

RISC-V External Debug Support

Version 0.14.0-DRAFT

034d0d6beac2a010315a755220bc88ad4690a660

Editors:

Ernie Edgar <ernie.edgar@sifive.com >, SiFive, Inc.

Tim Newsome <tim@sifive.com>, SiFive, Inc.

Tue Apr 30 10:29:54 2019 -0700

Contributors to all versions of the spec in alphabetical order (please contact editors to suggest corrections): Bruce Ableidinger, Krste Asanović, Peter Ashenden, Allen Baum, Mark Beal, Alex Bradbury, Chuanhua Chang, Zhong-Ho Chen, Monte Dalrymple, Vyacheslav Dyachenko, Peter Egold, Markus Goehrlé, Robert Golla, John Hauser, Richard Herveille, Yung-ching Hsiao, Po-wei Huang, Scott Johnson, Grigorios Magklis, Jean-Luc Nagel, Aram Nahidipour, Rishiyur Nikhil, Gajinder Panesar, Deepak Panwar, Antony Pavlov, Klaus Kruse Pedersen, Ken Pettit, Joe Rahmeh, Gavin Stark, Wesley Terpstra, Megan Wachs, Jan-Willem van de Waardt, Stefan Wallentowitz, Ray Van De Walker, Andrew Waterman, Andy Wright, and Bryan Wyatt.

# 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.**

# Contents

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

|         |  |    |
|---------|--|----|
| 3.5     | Run Control . . . . .  | 10 |
| 3.6     | Halt Groups and External Triggers . . . . .                                    | 11 |
| 3.7     | Abstract Commands . . . . .  | 11 |
| 3.7.1   | Abstract Command Listing . . . . .   | 12 |
| 3.7.1.1 | Access Register . . . . .  | 13 |
| 3.7.1.2 | Quick Access . . . . .   | 14 |
| 3.7.1.3 | Access Memory . . . . .  | 15 |
| 3.8     | Program Buffer . . . . .   | 16 |
| 3.9     | Overview of States . . . . .   | 17 |
| 3.10    | System Bus Access . . . . .  | 17 |
| 3.11    | Minimally Intrusive Debugging . . . . .  | 19 |
| 3.12    | Security . . . . .   | 19 |
| 3.13    | Version Detection . . . . .  | 20 |
| 3.14    | Debug Module Registers . . . . .   | 20 |
| 3.14.1  | Debug Module Status ( <code>dmstatus</code> , at 0x11) . . . . .               | 21 |
| 3.14.2  | Debug Module Control ( <code>dmcontrol</code> , at 0x10) . . . . .             | 23 |
| 3.14.3  | Hart Info ( <code>hartinfo</code> , at 0x12) . . . . .                         | 26 |
| 3.14.4  | Hart Array Window Select ( <code>hawindowse1</code> , at 0x14) . . . . .       | 27 |
| 3.14.5  | Hart Array Window ( <code>hawindow</code> , at 0x15) . . . . .                 | 27 |
| 3.14.6  | Abstract Control and Status ( <code>abstractcs</code> , at 0x16) . . . . .     | 28 |
| 3.14.7  | Abstract Command ( <code>command</code> , at 0x17) . . . . .                   | 29 |
| 3.14.8  | Abstract Command Autoexec ( <code>abstractauto</code> , at 0x18) . . . . .     | 30 |
| 3.14.9  | Configuration String Pointer 0 ( <code>confstrptr0</code> , at 0x19) . . . . . | 30 |
| 3.14.10 | Next Debug Module ( <code>nextdm</code> , at 0x1d) . . . . .                   | 31 |
| 3.14.11 | Abstract Data 0 ( <code>data0</code> , at 0x04) . . . . .                      | 31 |
| 3.14.12 | Program Buffer 0 ( <code>progbuf0</code> , at 0x20) . . . . .                  | 31 |
| 3.14.13 | Authentication Data ( <code>authdata</code> , at 0x30) . . . . .               | 32 |
| 3.14.14 | Debug Module Control and Status 2 ( <code>dmcs2</code> , at 0x32) . . . . .    | 32 |

|          |   |           |
|----------|---|-----------|
| 3.14.15  | Halt Summary 0 ( <code>haltsum0</code> , at 0x40)                   | 33        |
| 3.14.16  | Halt Summary 1 ( <code>haltsum1</code> , at 0x13)                   | 33        |
| 3.14.17  | Halt Summary 2 ( <code>haltsum2</code> , at 0x34)                   | 34        |
| 3.14.18  | Halt Summary 3 ( <code>haltsum3</code> , at 0x35)                   | 34        |
| 3.14.19  | System Bus Access Control and Status ( <code>sbc</code> s, at 0x38) | 34        |
| 3.14.20  | System Bus Address 31:0 ( <code>sbaddress0</code> , at 0x39)        | 36        |
| 3.14.21  | System Bus Address 63:32 ( <code>sbaddress1</code> , at 0x3a)       | 37        |
| 3.14.22  | System Bus Address 95:64 ( <code>sbaddress2</code> , at 0x3b)       | 37        |
| 3.14.23  | System Bus Address 127:96 ( <code>sbaddress3</code> , at 0x37)      | 38        |
| 3.14.24  | System Bus Data 31:0 ( <code>sbdata0</code> , at 0x3c)              | 38        |
| 3.14.25  | System Bus Data 63:32 ( <code>sbdata1</code> , at 0x3d)             | 39        |
| 3.14.26  | System Bus Data 95:64 ( <code>sbdata2</code> , at 0x3e)             | 39        |
| 3.14.27  | System Bus Data 127:96 ( <code>sbdata3</code> , at 0x3f)            | 40        |
| 3.14.28  | Custom Features ( <code>custom</code> , at 0x1f)                    | 40        |
| 3.14.29  | Custom Features 0 ( <code>custom0</code> , at 0x70)                 | 40        |
| 3.14.30  | Custom Features 15 ( <code>custom15</code> , at 0x7f)               | 40        |
| <b>4</b> | <b>RISC-V Debug</b>   | <b>41</b> |
| 4.1      | Debug Mode  | 41        |
| 4.2      | Load-Reserved/Store-Conditional Instructions                        | 42        |
| 4.3      | Wait for Interrupt Instruction                                      | 42        |
| 4.4      | Single Step   | 42        |
| 4.5      | Reset   | 43        |
| 4.6      | <code>dret</code> Instruction                                       | 43        |
| 4.7      | XLEN  | 43        |
| 4.8      | Core Debug Registers  | 43        |
| 4.8.1    | Debug Control and Status ( <code>dcsr</code> , at 0x7b0)            | 44        |
| 4.8.2    | Debug PC ( <code>dpc</code> , at 0x7b1)                             | 46        |

|          |   |           |
|----------|---|-----------|
| 4.8.3    | Debug Scratch Register 0 ( <code>dscratch0</code> , at 0x7b2) . . . . . | 47        |
| 4.8.4    | Debug Scratch Register 1 ( <code>dscratch1</code> , at 0x7b3) . . . . . | 47        |
| 4.9      | Virtual Debug Registers . . . . .                                       | 47        |
| 4.9.1    | Privilege Level ( <code>priv</code> , at virtual) . . . . .             | 47        |
| <b>5</b> | <b>Trigger Module</b>   | <b>49</b> |
| 5.1      | Native M-Mode Triggers . . . . .  | 50        |
| 5.2      | Trigger Registers . . . . .   | 50        |
| 5.2.1    | Trigger Select ( <code>tselect</code> , at 0x7a0) . . . . .             | 51        |
| 5.2.2    | Trigger Data 1 ( <code>tdata1</code> , at 0x7a1) . . . . .              | 52        |
| 5.2.3    | Trigger Data 2 ( <code>tdata2</code> , at 0x7a2) . . . . .              | 52        |
| 5.2.4    | Trigger Data 3 ( <code>tdata3</code> , at 0x7a3) . . . . .              | 53        |
| 5.2.5    | Trigger Info ( <code>tinfo</code> , at 0x7a4) . . . . .                 | 53        |
| 5.2.6    | Trigger Control ( <code>tcontrol1</code> , at 0x7a5) . . . . .          | 53        |
| 5.2.7    | Machine Context ( <code>mcontext</code> , at 0x7a8) . . . . .           | 54        |
| 5.2.8    | Supervisor Context ( <code>scontext</code> , at 0x7aa) . . . . .        | 54        |
| 5.2.9    | Match Control ( <code>mcontrol1</code> , at 0x7a1) . . . . .            | 55        |
| 5.2.10   | Instruction Count ( <code>icount</code> , at 0x7a1) . . . . .           | 60        |
| 5.2.11   | Interrupt Trigger ( <code>itrigger</code> , at 0x7a1) . . . . .         | 61        |
| 5.2.12   | Exception Trigger ( <code>etrigger</code> , at 0x7a1) . . . . .         | 62        |
| 5.2.13   | Trigger Extra (RV32) ( <code>textra32</code> , at 0x7a3) . . . . .      | 62        |
| 5.2.14   | Trigger Extra (RV64) ( <code>textra64</code> , at 0x7a3) . . . . .      | 63        |
| <b>6</b> | <b>Debug Transport Module (DTM)</b>                                     | <b>64</b> |
| 6.1      | JTAG Debug Transport Module . . . . .                                   | 64        |
| 6.1.1    | JTAG Background . . . . .   | 64        |
| 6.1.2    | JTAG DTM Registers . . . . .  | 65        |
| 6.1.3    | IDCODE (at 0x01) . . . . .  | 65        |
| 6.1.4    | DTM Control and Status ( <code>dtmcs</code> , at 0x10) . . . . .        | 66        |

|          |   |           |
|----------|---|-----------|
| 6.1.5    | Debug Module Interface Access ( <code>dmi</code> , at 0x11) | 67        |
| 6.1.6    | BYPASS (at 0x1f)  | 68        |
| 6.1.7    | Recommended JTAG Connector                                  | 69        |
| <b>A</b> | <b>Hardware Implementations</b>                             | <b>71</b> |
| A.1      | Abstract Command Based                                      | 71        |
| A.2      | Execution Based   | 71        |
| <b>B</b> | <b>Debugger Implementation</b>                              | <b>73</b> |
| B.1      | Debug Module Interface Access                               | 73        |
| B.2      | Checking for Halted Harts                                   | 74        |
| B.3      | Halting   | 74        |
| B.4      | Running   | 74        |
| B.5      | Single Step   | 74        |
| B.6      | Accessing Registers   | 74        |
| B.6.1    | Using Abstract Command                                      | 74        |
| B.6.2    | Using Program Buffer  | 75        |
| B.7      | Reading Memory  | 75        |
| B.7.1    | Using System Bus Access                                     | 75        |
| B.7.2    | Using Program Buffer  | 76        |
| B.7.3    | Using Abstract Memory Access                                | 77        |
| B.8      | Writing Memory  | 78        |
| B.8.1    | Using System Bus Access                                     | 78        |
| B.8.2    | Using Program Buffer  | 78        |
| B.8.3    | Using Abstract Memory Access                                | 79        |
| B.9      | Triggers  | 80        |
| B.10     | Handling Exceptions   | 81        |
| B.11     | Quick Access  | 81        |



**Index**

**82**

**C Change Log**

**85**

# List of Figures

|  |    |
|--|----|
| 2.1 RISC-V Debug System Overview . . . . . | 6  |
| 3.1 Run/Halt Debug State Machine . . . . . | 18 |

# List of Tables

|     |  |    |
|-----|--|----|
| 1.2 | Register Access Abbreviations . . . . .                | 3  |
| 3.1 | Use of Data Registers . . . . .                        | 12 |
| 3.2 | Meaning of <code>cmdtype</code> . . . . .              | 13 |
| 3.3 | Abstract Register Numbers . . . . .                    | 13 |
| 3.7 | System Bus Data Bits . . . . .                         | 19 |
| 3.8 | Debug Module Debug Bus Registers . . . . .             | 21 |
| 4.1 | Core Debug Registers . . . . .                         | 44 |
| 4.3 | Virtual address in DPC upon Debug Mode Entry . . . . . | 46 |
| 4.4 | Virtual Core Debug Registers . . . . .                 | 47 |
| 4.5 | Privilege Level Encoding . . . . .                     | 48 |
| 5.1 | action encoding . . . . .                              | 51 |
| 5.2 | Trigger Registers . . . . .                            | 51 |
| 5.8 | Suggested Breakpoint Timings . . . . .                 | 55 |
| 6.1 | JTAG DTM TAP Registers . . . . .                       | 65 |
| 6.5 | MIPI-10 Connector Diagram . . . . .                    | 69 |
| 6.6 | MIPI-20 Connector Diagram . . . . .                    | 69 |
| 6.7 | JTAG Connector Pinout . . . . .                        | 70 |



# 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*.

*DXLEN* of a hart is its widest supported XLEN, ignoring the current value of MXL in `misa`.

#### 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 (the ISA Spec)

2. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 (the Privileged Spec)

### 1.1.2 Versions

Version 0.13 of this document was ratified by the RISC-V Foundation’s board. Versions 0.13.x are bug fix releases to that ratified specification.

Version 0.14 will be forwards and backwards compatible with Version 0.13.

## 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 appendices is intended to clarify and provide examples, but is not part of the actual specification.

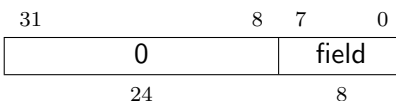
### 1.2.2 Register Definition Format

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 1.2. The reset value is either a constant or “Preset.” The latter means it is an implementation-specific legal value.

Names of registers and their fields are hyperlinks to their definition, and are also listed in the index on page 82.

#### 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    |

Table 1.2: Register Access Abbreviations

|       |   |
|-------|---|
| R     | Read-only.  |
| R/W   | Read/Write.   |
| 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.   |
| WARL  | Write any, read legal. A debugger may write any value. If a value is unsupported, the implementation converts the value to one that is supported. |

### 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.

### 1.4 Supported Features

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

1. All hart registers (including CSRs) can be read/written.
2. Memory can be accessed either from the hart's point of view, through the system bus directly, or both.
3. RV32, RV64, and future RV128 are all supported.
4. Any hart in the platform can be independently debugged.
5. A debugger can discover almost<sup>1</sup> everything it needs to know itself, without user configuration.

