module. Rename dm1 to dm  |
| d979a13 | 2017-04-04 | mwachs5 | remove core-side serial port specification, as these should look like implementation-specific devices with appropriate drivers.  |
| b56870b | 2017-04-04 | mwachs5 | Remove the wording about 'debug exception', as it is called breakpoint exception in the RISC-V Spec.   |
| 1e9347d | 2017-04-03 | mwachs5 | Add description of hasel   |
| 0dda84d | 2017-04-03 | mwachs5 | JTAG DTM: Clean up TAP register descriptions   |
| 82ccde5 | 2017-04-03 | mwachs5 | JTAG DTM: Add a hard DMI bit which cancels the outstanding DMI transaction   |
| bd2a3d1 | 2017-04-03 | mwachs5 | remove preexec   |
| 02c733a | 2017-04-03 | mwachs5 | remove preexec from Abstract State diagram.  |
| 1e271d6 | 2017-04-03 | mwachs5 | Update Debugger implementation for DMI register access, and fix tex compile issues.  |
| 155dda4 | 2017-04-03 | mwachs5 | Rewrite HW Implementation examples to describe a pure abstract command approach, and to not rely on harts executing every instruction which is fetched from the Debug Module |
| 556c2be | 2017-04-03 | mwachs5 | minor wording edits about RISC-V core registers  |
| 523c64a | 2017-04-03 | mwachs5 | Edits to the Debug Module section.   |
| b9a371f | 2017-04-03 | mwachs5 | add missing trace.tex file.  |
| 58b2396 | 2017-04-03 | mwachs5 | Re-order the JTAG DTM Sections   |
| a8827e2 | 2017-04-03 | mwachs5 | Edits to the System Overview.  |
| c5417ce | 2017-04-03 | mwachs5 | add more sections as separate files.   |
| 287d5c6 | 2017-04-03 | mwachs5 | moving more files to separate tex files.   |
| 9e873f4 | 2017-04-03 | mwachs5 | move trigger info into separate file.  |
| 2c89a86 | 2017-04-03 | mwachs5 | move risc-v core debug info into separate file.  |
| e676491 | 2017-04-03 | mwachs5 | Move System Overview to separate file  |
| 03df6ee | 2017-04-03 | mwachs5 | Move Debug Module description to a separate file.  |
| 5faa430 | 2017-04-03 | mwachs5 | add back in JTAG DTM in appendix   |
| 7b28b11 | 2017-04-03 | mwachs5 | Move jtag DTM to appendix. Move some text to commentary.   |
| cc183ba | 2017-04-03 | mwachs5 | move introduction to a separate file. Comment out reading order.   |

|         |            |              |  |
|---------|------------|--------------|--|
| 2c83830 | 2017-04-03 | mwachs5      | Merge remote-tracking branch 'origin/0.13' into 0.13   |
| e3cf6ab | 2017-04-03 | Megan Wachs  | Merge pull request #18 from riscv/intro_edits  |
| 60c5a1c | 2017-04-03 | Megan Wachs  | Merge branch '0.13' into intro_edits   |
| f727d14 | 2017-04-03 | mwachs5      | Use Chapters vs Sections. Needs reorganization.  |
| 815951d | 2017-04-03 | mwachs5      | Formatting updates. Make this look more like the RISC-V specs. Need to use chapter vs. section   |
| 69ffaf8 | 2017-03-31 | mwachs5      | Move XML files into a subdirectory.  |
| b276384 | 2017-03-31 | mwachs5      | Remove debug_rom.S   |
| 112bbac | 2017-03-31 | mwachs5      | figures: reorganize the figures into directories.  |
| 2d05746 | 2017-03-31 | Megan Wachs  | Merge pull request #50 from riscv/add_license  |
| 1e5c068 | 2017-03-27 | Megan Wachs  | Add LICENSE  |
| 0e2d08a | 2017-03-22 | Megan Wachs  | Merge pull request #47 from poweihuang17/0.13  |
| fc17730 | 2017-03-22 | Po-wei Huang | Change some halt mode into debug mode.   |
| 8ccf029 | 2017-03-22 | Po-wei Huang | All halt mode changed to debug mode to synchronize with the priv spec.   |
| f143d9e | 2017-03-21 | mwachs5      | Correct duplicated progbuf register names  |
| 0797ec1 | 2017-03-17 | mwachs5      | autoexec: make autoexec bits match the number of data words there really are.  |
| 8e76d93 | 2017-03-17 | mwachs5      | dm1_registers: move a few more things around. Reduce abstract data words back to 12.   |
| f8bf292 | 2017-03-17 | mwachs5      | dm1_registers: resolve some address conflicts and inconsistencies  |
| a74dff9 | 2017-03-17 | mwachs5      | access_register: some small bit changes  |
| 2e6b0ca | 2017-03-15 | mwachs5      | config string: Fix LaTeX compile errors.   |
| f83260a | 2017-03-10 | mwachs5      | Abstract Commands: clarify that 32-bit reads should always work. This allows reading MISA.   |
| 6f9347a | 2017-03-10 | mwachs5      | Config String: change the Abstract Command to DMI registers. Allow the same registers to be used for unspecified identifier information. |
| 4ea10ff | 2017-03-10 | mwachs5      | abstract: Make autoexec apply to all data and progbuf words. Make a separate register which is optional.                                 |
| 5008436 | 2017-03-10 | mwachs5      | abstract: Allow up to 16 progbuf and/or data words. Inform debugger about dscratch registers available for its use.                      |
| aaa13e5 | 2017-03-06 | mwachs5      | Command: use the name 'cmdtype' not 'type' to allow easier auto-generation of Scala code.  |
| e9bb72c | 2017-03-06 | mwachs5      | Hart Array: Add registers for hart array.  |
| 5d17a35 | 2017-03-06 | mwachs5      | DM: Move addresses around for better separation of functionalities in HW   |
| 25ccaa8 | 2017-03-06 | mwachs5      | CONTROL: Rename control and status registers to ___CS for consistency and to accurately reflect their functionality.                     |
| 45cf6c2 | 2017-03-06 | mwachs5      | Errors: fix up the bit assignments in SERSTATUS with the addition of error bit.  |
| 38cb5a0 | 2017-03-06 | mwachs5      | Errors: Make errors write-1-to-clear.  |
| b436d77 | 2017-03-03 | mwachs5      | triggers: Clarify that matches are against virtual addresses.  |

|         |            |               |   |
|---------|------------|---------------|---|
| 793bb85 | 2017-03-03 | mwachs5       | triggers: Add suggested timings for best user experience.                             |
| 2669866 | 2017-03-03 | mwachs5       | stoptime/stopcycle: Make their functionality match their name. Allow any reset value. |
| c85a1cf | 2017-03-01 | mwachs5       | config_string: Simplify the Config String Address abstract command.                   |
| a303a6b | 2017-03-02 | Megan Wachs   | Update README.md  |
| 1951ae3 | 2017-03-01 | Megan Wachs   | Merge pull request #35 from sifive/generate_chisel                                    |
| 2e2dc28 | 2017-03-01 | Megan Wachs   | Merge pull request #34 from sifive/serial_addr  |
| c087c34 | 2017-03-01 | mwachs5       | Merge remote-tracking branch 'origin/0.13' into generate_chisel                       |
| 92a4923 | 2017-03-01 | mwachs5       | serial: tweak addresses.  |
| b09f460 | 2017-03-01 | mwachs5       | serial: tweak addresses.  |
| 6477837 | 2017-03-01 | mwachs5       | chisel: tweaks to class names.  |
| be83e3e | 2017-02-28 | Tim Newsome   | Clarify stoptime, stopcycle.  |
| 7f94662 | 2017-02-27 | mwachs5       | Merge remote-tracking branch 'origin/0.13' into generate_chisel                       |
| c17c17c | 2017-02-27 | Tim Newsome   | Abstract command that returns config string addr.                                     |
| 096dfbc | 2017-02-27 | Tim Newsome   | Acknowledge Alex.   |
| c0253ab | 2017-02-24 | Tim Newsome   | Explain tdata1 type a bit more.   |
| e43ac2e | 2017-02-24 | Tim Newsome   | Clarify how to enumerate triggers again.  |
| c6e3e20 | 2017-02-23 | Tim Newsome   | Revert previous commit.   |
| ef770bf | 2017-02-23 | Tim Newsome   | mcontrol and icount mask tdata2, not tdata1.  |
| 27806f2 | 2017-02-23 | mwachs5       | rename 'type' to 'cmdtype' purely so my auto-generation scripts work.                 |
| e46798d | 2017-02-22 | mwachs5       | Add Abstract Commands to automatic chisel   |
| b3bb939 | 2017-02-21 | mwachs5       | Generate Chisel headers as well for Debug Module.                                     |