---

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

6. Each hart can be debugged from the very first instruction executed.
7. A RISC-V hart can be halted when a software breakpoint instruction is executed.
8. Hardware single-step can execute one instruction at a time.
9. Debug functionality is independent of the debug transport used.
10. The debugger does not need to know anything about the microarchitecture of the harts it is debugging.
11. Arbitrary subsets of harts can be halted and resumed simultaneously. (Optional)
12. 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)
13. Registers can be accessed without halting. (Optional)
14. A running hart can be directed to execute a short sequence of instructions, with little overhead. (Optional)
15. A system bus master allows memory access without involving any hart. (Optional)
16. A RISC-V hart can be halted when a trigger matches the PC, read/write address/data, or an instruction opcode. (Optional)
17. Harts can be grouped, and harts in the same group will all halt when any of them halts. These groups can also react to or notify external triggers. (Optional)

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.

It is possible to debug code that uses software threads, but there is no special debug support for it.



## 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 (e.g. laptop), which is running a debugger (e.g. gdb). The debugger communicates with a Debug Translator (e.g. OpenOCD, which may include a hardware driver) to communicate with Debug Transport Hardware (e.g. 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 one or more Debug Modules (DMs) using the Debug Module Interface (DMI).

Each hart in the platform is controlled by exactly one DM. Harts may be heterogeneous. There is no further limit on the hart-DM mapping, but usually all harts in a single core are controlled by the same DM. In most platforms there will only be one DM that controls all the harts in the platform.

DMs provide run control of their harts 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 also 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.

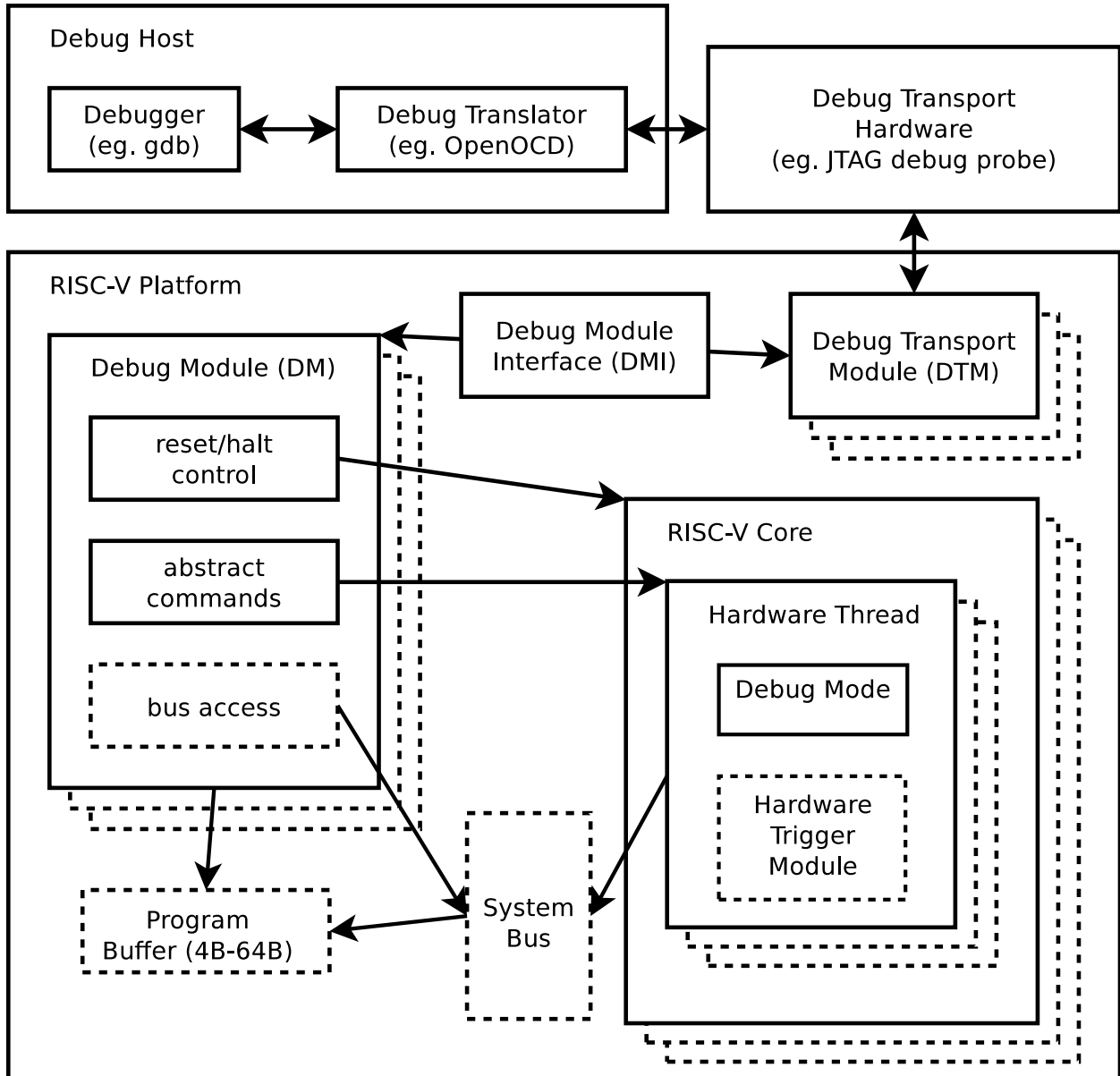


Figure 2.1: RISC-V Debug System Overview

## 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 abstract 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 a mechanism to allow debugging harts immediately out of reset (regardless of the reset cause). (Optional)
7. Provide abstract access to non-GPR hart registers. (Optional)
8. Provide a Program Buffer to force the hart to execute arbitrary instructions. (Optional)
9. Allow multiple harts to be halted, resumed, and/or reset at the same time. (Optional)
10. Allow memory access from a hart's point of view. (Optional)
11. Allow direct System Bus Access. (Optional)
12. Group harts. When any hart in the group halts, they all halt. (Optional)
13. Respond to external triggers by halting each hart in a configured group. (Optional)
14. Signal an external trigger when a hart in a group halts. (Optional)

In order to be compliant with this specification an implementation must:

1. Implement all the required features listed above.
2. Implement at least one of Program Buffer, System Bus Access, or Abstract Access Memory command mechanisms.
3. Do at least one of:
  - (a) Implement the Program Buffer.
  - (b) Implement abstract access to all registers that are visible to software running on the hart including all the registers that are present on the hart and listed in Table 3.3.
  - (c) Implement abstract access to at least all GPRs, `dcsr`, and `dpc`, and advertise the implementation as conforming to the “Minimal RISC-V Debug Specification 0.14.0-DRAFT”,

instead of the “RISC-V Debug Specification 0.14.0-DRAFT”.

A single DM can debug up to  $2^{20}$  harts.

### 3.1 Debug Module Interface (DMI)

Debug Modules are slaves 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 first (and usually only) DM. Extra space can be used for custom debug devices, other cores, additional DMs, etc. If there are additional DMs on this DMI, the base address of the next DM in the DMI address space is given in `nextdm`.

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

### 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 operations are reading and writing `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`.

Individual harts (or several at once) can be reset by selecting them, setting and then clearing `hartreset`. In this case an implementation may reset more harts than just the ones that are selected. The debugger can discover which other harts are reset (if any) by selecting them and checking `anyhavereset` and `allhavereset`.

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.

When a hart comes out of reset and `haltreq` or `resethaltreq` are set, the hart will immediately enter Debug Mode. Otherwise it will execute normally.

### 3.3 Selecting Harts

Up to  $2^{20}$  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.

To enumerate all the harts, a debugger must first determine `HARTSELLEN` by writing all ones to `hartsel` (assuming the maximum size) and reading back the value to see which bits were actually set. Then it selects each hart starting from 0 until either `anynonexistent` in `dmstatus` is 1, or the highest index (depending on `HARTSELLEN`) is reached.

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

#### 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 implement a Hart Array Mask register to allow selecting multiple harts at once. The  $n$ th bit in the Hart Array Mask register applies to the hart with index  $n$ . If the bit is 1 then the hart is selected. Usually a DM will have a Hart Array Mask register exactly wide enough to select all the harts it supports, but it's allowed to tie any of these bits to 0.

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. The state of the hart array mask register is not affected by setting or clearing `hasel`.

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 Hart States

Every hart that can be selected is in exactly one of the following four states: non-existent, unavailable, running, or halted. Which state the selected harts are in is reflected by `allnexistent`, `anynonexistent`, `allunavail`, `anyunavail`, `allrunning`, `anyrunning`, `allhalted`, and `anyhalted`.

Harts are nonexistent if they will never be part of this system, no matter how long a user waits. E.g. in a simple single-hart system only one hart exists, and all others are nonexistent. Debuggers may assume that a system has no harts with indexes higher than the first nonexistent one.

Harts are unavailable if they might exist/become available at a later time, or if there are other harts with higher indexes than this one. Harts may be unavailable for a variety of reasons including being reset, temporarily powered down, and not being plugged into the system. Systems with very large number of harts may permanently disable some during manufacturing, leaving holes in the otherwise continuous hart index space. In order to let the debugger discover all harts, they must show up as unavailable even if there is no chance of them ever becoming available.

Harts are running when they are executing normally, as if no debugger was attached. This includes being in a low power mode or waiting for an interrupt, as long as a halt request will result in the hart being halted.

Harts are halted when they are in Debug Mode, only performing tasks on behalf of the debugger.

Which states a hart that is reset goes through is implementation dependent. Harts may be unavailable while reset is asserted, and some time after reset is deasserted. They might transition to running for some time after reset is deasserted. Finally they end up either running or halted, depending on [haltreq](#) and [resethaltreq](#).

### 3.5 Run Control

For every hart, the Debug Module tracks 4 conceptual bits of state: halt request, resume ack, halt-on-reset request, and hart reset. (The hart reset and halt-on-reset request bits are optional.) These 4 bits reset to 0, except for resume ack, which may reset to either 0 or 1. The DM receives halted, running, and havereset signals from each hart. The debugger can observe the state of resume ack in [allresumeack](#) and [anyresumeack](#), and the state of halted, running, and havereset signals in [allhalted](#), [anyhalted](#), [allrunning](#), [anyrunning](#), [allhavereset](#), and [anyhavereset](#). The state of the other bits cannot be observed directly.

When a debugger writes 1 to [haltreq](#), each selected hart's halt request bit is set. When a running hart, or a hart just coming out of reset, sees its halt request bit high, it responds by halting, deasserting its running signal, and asserting its halted signal. Halted harts ignore their halt request bit.

When a debugger writes 1 to [resumereq](#), each selected hart's resume ack bit is cleared and each selected, halted hart is sent a resume request. Harts respond by resuming, clearing their halted signal, and asserting their running signal. At the end of this process the resume ack bit is set. These status signals of all selected harts are reflected in [allresumeack](#), [anyresumeack](#), [allrunning](#), and [anyrunning](#). Resume requests are 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).

The DM can implement optional halt-on-reset bits for each hart, which it indicates by setting

`hasresethaltreq` to 1. This means the DM implements the `setresethaltreq` and `clrresethaltreq` bits. Writing 1 to `setresethaltreq` sets the halt-on-reset request bit for each selected hart. When a hart's halt-on-reset request bit is set, the hart will immediately enter debug mode on the next deassertion of its reset. This is true regardless of the reset's cause. The hart's halt-on-reset request bit remains set until cleared by the debugger writing 1 to `clrresethaltreq` while the hart is selected, or by DM reset.

### 3.6 Halt Groups and External Triggers

An optional feature allows a debugger to create halt groups. When any hart in a halt group halts all the other harts in that group will quickly halt, even if they are currently in the process of resuming. Adding a hart to a halt group does not automatically halt that hart, even if other harts in the group are already halted.

It is also possible to add external triggers to a halt group. External triggers are abstract concepts that can signal the DM and/or receive signals from the DM. When an external trigger fires, all harts in its halt group are quickly halted. When a hart in a halt group halts, the external triggers in the same halt group are notified.

---

*External triggers could be used to implement near simultaneous halting of all cores in a system, when not all cores are RISC-V cores.*

Halt group 0 is special, and has none of this behavior. Harts in halt group 0 halt as if halt groups aren't implemented at all.

To near simultaneously resume all harts in a group, the debugger can select all the harts, and write 1 to `resumereq` as described above. An implementation must guarantee that all harts in the group halt again as soon as one of them halts, even if that hart halts before all harts have resumed.

### 3.7 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. Commands may be supported with some options set, but not with other options set. If a command has unsupported options set, the DM must set `cmderr` to 2 (not supported).

---

*Example: Every system must support the Access Register command, but may not support accessing CSRs. If the debugger requests to read a CSR in that case, the command will return "not supported."*

Debuggers execute abstract commands by writing them to `command`. They can determine whether an abstract command is complete by reading `busy` in `abstractcs`. After completion, `cmderr` indicates whether the command was successful or not. Commands may fail because a hart is not halted, not running, unavailable, or because they encounter an error during execution.

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 3.1. In all cases the least-significant word is placed in the lowest-numbered `data` register. The argument width depends on the command being executed, and is `DXLEN` where not explicitly specified.

Table 3.1: Use of Data Registers

| Argument Width | 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> |

---

*The Abstract Command interface is designed to allow a debugger to write commands as fast as possible, and then later check whether they completed without error. In the common case the debugger will be much slower than the target and commands succeed, which allows for maximum throughput. If there is a failure, the interface ensures that no commands execute after the failing one. To discover which command failed, the debugger has to look at the state of the DM (e.g. contents of `data0`) or hart (e.g. contents of a register modified by a Program Buffer program) to determine which one failed.*

Before starting an abstract command, a debugger must ensure that `haltreq`, `resumereq`, and `ackhavereset` are all 0.