| 3d5b6f6 | 2017-02-22 | Tim Newsome   | Merge pull request #31 from sifive/abstract_command_types                             |
| c9db98c | 2017-02-22 | Tim Newsome   | Simplify description of op statuses.  |
| bda39cc | 2017-02-22 | mwachs5       | Add explicit type field to Abstract Command.  |
| 34ff1d8 | 2017-02-22 | Tim Newsome   | Merge pull request #30 from sifive/-more_ibuf_progbuf                                 |
| f83a1ca | 2017-02-22 | mwachs5       | Finish up replacement of ibuf->progbuf  |
| ddde0a2 | 2017-02-22 | Tim Newsome   | Merge pull request #28 from sifive/inst_supply_vs_progbuf                             |
| 9666e51 | 2017-02-22 | mwachs5       | IBUF->PROGBUF   |
| 5308ecd | 2017-02-22 | mwachs5       | Remove last references to "Instruction Supply"  |
| f6ebde9 | 2017-02-22 | Tim Newsome   | Move authentication to a serial protocol.   |
| 0f079c8 | 2017-02-22 | Tim Newsome   | Reserve bit for per-hart reset.   |
| f2c93ac | 2017-02-22 | Tim Newsome   | Clarify that dmactive resets authentication.  |
| 59154ac | 2017-02-22 | Tim Newsome   | Merge pull request #27 from asb/clarify_reset   |
| f5e7b1c | 2017-02-22 | Alex Bradbury | Clarify that the halt state of all harts is maintained through reset                  |
| 3dfe8fd | 2017-02-22 | Tim Newsome   | More Debug Mode -> Halt Mode.   |
| d29fc1f | 2017-02-22 | Tim Newsome   | Debug Mode -> Halt Mode   |
| 55d6030 | 2017-02-21 | Tim Newsome   | Generate debug_defines.h as part of normal make                                       |
| b0e6a7f | 2017-02-21 | Tim Newsome   | Minor clarifications.   |

|         |            |             |   |
|---------|------------|-------------|---|
| 0f9885c | 2017-02-20 | Tim Newsome | Various clarifications.   |
| e443ab9 | 2017-02-15 | Tim Newsome | Merge pull request #25 from sifive/ctrl_status  |
| 3b08e90 | 2017-02-15 | Tim Newsome | Merge pull request #24 from sifive/sm_diagram_resumereq   |
| 0802d5a | 2017-02-15 | mwachs5     | Use consistent 'Control and Status' naming for CS registers.  |
| 5accc7d | 2017-02-15 | Tim Newsome | Change all the "other" JTAG IRs to just reserved.   |
| bcbd7da | 2017-02-15 | mwachs5     | sm_diagram: Show using resumereq bit to resume.   |
| 18f6e55 | 2017-02-14 | Tim Newsome | Introduce resumereq command, similar to haltreq.  |
| fb40538 | 2017-02-14 | Tim Newsome | Merge pull request #22 from sifive/sb_errors  |
| 4b62c40 | 2017-02-14 | mwachs5     | SystemBus: Clean up some formatting and error specification notes.  |
| 0f346e4 | 2017-02-14 | Tim Newsome | Merge pull request #21 from sifive/sm_for_quick_access  |
| bc97723 | 2017-02-14 | mwachs5     | quick-access: Update SM Diagram for Quick Access  |
| d27066e | 2017-02-14 | Tim Newsome | Clarify haltreq bit.  |
| 6f8ec43 | 2017-02-14 | Tim Newsome | Always generate long constants when required.   |
| c6ac6bc | 2017-02-13 | Tim Newsome | Include field descriptions in C header file.  |
| b849213 | 2017-02-13 | Tim Newsome | Fix the build.  |
| c82c62e | 2017-02-12 | Tim Newsome | Merge pull request #20 from sifive/jtag_ir_minimum  |
| 1cf8033 | 2017-02-12 | mwachs5     | jtag: More clarifications   |
| 6203bd6 | 2017-02-12 | Megan Wachs | Update requirements- W GPRs Required  |
| f2b43a7 | 2017-02-12 | Megan Wachs | Remove double 'the'   |
| 2c64ef1 | 2017-02-12 | Megan Wachs | Remove comma  |
| f84abce | 2017-02-12 | Megan Wachs | Whitespace edits and address come comments  |
| 7246b44 | 2017-02-12 | Tim Newsome | Merge pull request #19 from sifive/jtag_dtm_edits   |
| 23c2648 | 2017-02-11 | mwachs5     | jtag_dtm: ask for clarification on TAP sharing.   |
| 7020d23 | 2017-02-11 | mwachs5     | jtag_dtm: Clarifications, DBUS- $\mathcal{I}$ DMI   |
| 292d49c | 2017-02-11 | Megan Wachs | fix indentation   |
| 55ef8d6 | 2017-02-11 | Tim Newsome | Merge pull request #17 from sifive/prog_buffer_size   |