While an abstract command is executing (`busy` in `abstractcs` is high), a debugger must not change `hartsel`, and must not write 1 to `haltreq`, `resumereq`, `ackhavereset`, `setresethaltreq`, or `clrresethaltreq`.

If an abstract command does not complete in the expected time and appears to be hung, the following procedure can be attempted to abort the command: First the debugger resets the hart (using `hartreset` or `ndmreset`), and then it resets the Debug Module (using `dmactive`).

If an abstract command is started while the selected hart is unavailable or if a hart becomes unavailable while executing an abstract command, then the Debug Module may terminate the abstract command, setting `busy` low, and `cmderr` to 4 (halt/resume). Alternatively, the command could just appear to be hung (`busy` never goes low).

### 3.7.1 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 3.2 lists all commands.



Table 3.2: Meaning of `cmdtype`

| <code>cmdtype</code> | Command                 | Page |
|----------------------|-------------------------|------|
| 0                    | Access Register Command | 13   |
| 1                    | Quick Access            | 14   |
| 2                    | Access Memory Command   | 15   |

### 3.7.1.1 Access Register

This command gives the debugger access to CPU registers and allows it to execute the Program Buffer. It performs the following sequence of operations:

1. If `write` is clear and `transfer` is set, then copy data from the register specified by `regno` into the `arg0` region of `data`, and perform any side effects that occur when this register is read from M-mode.
2. If `write` is set and `transfer` is set, then copy data from the `arg0` region of `data` into the register specified by `regno`, and perform any side effects that occur when this register is written from M-mode.
3. If `aarpostincrement` is set, increment `regno`.
4. 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. If the failure is that the requested register does not exist in the hart, `cmderr` must be set to 3 (exception).

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. Each individual register (aside from GPRs) may be supported differently across read, write, and halt status.

Table 3.3: 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.         |

---

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

This command modifies `arg0` only when a register is read. The other `data` registers are not changed.

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

| Field            | Description   |
|------------------|---|
| cmdtype          | This is 0 to indicate Access Register Command.  |
| aarsize          | 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>aarsize</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>aarsize</code> less than or equal to the register's actual size must be supported.<br>This field controls the Argument Width as referenced in Table 3.1. |
| aarpostincrement | 0: No effect. This variant must be supported.<br>1: After a successful register access, <code>regno</code> is incremented (wrapping around to 0). Supporting this variant is optional.  |
| postexec         | 0: No effect. This variant must be supported, and is the only supported one if <code>progbufsize</code> is 0.<br>1: Execute the program in the Program Buffer exactly once after performing the transfer, if any. Supporting this variant is optional.  |
| transfer         | 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>aarsize</code> or <code>regno</code> .  |
| write            | 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.  |
| regno            | Number of the register to access, as described in Table 3.3. <code>dpc</code> may be used as an alias for PC if this command is supported on a non-halted hart.   |

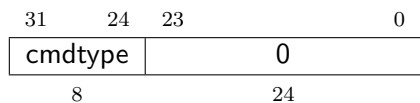
### 3.7.1.2 Quick Access

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.

This command does not touch the `data` registers.



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

### 3.7.1.3 Access Memory

This command lets the debugger perform memory accesses, with the exact same memory view and permissions as the selected hart has. This includes access to hart-local memory-mapped registers, etc. The command performs the following sequence of operations:

1. Copy data from the memory location specified in `arg1` into the `arg0` portion of `data`, if `write` is clear.
2. Copy data from the `arg0` portion of `data` into the memory location specified in `arg1`, if `write` is set.
3. If `aampostincrement` is set, increment `arg1`.

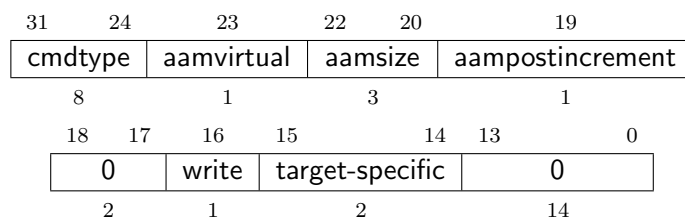
If any of these operations fail, `cmderr` is set and none of the remaining steps are executed. An access may only fail if the hart, running M-mode code, might encounter that same failure when it attempts the same access. An implementation may detect an upcoming failure early, and fail the overall command before it reaches the step that would cause failure.

Debug Modules may optionally implement this command and may support read and write access to memory locations when the selected hart is running or halted. If this command supports memory accesses while the hart is running, it must also support memory accesses while the hart is halted.

---

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

This command modifies `arg0` only when memory is read. It modifies `arg1` only if `aampostincrement` is set. The other `data` registers are not changed.



| Field                         | Description   |
|-------------------------------|---|
| <code>cmdtype</code>          | This is 2 to indicate Access Memory Command.  |
| <code>aamvirtual</code>       | An implementation does not have to implement both virtual and physical accesses, but it must fail accesses that it doesn't support.<br>0: Addresses are physical (to the hart they are performed on).<br>1: Addresses are virtual, and translated the way they would be from M-mode, with MPRV set. |
| <code>aamsize</code>          | 0: Access the lowest 8 bits of the memory location.<br>1: Access the lowest 16 bits of the memory location.<br>2: Access the lowest 32 bits of the memory location.<br>3: Access the lowest 64 bits of the memory location.<br>4: Access the lowest 128 bits of the memory location.                |
| <code>aampostincrement</code> | After a memory access has completed, if this bit is 1, increment <code>arg1</code> (which contains the address used) by the number of bytes encoded in <code>aamsize</code> .   |
| <code>write</code>            | 0: Copy data from the memory location specified in <code>arg1</code> into the low bits of <code>arg0</code> . Any remaining bits of <code>arg0</code> now have an undefined value.<br>1: Copy data from the low bits of <code>arg0</code> into the memory location specified in <code>arg1</code> . |
| <code>target-specific</code>  | These bits are reserved for target-specific uses.   |

### 3.8 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 `c.ebreak`. An implementation may support an implied `ebreak` that is executed when a hart runs off the end of the Program Buffer. This is indicated by `impebreak`. With this feature, a Program Buffer of just 2 32-bit words can offer efficient debugging.

If `progbufsize` is 1, `impebreak` must be 1. It is possible that the Program Buffer can hold only one 32-

or 16-bit instruction, so the debugger must only write a single instruction in this case, regardless of its size. 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.

---

*The slightly inconsistent behavior with a Program Buffer of size 1 is to accommodate hardware designs that prefer to stuff instructions directly into the pipeline when halted, instead of having the Program Buffer exist in the address space somewhere.*

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 with an `ebreak` instruction, the hart will remain in Debug Mode and the debugger will lose 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. 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.

## 3.9 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`.

## 3.10 System Bus Access

A debugger can access memory from a hart's point of view using a Program Buffer or the Abstract Access Memory command. (Both these features are optional.) 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.

The System Bus Access block may support 8-, 16-, 32-, 64-, and 128-bit accesses. Table 3.7 shows which bits in `sdata` are used for each access size.

Depending on the microarchitecture, data accessed through System Bus Access may not always be coherent with that observed by each hart. 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. Possibilities may include writing to special memory-mapped locations, or executing special instructions via the Program Buffer.

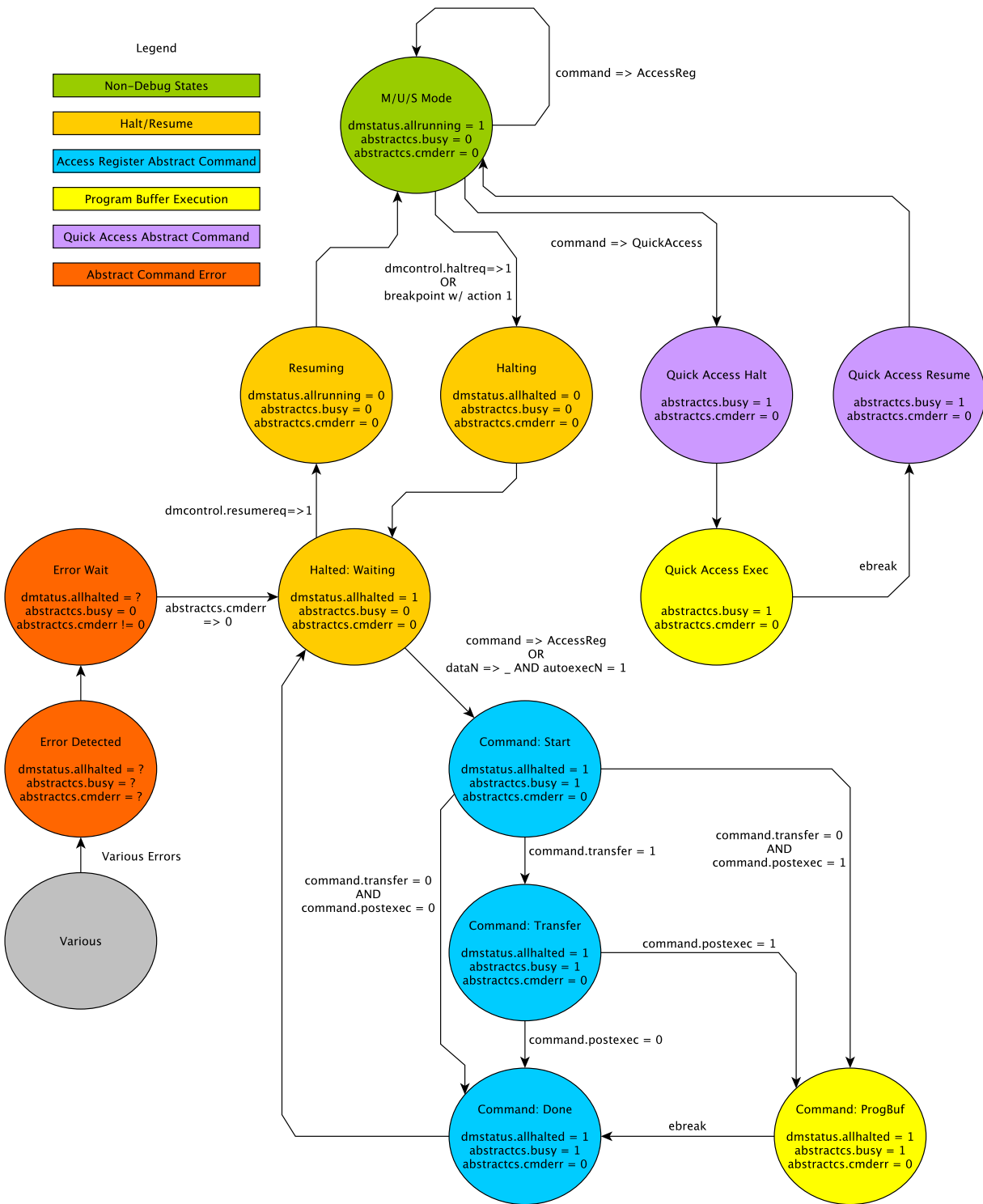


Figure 3.1: Run/Halt Debug State Machine for single-hart systems. As only a small amount of state is visible to the debugger, the states and transitions are conceptual.

Table 3.7: System Bus Data Bits

| Access Size | Data Bits   |
|-------------|---|
| 8           | <a href="#">sbdata0</a> bits 7:0  |
| 16          | <a href="#">sbdata0</a> bits 15:0   |
| 32          | <a href="#">sbdata0</a>   |
| 64          | <a href="#">sbdata1</a> , <a href="#">sbdata0</a>   |
| 128         | <a href="#">sbdata3</a> , <a href="#">sbdata2</a> , <a href="#">sbdata1</a> , <a href="#">sbdata0</a> |

---

*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.*

### 3.11 Minimally Intrusive Debugging

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 [3.14.3](#), 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.

### 3.12 Security

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

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

1. [authenticated](#) in `dmstatus` is readable.

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

### 3.13 Version Detection

To detect the version of the Debug Module with a minimum of side effects, use the following procedure:

1. Read `dmcontrol`.
2. Write `dmcontrol`, preserving `hartreset`, `hasel`, `hartsello`, and `hartselhi` from the value that was read, setting `dmactive`, and clearing all the other bits.
3. Read `dmstatus`, which contains `version`.

This has the following unavoidable side effects:

1. `haltreq` is cleared, potentially preventing a halt request made by a previous debugger from taking effect.
2. `resumereq` is cleared, potentially preventing a resume request made by a previous debugger from taking effect.
3. `ndmreset` is deasserted, releasing the system from reset if a previous debugger had set it.
4. `dmactive` is asserted, releasing the DM from reset. This in itself is not observable by any harts.

This procedure is guaranteed to work in future versions of this spec. The meaning of the `dmcontrol` bits where `hartreset`, `hasel`, `hartsello`, and `hartselhi` currently reside might change, but preserving them will have no side effects. Clearing the bits of `dmcontrol` not explicitly mentioned here will have no side effects beyond the ones mentioned above.

### 3.14 Debug Module Registers

The registers described in this section are accessed over the DMI bus. Each DM has a base address (which is 0 for the first DM). The register addresses below are offsets from this base address.

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

For each register it is possible to determine that it is implemented by reading it and getting a non-zero value (e.g. `sbc`), or by checking bits in another register (e.g. `progbufsize`).



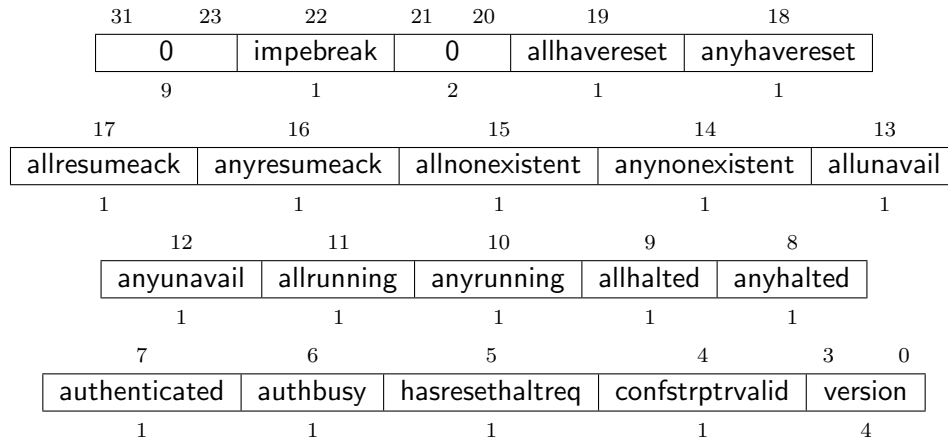
Table 3.8: Debug Module Debug Bus Registers

| Address | Name  | Page |
|---------|---|------|
| 0x04    | Abstract Data 0 ( <code>data0</code> )                      | 31   |
| 0x0f    | Abstract Data 11 ( <code>data11</code> )                    |      |
| 0x10    | Debug Module Control ( <code>dmcontrol</code> )             | 23   |
| 0x11    | Debug Module Status ( <code>dmstatus</code> )               | 21   |
| 0x12    | Hart Info ( <code>hartinfo</code> )                         | 26   |
| 0x13    | Halt Summary 1 ( <code>haltsum1</code> )                    | 33   |
| 0x14    | Hart Array Window Select ( <code>hawindowse1</code> )       | 27   |
| 0x15    | Hart Array Window ( <code>hawindow</code> )                 | 27   |
| 0x16    | Abstract Control and Status ( <code>abstractcs</code> )     | 28   |
| 0x17    | Abstract Command ( <code>command</code> )                   | 29   |
| 0x18    | Abstract Command Autoexec ( <code>abstractauto</code> )     | 30   |
| 0x19    | Configuration String Pointer 0 ( <code>confstrptr0</code> ) | 30   |
| 0x1a    | Configuration String Pointer 1 ( <code>confstrptr1</code> ) |      |
| 0x1b    | Configuration String Pointer 2 ( <code>confstrptr2</code> ) |      |
| 0x1c    | Configuration String Pointer 3 ( <code>confstrptr3</code> ) |      |
| 0x1d    | Next Debug Module ( <code>nextdm</code> )                   | 31   |
| 0x1f    | Custom Features ( <code>custom</code> )                     | 40   |
| 0x20    | Program Buffer 0 ( <code>progbuf0</code> )                  | 31   |
| 0x2f    | Program Buffer 15 ( <code>progbuf15</code> )                |      |
| 0x30    | Authentication Data ( <code>authdata</code> )               | 32   |
| 0x32    | Debug Module Control and Status 2 ( <code>dmcs2</code> )    | 32   |
| 0x34    | Halt Summary 2 ( <code>haltsum2</code> )                    | 34   |
| 0x35    | Halt Summary 3 ( <code>haltsum3</code> )                    | 34   |
| 0x37    | System Bus Address 127:96 ( <code>sbaddress3</code> )       | 38   |
| 0x38    | System Bus Access Control and Status ( <code>sbc</code> )   | 34   |
| 0x39    | System Bus Address 31:0 ( <code>sbaddress0</code> )         | 36   |
| 0x3a    | System Bus Address 63:32 ( <code>sbaddress1</code> )        | 37   |
| 0x3b    | System Bus Address 95:64 ( <code>sbaddress2</code> )        | 37   |
| 0x3c    | System Bus Data 31:0 ( <code>sbd</code> )                   | 38   |
| 0x3d    | System Bus Data 63:32 ( <code>sbd</code> )                  | 39   |
| 0x3e    | System Bus Data 95:64 ( <code>sbd</code> )                  | 39   |
| 0x3f    | System Bus Data 127:96 ( <code>sbd</code> )                 | 40   |
| 0x40    | Halt Summary 0 ( <code>haltsum0</code> )                    | 33   |
| 0x70    | Custom Features 0 ( <code>custom0</code> )                  | 40   |
| 0x7f    | Custom Features 15 ( <code>custom15</code> )                | 40   |

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

This register reports status for the overall Debug Module as well as the currently selected harts, as defined in [hasel](#). Its address will not change in the future, because it contains [version](#).

This entire register is read-only.



| Field           | Description  | Access | Reset  |
|-----------------|--|--------|--------|
| impebreak       | If 1, then there is an implicit <code>ebreak</code> instruction at the non-existent word immediately after the Program Buffer. This saves the debugger from having to write the <code>ebreak</code> itself, and allows the Program Buffer to be one word smaller. This must be 1 when <code>progbufsize</code> is 1. | R      | Preset |
| allhavereset    | This field is 1 when all currently selected harts have been reset and reset has not been acknowledged for any of them.   | R      | -      |
| anyhavereset    | This field is 1 when at least one currently selected hart has been reset and reset has not been acknowledged for that hart.  | R      | -      |
| allresumeack    | This field is 1 when all currently selected harts have acknowledged their last resume request.   | R      | -      |
| anyresumeack    | This field is 1 when any currently selected hart has acknowledged its last resume request.   | R      | -      |
| allnonexistent  | This field is 1 when all currently selected harts do not exist in this platform.   | R      | -      |
| anyononexistent | This field is 1 when any currently selected hart does not exist in this platform.  | 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      | -      |

*Continued on next page*

| Field           | Description  | Access | Reset  |
|-----------------|--|--------|--------|
| authenticated   | 0: Authentication is required before using the DM.<br>1: The authentication check has passed.<br>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 <code>authdata</code> .<br>1: The authentication module is busy. Accessing <code>authdata</code> results in unspecified behavior. <code>authbusy</code> only becomes set in immediate response to an access to <code>authdata</code> . | R      | 0      |
| hasresethaltreq | 1 if this Debug Module supports halt-on-reset functionality controllable by the <code>setresethaltreq</code> and <code>clrresethaltreq</code> bits. 0 otherwise.   | R      | Preset |
| confstrptrvalid | 0: <code>confstrptr0</code> – <code>confstrptr3</code> hold information which is not relevant to the configuration string.<br>1: <code>confstrptr0</code> – <code>confstrptr3</code> hold the address of the configuration string.   | 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      |

### 3.14.2 Debug Module Control (`dmcontrol`, at `0x10`)

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

Throughout this document we refer to `hartsel`, which is `hartselhi` combined with `hartsello`. While the spec allows for 20 `hartsel` bits, an implementation may choose to implement fewer than that. The actual width of `hartsel` is called `HARTSELLEN`. It must be at least 0 and at most 20. A debugger should discover `HARTSELLEN` by writing all ones to `hartsel` (assuming the maximum size) and reading back the value to see which bits were actually set. Debuggers must not change `hartsel` while an abstract command is executing.

---

*There are separate `setresethaltreq` and `clrresethaltreq` bits so that it is possible to write `dmcontrol` without changing the halt-on-reset request bit for each selected hart, when not all selected harts have the same configuration.*

On any given write, a debugger may only write 1 to at most one of the following bits: `resumereq`, `hartreset`, `ackhavereset`, `setresethaltreq`, and `clrresethaltreq`. The others must be written 0.

`resethaltreq` is an optional internal bit of per-hart state that cannot be read, but can be written with `setresethaltreq` and `clrresethaltreq`.

For forward compatibility, `version` will always be readable when bit 1 (`ndmreset`) is 0 and bit 0 (`dmactive`) is 1.



| Field                     | Description   | Access | Reset |
|---------------------------|---|--------|-------|
| <code>haltreq</code>      | Writing 0 clears the halt request bit for all currently selected harts. This may cancel outstanding halt requests for those harts.<br>Writing 1 sets the halt request bit for all currently selected harts. Running harts will halt whenever their halt request bit is set.<br>Writes apply to the new value of <code>hartsel</code> and <code>hasel</code> .   | W      | -     |
| <code>resumereq</code>    | Writing 1 causes the currently selected harts to resume once, if they are halted when the write occurs. It also clears the resume ack bit for those harts.<br><code>resumereq</code> is ignored if <code>haltreq</code> is set.<br>Writes apply to the new value of <code>hartsel</code> and <code>hasel</code> .   | W1     | -     |
| <code>hartreset</code>    | 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>While this bit is 1, the debugger must not change which harts are selected.<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 <code>hartsel</code> and <code>hasel</code> . | R/W    | 0     |
| <code>ackhavereset</code> | 0: No effect.<br>1: Clears <code>havereset</code> for any selected harts.<br>Writes apply to the new value of <code>hartsel</code> and <code>hasel</code> .   | W1     | -     |

*Continued on next page*

| Field           | Description  | Access | Reset |
|-----------------|--|--------|-------|
| hasel           | Selects the definition of currently selected harts.<br>0: There is a single currently selected hart, that is selected by <a href="#">hartsel</a> .<br>1: There may be multiple currently selected harts – the hart 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 must 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     |
| hartsello       | The low 10 bits of <a href="#">hartsel</a> : the DM-specific index of the hart to select. This hart is always part of the currently selected harts.  | R/W    | 0     |
| hartselhi       | The high 10 bits of <a href="#">hartsel</a> : the DM-specific index of the hart to select. This hart is always part of the currently selected harts.   | R/W    | 0     |
| setresethaltreq | This optional field writes the halt-on-reset request bit for all currently selected harts, unless <a href="#">clrresethaltreq</a> is simultaneously set to 1. When set to 1, each selected hart will halt upon the next deassertion of its reset. The halt-on-reset request bit is not automatically cleared. The debugger must write to <a href="#">clrresethaltreq</a> to clear it.<br>Writes apply to the new value of <a href="#">hartsel</a> and <a href="#">hasel</a> .<br>If <a href="#">hasresethaltreq</a> is 0, this field is not implemented.           | W1     | -     |
| clrresethaltreq | This optional field clears the halt-on-reset request bit for all currently selected harts.<br>Writes apply to the new value of <a href="#">hartsel</a> and <a href="#">hasel</a> .   | W1     | -     |
| ndmreset        | 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     |

*Continued on next page*

| Field    | Description  | Access | Reset |
|----------|--|--------|-------|
| dmactive | <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 <a href="#">dmactive</a> bit is the only bit which can be written to something other than its reset value). Any accesses to the module may fail. Specifically, <a href="#">version</a> may not return correct data.</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, with the possible (but not recommended) exception of a global reset signal that resets the entire platform.</p> <p>A debugger may pulse this bit low to get the Debug Module into a known state.</p> <p>Implementations may pay attention to this bit to further aid debugging, for example by preventing the Debug Module from being power gated while debugging is active.</p> | R/W    | 0     |

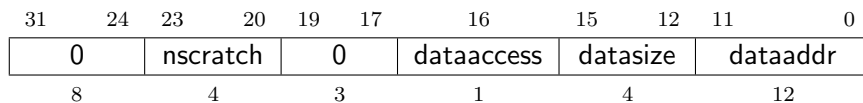
### 3.14.3 Hart Info (hartinfo, at 0x12)

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.



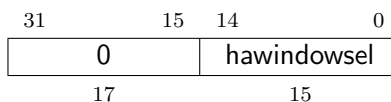
| Field    | Description  | Access | Reset  |
|----------|--|--------|--------|
| nscratch | Number of <code>dscratch</code> registers available for the debugger to use during program buffer execution, starting from <code>dscratch0</code> . The debugger can make no assumptions about the contents of these registers between commands. | R      | Preset |

*Continued on next page*

| Field      | Description   | Access | Reset  |
|------------|---|--------|--------|
| dataaccess | 0: The <b>data</b> registers are shadowed in the hart by CSRs. Each CSR is DXLEN bits in size, and corresponds to a single argument, per Table 3.1.<br>1: The <b>data</b> registers are shadowed in the hart's memory map. Each register takes up 4 bytes in the memory map.  | R      | Preset |
| datasize   | If <b>dataaccess</b> is 0: Number of CSRs dedicated to shadowing the <b>data</b> registers.<br>If <b>dataaccess</b> is 1: Number of 32-bit words in the memory map dedicated to shadowing the <b>data</b> registers.<br>Since there are at most 12 <b>data</b> registers, the value in this register must be 12 or smaller. | R      | Preset |
| dataaddr   | If <b>dataaccess</b> is 0: The number of the first CSR dedicated to shadowing the <b>data</b> registers.<br>If <b>dataaccess</b> is 1: Signed address of RAM where the <b>data</b> registers are shadowed, to be used to access relative to <b>zero</b> .   | R      | Preset |

### 3.14.4 Hart Array Window Select (**hawindowse1**, at 0x14)

This register selects which of the 32-bit portion of the hart array mask register (see Section 3.3.2) is accessible in **hawindow**.

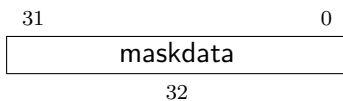


| Field       | Description  | Access | Reset |
|-------------|--|--------|-------|
| hawindowse1 | The high bits of this field may be tied to 0, depending on how large the array mask register is. E.g. on a system with 48 harts only bit 0 of this field may actually be writable. | R/W    | 0     |

### 3.14.5 Hart Array Window (**hawindow**, at 0x15)

This register provides R/W access to a 32-bit portion of the hart array mask register (see Section 3.3.2). The position of the window is determined by **hawindowse1**. I.e. bit 0 refers to hart **hawindowse1** \* 32, while bit 31 refers to hart **hawindowse1** \* 32 + 31.

Since some bits in the hart array mask register may be constant 0, some bits in this register may be constant 0, depending on the current value of `hawindowssel`.

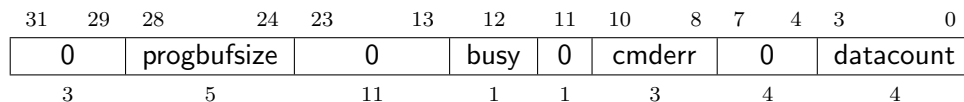


### 3.14.6 Abstract Control and Status (`abstractcs`, at `0x16`)

Writing this register while an abstract command is executing causes `cmderr` to be set to 1 (busy) if it is 0.