| b879b86 | 2017-02-11 | Megan Wachs | Add missing period  |
| bbe0521 | 2017-02-11 | mwachs5     | Make comments on program buffer size match the address map.   |
| 4ceaa37 | 2017-02-11 | mwachs5     | Flesh out and edit the introduction/background Add a description of use cases this spec has in mind, and what it doesn't cover. |
| cbf89d6 | 2017-02-11 | Tim Newsome | Rewrite Quick Access.   |
| 9115db1 | 2017-02-10 | Tim Newsome | Merge pull request #16 from sifive/reduce_prog_buffer_size  |
| 170bff1 | 2017-02-10 | Megan Wachs | Allow size 4 for the program buffer   |
| 9d46077 | 2017-02-10 | Tim Newsome | Merge pull request #15 from sifive/dmactive   |
| c911e6e | 2017-02-10 | Tim Newsome | Clarify use of dmactive.  |
| 2ca296f | 2017-02-09 | Tim Newsome | Reserve command register space for custom use.  |
| e49666e | 2017-02-09 | Tim Newsome | Clarify hart index change per Megan's comments.   |
| 84865e9 | 2017-02-09 | Tim Newsome | Add header prefix for abstract commands.  |
| 2434f4f | 2017-02-09 | Tim Newsome | Select harts by index instead of hart ID.   |
| 7bf112a | 2017-02-09 | Tim Newsome | Generate correct headers for $\mathcal{I}$ 32-bit registers.  |
| 7f0f09a | 2017-02-08 | Tim Newsome | Reset dbus status to "failure" to avoid confusion.  |

|         |            |             |   |
|---------|------------|-------------|---|
| 7b1803e | 2017-02-08 | Tim Newsome | Merge pull request #13 from sifive/arg0_clarification                                 |
| 8b1c6f0 | 2017-02-08 | Megan Wachs | Fix line wrap issue   |
| 345c33f | 2017-02-08 | Megan Wachs | Call out "arg0" specifically.   |
| 9f080f5 | 2017-02-08 | Megan Wachs | Clarify "arguments" to commands   |
| 259badd | 2017-02-08 | Tim Newsome | Make haltsum/halt registers mandatory.  |
| eb0f1d3 | 2017-02-07 | Tim Newsome | Allow for early abstract command failures.  |
| bb49bd1 | 2017-02-07 | Tim Newsome | Clarify error handling a little.  |
| 3fc0a97 | 2017-02-07 | Tim Newsome | Explain when abstract data regs may be clobbered.                                     |
| c37167e | 2017-02-07 | Tim Newsome | Fix old language in description of halt registers.                                    |
| 6943c96 | 2017-02-07 | Tim Newsome | Generate more useful C header files from reg defs                                     |
| d7a8045 | 2017-02-06 | Tim Newsome | Merge pull request #11 from sifive/sm_diagram   |
| 8bef40e | 2017-02-05 | mwachs5     | Merge remote-tracking branch 'origin/0.13' into sm_diagram                            |
| 98639df | 2017-02-05 | mwachs5     | Include the SM Diagram as a figure. Also some minor capitalization fixes.             |
| a95e4c3 | 2017-02-05 | mwachs5     | Update State Machine diagram to show uncertainty of halt bit during auto halt/resume. |
| ba76744 | 2017-02-05 | Tim Newsome | Combine loabits and hiabits.  |
| 02b1d92 | 2017-02-05 | Tim Newsome | DMI can get away with just 6 address bits.  |
| 35d6e33 | 2017-02-05 | mwachs5     | Update State machine diagram to show BUSY without HALTED                              |
| f511b05 | 2017-02-04 | Tim Newsome | Clarify command busy bit.   |
| a8e5ae7 | 2017-02-03 | mwachs5     | Merge remote-tracking branch 'origin/0.13' into sm_diagram                            |
| d0f8961 | 2017-02-03 | mwachs5     | Update figures  |
| e18a68d | 2017-02-03 | Tim Newsome | Clarify prehalt/postresume failure.   |
| ac3e2a9 | 2017-02-02 | Tim Newsome | Clarify abstract command failure behavior.  |
| ce4baee | 2017-02-02 | Tim Newsome | Add Quick Access section.   |
| 0490377 | 2017-02-02 | Tim Newsome | Add prehalt and postresume to reg command.  |
| 67515bd | 2017-02-02 | Tim Newsome | Deal with a few minor TODOs.  |
| 96456fc | 2017-02-02 | Tim Newsome | Turn register names into links.   |
| 317cd98 | 2017-02-02 | Tim Newsome | Explain what register access is required.   |
| f3ad2f2 | 2017-02-01 | Tim Newsome | Revert Plain Exception implementation to be simple                                    |
| a0ad281 | 2017-02-01 | Tim Newsome | execb -i preexec, execa -i postexec   |
| 1d4a2c3 | 2017-02-01 | Tim Newsome | Limit Program Buffer sizes to 0, 1, 8.  |
| cc40815 | 2017-02-01 | Tim Newsome | Incorporate Po-wei's feedback.  |
| c8b45d6 | 2017-02-01 | Tim Newsome | Clarify how all autoexec bits work.   |
| dbb1deb | 2017-02-01 | Tim Newsome | Remove stale TODO.  |
| c5f8f59 | 2017-02-01 | Tim Newsome | Explain why cmderr inhibits starting new commands.                                    |
| 5c69194 | 2017-02-01 | Tim Newsome | Fix editing error.  |
| 50f7c48 | 2017-02-01 | Tim Newsome | Remove empty hart info register.  |
| 781c68e | 2017-02-01 | Megan Wachs | Update README.md  |
| f46b32e | 2017-02-01 | mwachs5     | Add a diagram of Abstract Command flow.   |
| 633bd63 | 2017-02-01 | Tim Newsome | Move Reading Order into About This Document   |
| 51ec4d1 | 2017-02-01 | Tim Newsome | Add reading order section.  |
| 03d20ad | 2017-02-01 | Tim Newsome | autoexec0 applies to data0, not inst0.  |
| c302353 | 2017-01-31 | Tim Newsome | Don't rely on hart fetching instructions once.  |
| 2558c25 | 2017-01-31 | Tim Newsome | Change how exceptions in Halt Mode are handled.                                       |

|         |            |             |  |
|---------|------------|-------------|--|
| a36ddce | 2017-01-31 | Tim Newsome | Add size to abstract register command.   |
| 64de458 | 2017-01-31 | Tim Newsome | Detail bus master reads.   |
| c08486f | 2017-01-31 | Megan Wachs | reset: Add some comments (#5)  |
| 1558049 | 2017-01-30 | Tim Newsome | Automate Change Log.   |
| 51525a4 | 2017-01-29 | Tim Newsome | Update System Overview   |
| 7d39ac0 | 2017-01-29 | Tim Newsome | Update Supported Features.   |
| 9e7cbea | 2017-01-29 | Tim Newsome | Update RISC-V Core section.  |
| 515188d | 2017-01-29 | Tim Newsome | Update Hardware Implementations section.   |
| 4b19ed8 | 2017-01-29 | mwachs5     | system_bus: be consistent and always call it 'System Bus'. Even if some dislike the name, we should be consistent and clear in the spec. |
| 9cce3d  | 2017-01-29 | Tim Newsome | Fleshed out some debugger implementation.  |
| 04b9176 | 2017-01-28 | Tim Newsome | Rename debug exception to breakpoint exception.  |
| 5ac4ea1 | 2017-01-27 | Tim Newsome | WIP on big update on instruction supply.   |
| 2d9c3e2 | 2017-01-27 | Tim Newsome | Reorganize dm registers.   |
| de50ba8 | 2017-01-27 | Tim Newsome | Abstract command support is already addressed.   |
| 27cb0da | 2017-01-26 | Tim Newsome | Merge pull request #4 from sifive/access_renames   |
| 5085046 | 2017-01-26 | mwachs5     | Rename registers and fields like 'access' that were confusingly the same name.   |
| 10bbf6f | 2017-01-26 | Tim Newsome | Fix #2: DM address space table   |
| a05c582 | 2017-01-26 | Tim Newsome | Add debugger inspection as a feature.  |
| 4062681 | 2017-01-24 | Tim Newsome | Add publish target.  |
| 5c8bb83 | 2017-01-24 | Tim Newsome | Clarify use of data registers.   |
| 1504da6 | 2017-01-24 | Tim Newsome | Replace manual date with automatic git hash/date.  |
| 997f2a0 | 2017-01-23 | Tim Newsome | Deal with unsupported abstract commands.   |
| cb6f2b8 | 2017-01-23 | Tim Newsome | ReNUMBER registers to prevent duplicates.  |
| 8b4db96 | 2017-01-23 | Tim Newsome | Don't print out addresses if they're not provided.   |
| b00cd21 | 2017-01-23 | Tim Newsome | Add an abstract command.   |
| 675b556 | 2017-01-23 | Tim Newsome | Reorganize DM bits into functional group regs.   |
| 5fc7512 | 2017-01-23 | Tim Newsome | Remove bits 33:32 from sbdata[23].   |
| ceb5d66 | 2017-01-20 | Tim Newsome | Starting point for a comprehensive spec  |