---

`datacount` must be at least 1 to support RV32 harts, 2 to support RV64 harts, or 4 to support RV128 harts.



| Field                    | Description   | Access | Reset  |
|--------------------------|---|--------|--------|
| <code>progbufsize</code> | Size of the Program Buffer, in 32-bit words. Valid sizes are 0 - 16.  | R      | Preset |
| <code>busy</code>        | 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      |

*Continued on next page*



| Field                  | Description  | Access | Reset  |
|------------------------|--|--------|--------|
| <code>cmderr</code>    | 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>This field only contains a valid value if <code>busy</code> is 0.<br>0 (none): No error.<br>1 (busy): An abstract command was executing while <code>command</code> , <code>abstractcs</code> , or <code>abstractauto</code> was written, or when one of the <code>data</code> or <code>progbuf</code> registers was read or written. This status is only written if <code>cmderr</code> contains 0.<br>2 (not supported): The command in <code>command</code> is not supported. It may be supported with different options set, but it will not be supported at a later time when the hart or system state are different.<br>3 (exception): An exception occurred while executing the command (e.g. while executing the Program Buffer).<br>4 (halt/resume): The abstract command couldn't execute because the hart wasn't in the required state (running/halted), or unavailable.<br>5 (bus): The abstract command failed due to a bus error (e.g. alignment, access size, or timeout).<br>7 (other): The command failed for another reason. | R/W1C  | 0      |
| <code>datacount</code> | Number of <code>data</code> registers that are implemented as part of the abstract command interface. Valid sizes are 1 – 12.  | R      | Preset |

### 3.14.7 Abstract Command (`command`, at 0x17)

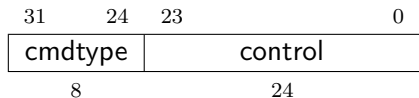
Writes to this register cause the corresponding abstract command to be executed.

Writing this register while an abstract command is executing causes `cmderr` to be set to 1 (busy) if it is 0.

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.*

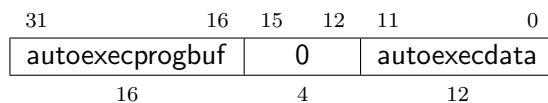


| Field   | Description  | Access | Reset |
|---------|--|--------|-------|
| cmdtype | The type determines the overall functionality of this abstract command.                      | W      | 0     |
| control | This field is interpreted in a command-specific manner, described for each abstract command.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

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

This register is optional. Including it allows more efficient burst accesses. A debugger can detect whether it is support by setting bits and reading them back.

Writing this register while an abstract command is executing causes `cmderr` to be set to 1 (busy) if it is 0.



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

### 3.14.9 Configuration String Pointer 0 (confstrptr0, at 0x19)

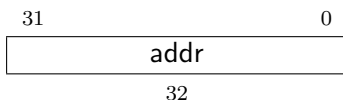
When `confstrptrvalid` is set, reading this register returns bits 31:0 of the configuration string pointer. Reading the other `confstrptr` registers returns the upper bits of the address.

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 configuration string from the hart with ID 0.

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

The configuration string itself is described in the Privileged Spec.

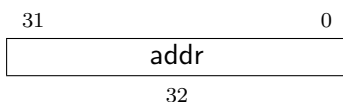
This entire register is read-only.



### 3.14.10 Next Debug Module (`nextdm`, at `0x1d`)

If there is more than one DM accessible on this DMI, this register contains the base address of the next one in the chain, or 0 if this is the last one in the chain.

This entire register is read-only.



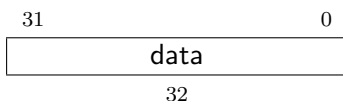
### 3.14.11 Abstract Data 0 (`data0`, at `0x04`)

`data0` through `data11` are basic read/write registers that may be read or changed by abstract commands. `datacount` indicates how many of them are implemented, starting at `data0`, counting up. Table 3.1 shows how abstract commands use these registers.

Accessing these registers while an abstract command is executing causes `cmderr` to be set to 1 (busy) if it is 0.

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.

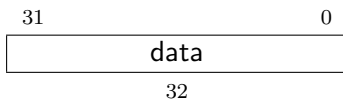


### 3.14.12 Program Buffer 0 (`progbuf0`, at `0x20`)

`progbuf0` through `progbuf15` provide read/write access to the optional program buffer. `progbufsize` indicates how many of them are implemented starting at `progbuf0`, counting up.

Accessing these registers while an abstract command is executing causes `cmderr` to be set to 1 (busy) if it is 0.

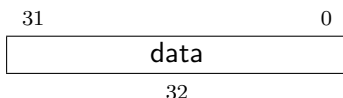
Attempts to write them while `busy` is set does not change their value.



### 3.14.13 Authentication Data (`authdata`, at `0x30`)

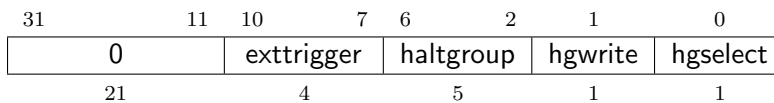
This register serves as a 32-bit serial port to/from the authentication module.

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.



### 3.14.14 Debug Module Control and Status 2 (`dmcs2`, at `0x32`)

This register contains DM control and status bits that didn't easily fit in `dmcontrol` and `dmstatus`. All are optional.



| Field                   | Description   | Access | Reset |
|-------------------------|---|--------|-------|
| <code>exttrigger</code> | This field contains the currently selected external trigger.<br>If a non-existent trigger value is written here, the hardware will change it to a valid one or 0 if no external triggers exist.   | WARL   | 0     |
| <code>haltgroup</code>  | When <code>hgselect</code> is 0, contains the halt group of the hart specified by <code>hartsel</code> .<br>When <code>hgselect</code> is 1, contains the halt group of the external trigger selected by <code>exttrigger</code> .<br>Writes only have an effect if <code>hgwrite</code> is also written 1.<br>An implementation may tie any number of upper bits in this field to 0. If halt groups aren't implemented, then this entire field is 0. | WARL   | 0     |

*Continued on next page*

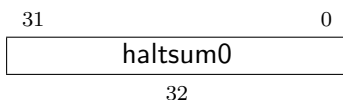
| Field    | Description  | Access | Reset |
|----------|--|--------|-------|
| hgwrite  | When <code>hgselect</code> is 0, writing 1 changes the halt group of all selected harts to the value written to <code>haltgroup</code> .<br>When <code>hgselect</code> is 1, writing 1 changes the halt group of the external trigger selected by <code>exttrigger</code> to the value written to <code>haltgroup</code> .<br>Writing 0 has no effect. | W      | -     |
| hgselect | 0: Operate on harts.<br>1: Operate on external triggers.<br>If there are no external triggers, this field must be tied to 0.   | WARL   | 0     |

### 3.14.15 Halt Summary 0 (`haltsum0`, at 0x40)

Each bit in this read-only register indicates whether one specific hart is halted or not. Unavailable/nonexistent harts are not considered to be halted.

The LSB reflects the halt status of hart `{hartsel[19:5],5'h0}`, and the MSB reflects halt status of hart `{hartsel[19:5],5'h1f}`.

This entire register is read-only.



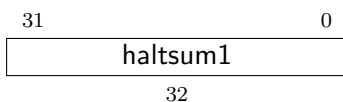
### 3.14.16 Halt Summary 1 (`haltsum1`, at 0x13)

Each bit in this read-only register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted.

This register may not be present in systems with fewer than 33 harts.

The LSB reflects the halt status of harts `{hartsel[19:10],10'h0}` through `{hartsel[19:10],10'h1f}`. The MSB reflects the halt status of harts `{hartsel[19:10],10'h3e0}` through `{hartsel[19:10],10'h3ff}`.

This entire register is read-only.



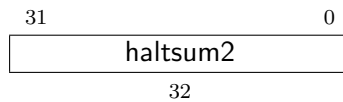
### 3.14.17 Halt Summary 2 (haltsum2, at 0x34)

Each bit in this read-only register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted.

This register may not be present in systems with fewer than 1025 harts.

The LSB reflects the halt status of harts  $\{\text{hartsel}[19:15], 15'h0\}$  through  $\{\text{hartsel}[19:15], 15'h3ff\}$ . The MSB reflects the halt status of harts  $\{\text{hartsel}[19:15], 15'h7c00\}$  through  $\{\text{hartsel}[19:15], 15'h7fff\}$ .

This entire register is read-only.



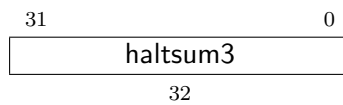
### 3.14.18 Halt Summary 3 (haltsum3, at 0x35)

Each bit in this read-only register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted.

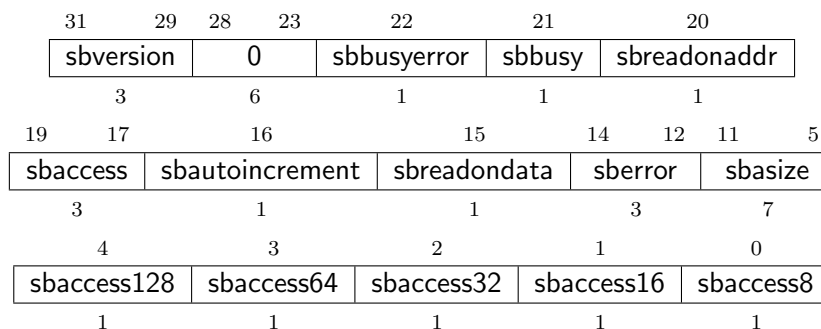
This register may not be present in systems with fewer than 32769 harts.

The LSB reflects the halt status of harts  $20'h0$  through  $20'h7fff$ . The MSB reflects the halt status of harts  $20'hf8000$  through  $20'hffff$ .

This entire register is read-only.



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



| Field           | Description   | Access | Reset |
|-----------------|---|--------|-------|
| sbversion       | 0: The System Bus interface conforms to mainline drafts of this spec older than 1 January, 2018.<br>1: The System Bus interface conforms to this version of the spec.<br>Other values are reserved for future versions.   | R      | 1     |
| sbbusyerror     | Set when the debugger attempts to read data while a read is in progress, or when the debugger initiates a new access while one is already in progress (while <code>sbbusy</code> is set). It remains set until it's explicitly cleared by the debugger.<br>While this field is set, no more system bus accesses can be initiated by the Debug Module.   | R/W1C  | 0     |
| sbbusy          | When 1, indicates the system bus master is busy. (Whether the system bus itself is busy is related, but not the same thing.) This bit goes high immediately when a read or write is requested for any reason, and does not go low until the access is fully completed.<br>Writes to <code>sbc</code> s while <code>sbbusy</code> is high result in undefined behavior. A debugger must not write to <code>sbc</code> s until it reads <code>sbbusy</code> as 0. | R      | 0     |
| sbreadonaddr    | When 1, every write to <code>sbaddress0</code> automatically triggers a system bus read at the new address.   | R/W    | 0     |
| sbaccess        | Select the access size to use for system bus accesses.<br>0: 8-bit<br>1: 16-bit<br>2: 32-bit<br>3: 64-bit<br>4: 128-bit<br>If <code>sbaccess</code> has an unsupported value when the DM starts a bus access, the access is not performed and <code>sberror</code> is set to 4.   | R/W    | 2     |
| sbautoincrement | When 1, <code>sbaddress</code> is incremented by the access size (in bytes) selected in <code>sbaccess</code> after every system bus access.  | R/W    | 0     |
| sbreadondata    | When 1, every read from <code>sbdatab0</code> automatically triggers a system bus read at the (possibly auto-incremented) address.  | R/W    | 0     |

*Continued on next page*

| Field                    | Description  | Access | Reset  |
|--------------------------|--|--------|--------|
| <code>sberror</code>     | When the Debug Module’s system bus master encounters an 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>An implementation may report “Other” (7) for any error condition.<br>0: There was no bus error.<br>1: There was a timeout.<br>2: A bad address was accessed.<br>3: There was an alignment error.<br>4: An access of unsupported size was requested.<br>7: Other. | R/W1C  | 0      |
| <code>sbasize</code>     | Width of system bus addresses in bits. (0 indicates there is no bus access support.)   | R      | Preset |
| <code>sbaccess128</code> | 1 when 128-bit system bus accesses are supported.  | R      | Preset |
| <code>sbaccess64</code>  | 1 when 64-bit system bus accesses are supported.   | R      | Preset |
| <code>sbaccess32</code>  | 1 when 32-bit system bus accesses are supported.   | R      | Preset |
| <code>sbaccess16</code>  | 1 when 16-bit system bus accesses are supported.   | R      | Preset |
| <code>sbaccess8</code>   | 1 when 8-bit system bus accesses are supported.  | R      | Preset |

### 3.14.20 System Bus Address 31:0 (`sbaddress0`, at 0x39)

If `sbasize` is 0, then this register is not present.

When the system bus master is busy, writes to this register will set `sbbusyerror` and don’t do anything else.

If `sberror` is 0, `sbbusyerror` is 0, and `sbreadonaddr` is set then writes to this register start the following:

1. Set `sbbusy`.
2. Perform a bus read from the new value of `sbaddress`.
3. If the read succeeded and `sbautoincrement` is set, increment `sbaddress`.
4. Clear `sbbusy`.



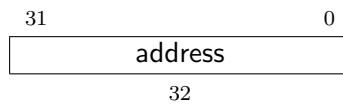


| Field   | Description  | Access | Reset |
|---------|--|--------|-------|
| address | Accesses bits 31:0 of the physical address in <code>sbaddress</code> . | R/W    | 0     |

### 3.14.21 System Bus Address 63:32 (`sbaddress1`, at 0x3a)

If `sbsize` is less than 33, then this register is not present.

When the system bus master is busy, writes to this register will set `sbbusyerror` and don't do anything else.

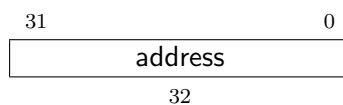


| Field   | Description   | Access | Reset |
|---------|---|--------|-------|
| address | Accesses bits 63:32 of the physical address in <code>sbaddress</code> (if the system address bus is that wide). | R/W    | 0     |

### 3.14.22 System Bus Address 95:64 (`sbaddress2`, at 0x3b)

If `sbsize` is less than 65, then this register is not present.

When the system bus master is busy, writes to this register will set `sbbusyerror` and don't do anything else.

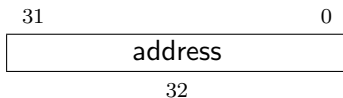


| Field   | Description   | Access | Reset |
|---------|---|--------|-------|
| address | Accesses bits 95:64 of the physical address in <code>sbaddress</code> (if the system address bus is that wide). | R/W    | 0     |

### 3.14.23 System Bus Address 127:96 (sbaddress3, at 0x37)

If `sbsize` is less than 97, then this register is not present.

When the system bus master is busy, writes to this register will set `sbbusyerror` and don't do anything else.



| Field   | Description  | Access | Reset |
|---------|--|--------|-------|
| address | Accesses bits 127:96 of the physical address in <code>sbaddress</code> (if the system address bus is that wide). | R/W    | 0     |

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

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

Any successful system bus read updates `sbdata`. If the width of the read access is less than the width of `sbdata`, the contents of the remaining high bits may take on any value.

If either `sberror` or `sbbusyerror` isn't 0 then accesses do nothing.

If the bus master is busy then accesses set `sbbusyerror`, and don't do anything else.

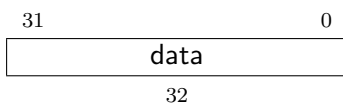
Writes to this register start the following:

1. Set `sbbusy`.
2. Perform a bus write of the new value of `sbdata` to `sbaddress`.
3. If the write succeeded and `sbautoincrement` is set, increment `sbaddress`.
4. Clear `sbbusy`.

Reads from this register start the following:

1. "Return" the data.
2. Set `sbbusy`.
3. If `sbreadondata` is set, perform a system bus read from the address contained in `sbaddress`, placing the result in `sbdata`.
4. If `sbautoincrement` is set, increment `sbaddress`.
5. Clear `sbbusy`.

Only `sbdatab0` has this behavior. The other `sbdatab` registers have no side effects. On systems that have buses wider than 32 bits, a debugger should access `sbdatab0` after accessing the other `sbdatab` registers.

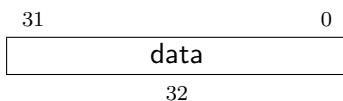


| Field             | Description                                  | Access | Reset |
|-------------------|--|--------|-------|
| <code>data</code> | Accesses bits 31:0 of <code>sbdatab</code> . | R/W    | 0     |

### 3.14.25 System Bus Data 63:32 (`sbdatab1`, at 0x3d)

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

If the bus master is busy then accesses set `sbbusyerror`, and don't do anything else.

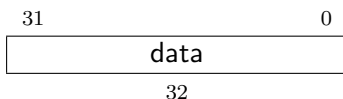


| Field             | Description   | Access | Reset |
|-------------------|---|--------|-------|
| <code>data</code> | Accesses bits 63:32 of <code>sbdatab</code> (if the system bus is that wide). | R/W    | 0     |

### 3.14.26 System Bus Data 95:64 (`sbdatab2`, at 0x3e)

This register only exists if `sbaccess128` is 1.

If the bus master is busy then accesses set `sbbusyerror`, and don't do anything else.

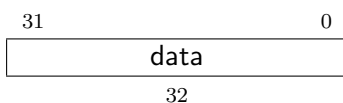


| Field       | Description   | Access | Reset |
|-------------|---|--------|-------|
| <b>data</b> | Accesses bits 95:64 of <code>sdata</code> (if the system bus is that wide). | R/W    | 0     |

### 3.14.27 System Bus Data 127:96 (`sdata3`, at `0x3f`)

This register only exists if `sbaccess128` is 1.

If the bus master is busy then accesses set `sbusyerror`, and don't do anything else.



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

### 3.14.28 Custom Features (`custom`, at `0x1f`)

This optional register may be used for non-standard features. Future version of the debug spec will not use this address.

### 3.14.29 Custom Features 0 (`custom0`, at `0x70`)

This optional register may be used for non-standard features. Future version of the debug spec will not use this address.

### 3.14.30 Custom Features 15 (`custom15`, at `0x7f`)

This optional register may be used for non-standard features. Future version of the debug spec will not use this address.

## 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.

In order to be compliant with this specification an implementation must implement everything described in this section that is not explicitly listed as optional.

### 4.1 Debug Mode

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

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

1. All operations are executed at machine mode privilege level, except that MPRV in `mstatus` may be ignored according to `mprven`.
2. All interrupts (including NMI) are masked.
3. Exceptions don't update any registers. That includes `cause`, `epc`, `tval`, `dpc`, and `mstatus`. They do end execution of the Program Buffer.
4. No action is taken if a trigger matches.
5. Counters may be stopped, depending on `stopcount` in `dcsr`.
6. Timers may be stopped, depending on `stoptime` in `dcsr`.
7. The `wfi` instruction acts as a `nop`.
8. Almost all instructions that change the privilege level have undefined behavior. This includes `ecall`, `mret`, `sret`, and `uret`. (To change the privilege level, the debugger can write `prv` in `dcsr`). The only exception is `ebreak`. When that is executed in Debug Mode, it halts the hart again but without updating `dpc` or `dcsr`.
9. Completing Program Buffer execution is considered output for the purpose of `fence` instructions.
10. All control transfer instructions may act as illegal instructions if their destination is in the Program Buffer. If one such instruction acts as an illegal instruction, all such instructions

must act as an illegal instruction.

11. All control transfer instructions may act as illegal instructions if their destination is outside the Program Buffer. If one such instruction acts as an illegal instruction, all such instructions must act as an illegal instruction.
12. Instructions that depend on the value of the PC (e.g. `auipc`) may act as illegal instructions.
13. Effective XLEN is DXLEN.

---

*In general, the debugger is expected to be able to simulate all the effects of MPRV. The exception is the case of Sv32 systems, which need MPRV functionality in order to access 34-bit physical addresses. Other systems are likely to tie `mprven` to 0.*

## 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.

---

*This is a behavior that debug users must be aware of. If they have a breakpoint set between a `lr` and `sc` pair, or are stepping through such code, the `sc` may never succeed. Fortunately in general use there will be very few instructions in such a sequence, and anybody debugging it will quickly notice that the reservation is not occurring. The solution in that case is to set a breakpoint on the first instruction after the `sc` and run to it. A higher level debugger may choose to automate this.*

## 4.3 Wait for Interrupt Instruction

If halt is requested while `wfi` is executing, then the hart must leave the stalled state, completing this instruction's execution, and then enter Debug Mode.

## 4.4 Single Step

A debugger can cause a halted hart to execute a single instruction and then re-enter Debug Mode by setting `step` before setting `resumereq`.

If executing or fetching that instruction causes an exception, Debug Mode is re-entered immediately after the PC is changed to the exception handler and the appropriate `tval` and `cause` registers are updated.

If executing or fetching the instruction causes a trigger to fire, Debug Mode is re-entered immediately after that trigger has fired. In that case `cause` is set to 2 (trigger) instead of 4 (single step). Whether the instruction is executed or not depends on the specific configuration of the trigger.

If the instruction that is executed causes the PC to change to an address where an instruction fetch causes an exception, that exception does not occur until the next time the hart is resumed.

Similarly, a trigger at the new address does not fire until the hart actually attempts to execute that instruction.

If the instruction being stepped over is `wfi` and would normally stall the hart, then instead the instruction is treated as `nop`.

## 4.5 Reset

If the halt signal (driven by the hart's halt request bit in the Debug Module) or `resethaltreq` are 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.

## 4.6 `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.

Executing `dret` outside of Debug Mode causes an illegal instruction exception.

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.

## 4.7 XLEN

While in Debug Mode, XLEN is DXLEN. It is up to the debugger to determine the XLEN during normal program execution (by looking at `misalr`) and to clearly communicate this to the user.

## 4.8 Core Debug Registers

The supported Core Debug Registers must be implemented for each hart that can be debugged. They are CSRs, accessible using the RISC-V `csr` opcodes and optionally also using abstract debug commands.

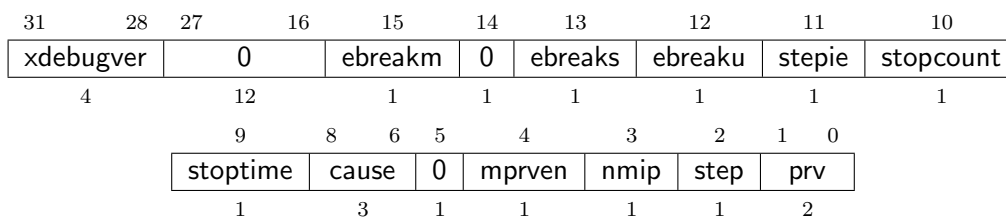
These registers are only accessible from Debug Mode.

Table 4.1: Core Debug Registers

| Address | Name  | Page |
|---------|---|------|
| 0x7b0   | Debug Control and Status ( <b>dcsr</b> )      | 44   |
| 0x7b1   | Debug PC ( <b>dpc</b> )                       | 46   |
| 0x7b2   | Debug Scratch Register 0 ( <b>dscratch0</b> ) | 47   |
| 0x7b3   | Debug Scratch Register 1 ( <b>dscratch1</b> ) | 47   |

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

**cause** priorities are assigned such that the least predictable events have the highest priority.



| 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   | 0: <b>ebreak</b> instructions in M-mode behave as described in the Privileged Spec.<br>1: <b>ebreak</b> instructions in M-mode enter Debug Mode.  | R/W    | 0      |
| ebreaks   | 0: <b>ebreak</b> instructions in S-mode behave as described in the Privileged Spec.<br>1: <b>ebreak</b> instructions in S-mode enter Debug Mode.  | R/W    | 0      |
| ebreaku   | 0: <b>ebreak</b> instructions in U-mode behave as described in the Privileged Spec.<br>1: <b>ebreak</b> instructions in U-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. In that case interrupt behavior can be emulated by the debugger.<br>The debugger must not change the value of this bit while the hart is running. | WARL   | 0      |

*Continued on next page*



| Field                  | Description   | Access | Reset  |
|------------------------|---|--------|--------|
| <code>stopcount</code> | 0: Increment counters as usual.<br>1: Don't increment any hart-local counters while in Debug Mode or on <code>ebreak</code> instructions that cause entry into Debug Mode. These counters include the <code>instret</code> CSR. On single-hart cores <code>cycle</code> should be stopped, but on multi-hart cores it must keep incrementing.<br>An implementation may hardwire this bit to 0 or 1.   | WARL   | Preset |
| <code>stoptime</code>  | 0: Increment timers as usual.<br>1: Don't increment any hart-local timers while in Debug Mode.<br>An implementation may hardwire this bit to 0 or 1.  | WARL   | Preset |
| <code>cause</code>     | Explains why Debug Mode was entered.<br>When there are multiple reasons to enter Debug Mode in a single cycle, hardware should set <code>cause</code> to the cause with the highest priority.<br>1: An <code>ebreak</code> instruction was executed. (priority 3)<br>2: The Trigger Module caused a breakpoint exception. (priority 4)<br>3: The debugger requested entry to Debug Mode using <code>haltreq</code> . (priority 1)<br>4: The hart single stepped because <code>step</code> was set. (priority 0, lowest)<br>5: The hart halted directly out of reset due to <code>resethaltreq</code> . It is also acceptable to report 3 when this happens. (priority 2)<br>6: The hart halted because it's part of a halt group. (priority 5, highest) Harts may report 3 for this cause instead.<br>Other values are reserved for future use. | R      | 0      |
| <code>mprven</code>    | 0: MPRV in <code>mstatus</code> is ignored in Debug Mode.<br>1: MPRV in <code>mstatus</code> takes effect in Debug Mode.<br>Implementing this bit is optional. It may be tied to either 0 or 1.   | WARL   | Preset |
| <code>nmip</code>      | When set, there is a Non-Maskable-Interrupt (NMI) pending for the hart.<br>Since an NMI can indicate a hardware error condition, reliable debugging may no longer be possible once this bit becomes set. This is implementation-dependent.  | R      | 0      |

*Continued on next page*

| Field | Description   | Access | Reset |
|-------|---|--------|-------|
| step  | 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. The debugger must not change the value of this bit while the hart is running.  | R/W    | 0     |
| prv   | Contains the privilege level the hart was operating in when Debug Mode was entered. The encoding is described in Table 4.5. A debugger can change this value to change the hart's privilege level when exiting Debug Mode.<br>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. | R/W    | 3     |

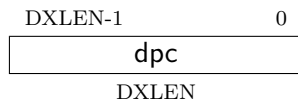
#### 4.8.2 Debug PC (dpc, at 0x7b1)

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 4.3.

Table 4.3: Virtual address in DPC upon Debug Mode Entry

| Cause               | Virtual Address in DPC   |
|---------------------|--|
| <code>ebreak</code> | Address of the <code>ebreak</code> 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 <code>timing</code> is 0, the address of the instruction which caused the trigger to fire. If <code>timing</code> 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.



### 4.8.3 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).

### 4.8.4 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).

## 4.9 Virtual Debug Registers

A virtual register is one that doesn't exist directly in the hardware, but that the debugger exposes as if it does. Debug software should implement them, but hardware can skip this section. Virtual registers exist to give users access to functionality that's not part of standard debuggers without requiring them to carefully modify debug registers while the debugger is also accessing those same registers.

Table 4.4: Virtual Core Debug Registers

| Address | Name                                     | Page               |
|---------|--|--------------------|
| virtual | Privilege Level ( <a href="#">priv</a> ) | <a href="#">47</a> |

### 4.9.1 Privilege Level ([priv](#), at [virtual](#))

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

This register contains [prv](#) 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.



Table 4.5: Privilege Level Encoding

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

| Field            | Description  | Access | Reset |
|------------------|--|--------|-------|
| <code>prv</code> | Contains the privilege level the hart was operating in when Debug Mode was entered. The encoding is described in Table 4.5, and matches the privilege level encoding from the Privileged Spec. 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.

A hart can be compliant with this specification without implementing any trigger functionality at all, but if it is implemented then it must conform to this section. If triggers aren't implemented, the CSRs may not exist at all and accessing them results in an illegal instruction exception.

Triggers do not fire while in Debug Mode.

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`. If this results in an illegal instruction exception, then there are no triggers implemented.
2. Read back `tselect` and check that it contains the written value. If not, exit the loop.
3. Read `tinfo`.
4. If that caused an exception, the debugger must read `tdata1` to discover the type. (If `type` is 0, this trigger doesn't exist. Exit the loop.)
5. If `info` is 1, this trigger doesn't exist. Exit the loop.
6. Otherwise, the selected trigger supports the types discovered in `info`.
7. Repeat, incrementing the value in `tselect`.

---

*The above algorithm reads back `tselect` so that implementations which have  $2^n$  triggers only need to implement  $n$  bits of `tselect`.*

*The algorithm checks `tinfo` and `type` in case the implementation has  $m$  bits of `tselect` but fewer than  $2^m$  triggers.*

It is possible for a trigger with the “enter Debug Mode” action (1) and another trigger with the “raise a breakpoint exception” action (0) to fire at the same time. The preferred behavior is to have both actions take place. It is implementation-dependent which of the two happens first. This ensures both that the presence of an external debugger doesn't affect execution and that a trigger

set by user code doesn't affect the external debugger. If this is not implemented, then the hart must enter Debug Mode and ignore the breakpoint exception. In the latter case, `hit` of the trigger whose action is 0 must still be set, giving a debugger an opportunity to handle this case. What happens with trace actions when triggers with different actions are also firing is left to the trace specification.

## 5.1 Native M-Mode Triggers

Triggers can be used for native debugging. On a fully featured system triggers will be set using `u` or `s`, and when firing they can cause a breakpoint exception to trap to a more privileged mode. It is possible to set triggers natively to fire in M mode as well. In that case there is no higher privilege mode to trap to. When such a trigger causes a breakpoint exception while already in a trap handler, this will leave the system unable to resume normal execution.

On full-featured systems this is a remote corner case that can probably be ignored. On systems that only implement M mode, however, it is recommended to implement one of two solutions to this problem. This way triggers can be useful for native debugging of even M mode code.

The simple solution is to have the hardware prevent triggers with `action=0` from firing while in M mode and while MIE in `mstatus` is 0. Its limitation is that interrupts might be disabled at other times when a user might want triggers to fire.

A more complex solution is to implement `mte` and `mpte` in `tcontrol`. This solution has the benefit that it only disables triggers during the trap handler.

A user setting M mode triggers that cause breakpoint exceptions will have to be aware of any problems that might come up with the particular system they are working on.

## 5.2 Trigger Registers

These registers are CSRs, accessible using the RISC-V `csr` opcodes and optionally also using abstract debug commands.

Most trigger functionality is optional. All `tdata` registers follow write-any-read-legal semantics. If a debugger writes an unsupported configuration, the register will read back a value that is supported (which may simply be a disabled trigger). This means that a debugger must always read back values it writes to `tdata` registers, unless it already knows already what is supported. Writes to one `tdata` register may not modify the contents of other `tdata` registers, nor the configuration of any trigger besides the one that is currently selected.

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.

In this section XLEN means MXLEN when in M-mode, and DXLEN when in Debug Mode. Note that this makes several of the fields in `tdata1` move around based on the current execution mode and value of MXLEN.

Table 5.1: `action` encoding

| Value | Description   |
|-------|---|
| 0     | Raise a breakpoint exception. (Used when software wants to use the trigger module without an external debugger attached.) |
| 1     | Enter Debug Mode. (Only supported when the trigger's <code>dmode</code> is 1.)  |
| 2 – 5 | Reserved for use by the trace specification.  |
| other | Reserved for future use.  |

Table 5.2: Trigger Registers

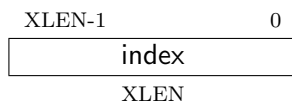
| Address | Name   | Page |
|---------|--|------|
| 0x7a0   | Trigger Select ( <code>tselect</code> )        | 51   |
| 0x7a1   | Trigger Data 1 ( <code>tdata1</code> )         | 52   |
| 0x7a1   | Match Control ( <code>mcontrol</code> )        | 55   |
| 0x7a1   | Instruction Count ( <code>icount</code> )      | 60   |
| 0x7a1   | Interrupt Trigger ( <code>itrigger</code> )    | 61   |
| 0x7a1   | Exception Trigger ( <code>etrigger</code> )    | 62   |
| 0x7a2   | Trigger Data 2 ( <code>tdata2</code> )         | 52   |
| 0x7a3   | Trigger Data 3 ( <code>tdata3</code> )         | 53   |
| 0x7a3   | Trigger Extra (RV32) ( <code>textra32</code> ) | 62   |
| 0x7a3   | Trigger Extra (RV64) ( <code>textra64</code> ) | 63   |
| 0x7a4   | Trigger Info ( <code>tinfo</code> )            | 53   |
| 0x7a5   | Trigger Control ( <code>tcontrol</code> )      | 53   |
| 0x7a8   | Machine Context ( <code>mcontext</code> )      | 54   |
| 0x7aa   | Supervisor Context ( <code>scontext</code> )   | 54   |

### 5.2.1 Trigger Select (`tselect`, at 0x7a0)

This register determines which trigger is accessible through the other trigger registers. It is optional if no triggers are implemented. The set of accessible triggers must start at 0, and be contiguous.

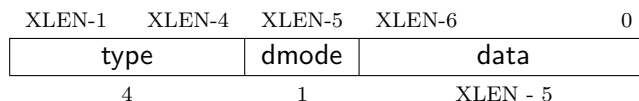
Writes of values greater than or equal to the number of supported triggers may result in a different value in this register than what was written. To verify that what they wrote is a valid index, debuggers can read back the value and check that `tselect` holds what they wrote.

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



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

This register is optional if no triggers are implemented.



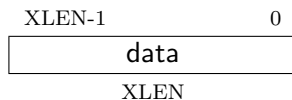
| Field | Description   | Access | Reset  |
|-------|---|--------|--------|
| type  | 0: There is no trigger at this <a href="#">tselect</a> .<br>1: The trigger is a legacy SiFive address match trigger. These should not be implemented and aren't further documented here.<br>2: The trigger is an address/data 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>4: The trigger is an interrupt trigger. The remaining bits in this register act as described in <a href="#">itrigger</a> .<br>5: The trigger is an exception trigger. The remaining bits in this register act as described in <a href="#">etrigger</a> .<br>12–14: These trigger types are available for non-standard use.<br>15: This trigger exists (so enumeration shouldn't terminate), but is not currently available.<br>Other values are reserved for future use. | R/W    | 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 |

### 5.2.3 Trigger Data 2 (tdata2, at 0x7a2)

Trigger-specific data. It is optional if no implemented triggers use it.

If XLEN is less than DXLEN, writes to this register are sign-extended.

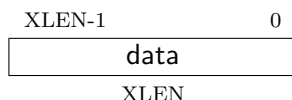




### 5.2.4 Trigger Data 3 (tdata3, at 0x7a3)

Trigger-specific data. It is optional if no implemented triggers use it.

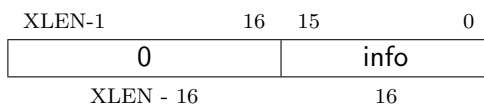
If XLEN is less than DXLEN, writes to this register are sign-extended.



### 5.2.5 Trigger Info (tinfo, at 0x7a4)

This register is optional if no triggers are implemented, or if `type` is not writable. In this case the debugger can read the only supported type from `tdata1`.

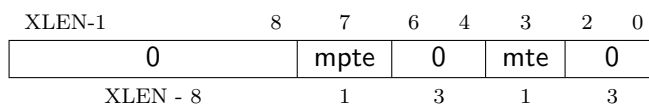
This entire register is read-only.



| Field | Description   | Access | Reset  |
|-------|---|--------|--------|
| info  | One bit for each possible <code>type</code> enumerated in <code>tdata1</code> . Bit N corresponds to type N. If the bit is set, then that type is supported by the currently selected trigger.<br>If the currently selected trigger doesn't exist, this field contains 1. | R      | Preset |

### 5.2.6 Trigger Control (tcontrol, at 0x7a5)

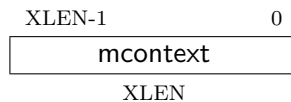
This optional register is one solution to a problem regarding triggers with `action=0` firing in M-mode trap handlers. See Section 5.1 for more details.



| Field | Description   | Access | Reset |
|-------|---|--------|-------|
| mpte  | M-mode previous trigger enable field.<br>When a trap into M-mode is taken, <b>mpte</b> is set to the value of <b>mte</b> .  | R/W    | 0     |
| mte   | M-mode trigger enable field.<br>0: Triggers with action=0 do not match/fire while the hart is in M-mode.<br>1: Triggers do match/fire while the hart is in M-mode.<br>When a trap into M-mode is taken, <b>mte</b> is set to 0. When <b>mret</b> is executed, <b>mte</b> is set to the value of <b>mpte</b> . | R/W    | 0     |

### 5.2.7 Machine Context (mcontext, at 0x7a8)

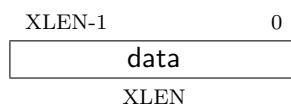
This optional register is only writable in M mode and Debug Mode.



| Field    | Description  | Access | Reset |
|----------|--|--------|-------|
| mcontext | Machine mode software can write a context number to this register, which can be used to set triggers that only fire in that specific context.<br>An implementation may tie any number of upper bits in this field to 0. It's recommended to implement no more than 6 bits on RV32, and 13 on RV64. | R/W    | 0     |

### 5.2.8 Supervisor Context (scontext, at 0x7aa)

This optional register is only writable in S mode, M mode and Debug Mode.



| Field | Description  | Access | Reset |
|-------|--|--------|-------|
| data  | Supervisor mode software can write a context number to this register, which can be used to set triggers that only fire in that specific context. An implementation may tie any number of high bits in this field to 0. It's recommended to implement no more than 16 bits on RV32, and 34 on RV64. | R/W    | 0     |

### 5.2.9 Match Control (mcontrol, at 0x7a1)

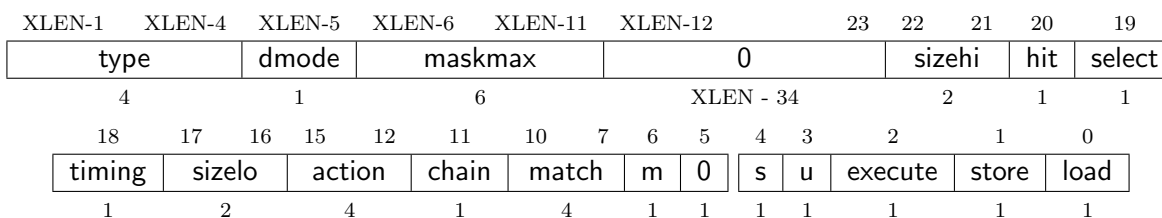
This register is accessible as `tdata1` when `type` is 2.

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 5.8 suggests timings for the best user experience.

Table 5.8: 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                   |

This trigger type may be limited to address comparisons (`select` is always 0) only. If that is the case, then `tdata2` must be able to hold all valid virtual addresses but it need not be capable of holding other values.



| Field   | Description   | Access | Reset  |
|---------|---|--------|--------|
| maskmax | Specifies the largest naturally aligned powers-of-two (NAPOT) range supported by the hardware when <code>match</code> is 1. 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 |
| sizehi  | This field only exists if XLEN is greater than 32. In that case it extends <code>size</code> . If it does not exist then hardware operates as if the field contains 0.  | R/W    | 0      |
| hit     | If this optional bit is implemented, the hardware sets it when this trigger matches. The trigger's user can set or clear it at any time. It is used to determine which trigger(s) matched. If the bit is not implemented, it is always 0 and writing it has no effect.  | R/W    | 0      |
| select  | 0: Perform a match on the lowest virtual address of the access. In addition, it is recommended that the trigger also fires if any of the other accessed virtual addresses match. (E.g. on a 32-bit read from 0x4000, the lowest address is 0x4000 and the other addresses are 0x4001, 0x4002, and 0x4003.)<br>1: Perform a match on the data value loaded or stored, or the instruction executed. | R/W    | 0      |

*Continued on next page*

| Field  | Description   | Access | Reset |
|--------|---|--------|-------|
| timing | <p>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.</p> <p>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.</p> <p>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.</p> <p>Data load triggers with <a href="#">timing</a> of 0 will result in the same load happening again when the debugger lets the hart run. For data load triggers, debuggers must first attempt to set the breakpoint with <a href="#">timing</a> of 1.</p> <p>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).</p> <p>If a trigger with <a href="#">timing</a> of 0 matches, it is implementation-dependent whether that prevents a trigger with <a href="#">timing</a> of 1 matching as well.</p> | R/W    | 0     |

*Continued on next page*

| Field              | Description   | Access | Reset |
|--------------------|---|--------|-------|
| size <sub>lo</sub> | <p>This field contains the 2 low bits of <a href="#">size</a>. The high bits come from <a href="#">size<sub>hi</sub></a>. The combined value is interpreted as follows:</p> <p>0: The trigger will attempt to match against an access of any size. The behavior is only well-defined if <code>select = 0</code>, or if the access size is XLEN.</p> <p>1: The trigger will only match against 8-bit memory accesses.</p> <p>2: The trigger will only match against 16-bit memory accesses or execution of 16-bit instructions.</p> <p>3: The trigger will only match against 32-bit memory accesses or execution of 32-bit instructions.</p> <p>4: The trigger will only match against execution of 48-bit instructions.</p> <p>5: The trigger will only match against 64-bit memory accesses or execution of 64-bit instructions.</p> <p>6: The trigger will only match against execution of 80-bit instructions.</p> <p>7: The trigger will only match against execution of 96-bit instructions.</p> <p>8: The trigger will only match against execution of 112-bit instructions.</p> <p>9: The trigger will only match against 128-bit memory accesses or execution of 128-bit instructions.</p> | R/W    | 0     |
| action             | The action to take when the trigger fires. The values are explained in <a href="#">Table 5.1</a> .  | R/W    | 0     |

*Continued on next page*

| Field | Description   | Access | Reset |
|-------|---|--------|-------|
| 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> <p>A trigger chain starts on the first trigger with <code>chain = 1</code> after a trigger with <code>chain = 0</code>, or simply on the first trigger if that has <code>chain = 1</code>. It ends on the first trigger after that which has <code>chain = 0</code>. This final trigger is part of the chain. The action on all but the final trigger is ignored. The action on that final trigger will be taken if and only if all the triggers in the chain match at the same time. Because <code>chain</code> affects the next trigger, hardware must zero it in writes to <code>mcontrol</code> that set <code>dmode</code> to 0 if the next trigger has <code>dmode</code> of 1. In addition hardware should ignore writes to <code>mcontrol</code> that set <code>dmode</code> to 1 if the previous trigger has both <code>dmode</code> of 0 and <code>chain</code> of 1. Debuggers must avoid the latter case by checking <code>chain</code> on the previous trigger if they're writing <code>mcontrol</code>.</p> <p>Implementations that wish to limit the maximum length of a trigger chain (eg. to meet timing requirements) may do so by zeroing <code>chain</code> in writes to <code>mcontrol</code> that would make the chain too long.</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>. Debuggers should only write values to <code>tdata2</code> such that <math>M + \text{maskmax} \geq \text{XLEN}</math>, otherwise it's undefined on what conditions the trigger will fire.</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     |
| m     | When set, enable this trigger in M-mode.  | R/W    | 0     |

Continued on next page

| Field          | Description  | Access | Reset |
|----------------|--|--------|-------|
| <b>s</b>       | When set, enable this trigger in S-mode.   | R/W    | 0     |
| <b>u</b>       | When set, enable this trigger in U-mode.   | R/W    | 0     |
| <b>execute</b> | When set, the trigger fires on the virtual address or opcode of an instruction that is executed. | R/W    | 0     |
| <b>store</b>   | When set, the trigger fires on the virtual address or data of a store.                           | R/W    | 0     |
| <b>load</b>    | When set, the trigger fires on the virtual address or data of a load.                            | R/W    | 0     |

### 5.2.10 Instruction Count (`icount`, at `0x7a1`)

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

---

*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`, `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` 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 | 25 | 24         | 23           | 10 | 9        | 8        | 7        | 6        | 5             | 0 |
| <b>type</b> | <b>dmode</b> | <b>0</b>  |        |    | <b>hit</b> | <b>count</b> |    | <b>m</b> | <b>0</b> | <b>s</b> | <b>u</b> | <b>action</b> |   |
| 4           | 1            | XLEN - 30 |        |    | 1          | 14           |    | 1        | 1        | 1        | 1        | 6             |   |

| Field        | Description   | Access | Reset |
|--------------|---|--------|-------|
| <b>hit</b>   | If this optional bit is implemented, the hardware sets it when this trigger matches. The trigger's user can set or clear it at any time. It is used to determine which trigger(s) matched. If the bit is not implemented, it is always 0 and writing it has no effect.  | R/W    | 0     |
| <b>count</b> | When <code>count</code> 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>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     |
| <b>m</b>     | When set, every instruction completed or exception taken in M-mode decrements <code>count</code> by 1.  | R/W    | 0     |
| <b>s</b>     | When set, every instruction completed or exception taken in S-mode decrements <code>count</code> by 1.  | R/W    | 0     |

*Continued on next page*



| Field         | Description  | Access | Reset |
|---------------|--|--------|-------|
| <b>u</b>      | When set, every instruction completed or exception taken in U-mode decrements <b>count</b> by 1. | R/W    | 0     |
| <b>action</b> | The action to take when the trigger fires. The values are explained in Table 5.1.                | R/W    | 0     |

### 5.2.11 Interrupt Trigger (**itrigger**, at 0x7a1)

This register is accessible as **tdata1** when **type** is 4.

This trigger may fire on any of the interrupts configurable in **mie** (described in the Privileged Spec). The interrupts to fire on are configured by setting the same bit in **tdata2** as would be set in **mie** to enable the interrupt.

Hardware may only support a subset of interrupts for this trigger. A debugger must read back **tdata2** after writing it to confirm the requested functionality is actually supported.

The trigger only fires if the hart takes a trap because of the interrupt. (E.g. it does not fire when a timer interrupt occurs but that interrupt is not enabled in **mie**.)

When the trigger fires, all CSRs are updated as defined by the Privileged Spec, and the requested action is taken just before the first instruction of the interrupt/exception handler is executed.

|        |        |        |           |        |    |   |   |   |   |        |   |
|--------|--------|--------|-----------|--------|----|---|---|---|---|--------|---|
| XLEN-1 | XLEN-4 | XLEN-5 | XLEN-6    | XLEN-7 | 10 | 9 | 8 | 7 | 6 | 5      | 0 |
| type   | dmode  | hit    | 0         |        |    | m | 0 | s | u | action |   |
| 4      | 1      | 1      | XLEN - 16 |        |    | 1 | 1 | 1 | 1 | 6      |   |

| Field         | Description  | Access | Reset |
|---------------|--|--------|-------|
| <b>hit</b>    | If this optional bit is implemented, the hardware sets it when this trigger matches. The trigger's user can set or clear it at any time. It is used to determine which trigger(s) matched. If the bit is not implemented, it is always 0 and writing it has no effect. | R/W    | 0     |
| <b>m</b>      | When set, enable this trigger for interrupts that are taken from M mode.   | R/W    | 0     |
| <b>s</b>      | When set, enable this trigger for interrupts that are taken from S mode.   | R/W    | 0     |
| <b>u</b>      | When set, enable this trigger for interrupts that are taken from U mode.   | R/W    | 0     |
| <b>action</b> | The action to take when the trigger fires. The values are explained in Table 5.1.  | R/W    | 0     |

### 5.2.12 Exception Trigger (etrigger, at 0x7a1)

This register is accessible as `tdata1` when `type` is 5.

This trigger may fire on up to XLEN of the Exception Codes defined in `mcause` (described in the Privileged Spec, with Interrupt=0). Those causes are configured by writing the corresponding bit in `tdata2`. (E.g. to trap on an illegal instruction, the debugger sets bit 2 in `tdata2`.)

Hardware may support only a subset of exceptions. A debugger must read back `tdata2` after writing it to confirm the requested functionality is actually supported.

When the trigger fires, all CSRs are updated as defined by the Privileged Spec, and the requested action is taken just before the first instruction of the interrupt/exception handler is executed.

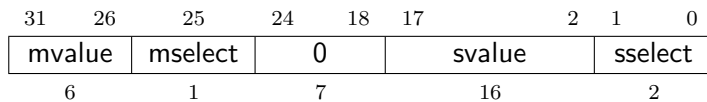
|        |        |        |           |        |    |     |   |   |   |   |        |   |
|--------|--------|--------|-----------|--------|----|-----|---|---|---|---|--------|---|
| XLEN-1 | XLEN-4 | XLEN-5 | XLEN-6    | XLEN-7 | 11 | 10  | 9 | 8 | 7 | 6 | 5      | 0 |
| type   | dmode  | hit    | 0         |        |    | nmi | m | 0 | s | u | action |   |
| 4      | 1      | 1      | XLEN - 17 |        |    | 1   | 1 | 1 | 1 | 1 | 6      |   |

| Field  | Description  | Access | Reset |
|--------|--|--------|-------|
| hit    | If this optional bit is implemented, the hardware sets it when this trigger matches. The trigger's user can set or clear it at any time. It is used to determine which trigger(s) matched. If the bit is not implemented, it is always 0 and writing it has no effect. | R/W    | 0     |
| nmi    | When this optional bit is set, non-maskable interrupts cause this trigger to fire, regardless of the values of <code>m</code> , <code>s</code> , and <code>u</code> .  | R/W    | 0     |
| m      | When set, enable this trigger for exceptions that are taken from M mode.   | R/W    | 0     |
| s      | When set, enable this trigger for exceptions that are taken from S mode.   | R/W    | 0     |
| u      | When set, enable this trigger for exceptions that are taken from U mode.   | R/W    | 0     |
| action | The action to take when the trigger fires. The values are explained in Table 5.1.  | R/W    | 0     |

### 5.2.13 Trigger Extra (RV32) (textra32, at 0x7a3)

This register is accessible as `tdata3` when `type` is 2, 3, 4, or 5.

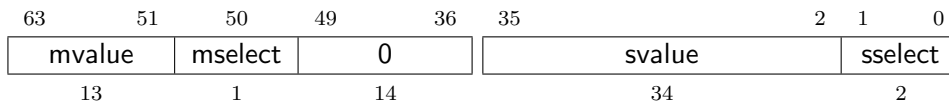
All functionality in this register is optional. The `value` bits may tie any number of upper bits to 0. The `select` bits may only support 0 (ignore).



| Field   | Description  | Access | Reset |
|---------|--|--------|-------|
| mvalue  | Data used together with <a href="#">mselect</a> .  | R/W    | 0     |
| mselect | 0: Ignore <a href="#">mvalue</a> .<br>1: This trigger will only match if the low bits of <a href="#">mcontext</a> equal <a href="#">mvalue</a> .   | WARL   | 0     |
| svalue  | Data used together with <a href="#">sselect</a> .  | R/W    | 0     |
| sselect | 0: Ignore <a href="#">svalue</a> .<br>1: This trigger will only match if the low bits of <a href="#">scontext</a> equal <a href="#">svalue</a> .<br>2: This trigger will only match if ASID in <a href="#">satp</a> equals the lower ASIDMAX (defined in the Privileged Spec) bits of <a href="#">svalue</a> . | WARL   | 0     |

#### 5.2.14 Trigger Extra (RV64) ([textra64](#), at [0x7a3](#))

This is the layout of [textra](#) if XLEN is 64. The fields are defined above, in [textra32](#).



## Chapter 6

# Debug Transport Module (DTM)

Debug Transport Modules provide access to the DM over one or more transports (e.g. 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.

An implementation can be compliant with this specification without implementing any of this section. In that case it must be advertised as conforming to “RISC-V Debug Specification 0.14.0-DRAFT, with custom DTM.” If the JTAG DTM described here is implemented, it must be advertised as conforming to the “RISC-V Debug Specification 0.14.0-DRAFT, with JTAG DTM.”

## 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, except for the 0x1f encoding of BYPASS, which must be extended with 1's in the 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          |      |
| 0x01    | IDCODE                                       | To identify a specific silicon version |      |
| 0x10    | DTM Control and Status ( <i>dtmcs</i> )      | For Debugging                          | 66   |
| 0x11    | Debug Module Interface Access ( <i>dmi</i> ) | For Debugging                          | 67   |
| 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.

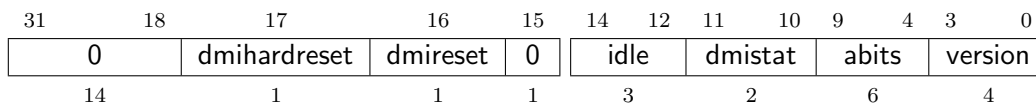
This entire register is read-only.

|         |    |            |    |         |   |   |
|---------|----|------------|----|---------|---|---|
| 31      | 28 | 27         | 12 | 11      | 1 | 0 |
| Version |    | PartNumber |    | Manufld |   | 1 |
| 4       |    | 16         |    | 11      |   | 1 |

| 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/manufacturer of this part. Bits 6:0 must be bits 6:0 of the designer/manufacturer'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.



| 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     | -      |
| dmireset     | Writing 1 to this bit clears the sticky error state and allows the DTM to retry or complete the previous transaction.   | W1     | -      |
| 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*

| Field   | Description   | Access | Reset  |
|---------|---|--------|--------|
| dmistat | 0: No error.<br>1: Reserved. Interpret the same as 2.<br>2: An operation failed (resulted in <code>op</code> of 2).<br>3: An operation was attempted while a DMI access was still in progress (resulted in <code>op</code> of 3). | R      | 0      |
| abits   | The size of <code>address</code> in <code>dmi</code> .  | R      | Preset |
| version | 0: Version described in spec version 0.11.<br>1: Version described in spec version 0.13.<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.

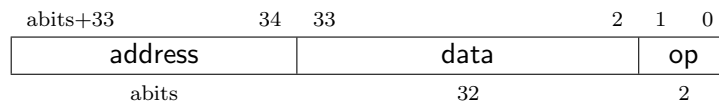
In Capture-DR, the DTM updates `data` with the result from that operation, updating `op` if the current `op` isn't sticky.

See Section B.1 and Table ?? 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 |
|----------------------|--|--------|-------|
| <code>address</code> | Address used for DMI access. In Update-DR this value is used to access the DM over the DMI.  | R/W    | 0     |
| <code>data</code>    | 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*

| Field | Description   | Access | Reset |
|-------|---|--------|-------|
| op    | <p>When the debugger writes this field, it has the following meaning:</p> <p>0: Ignore <code>data</code> and <code>address</code>. (nop)<br/>           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 <code>address</code>. (read)<br/>           2: Write <code>data</code> to <code>address</code>. (write)<br/>           3: Reserved.</p> <p>When the debugger reads this field, it means the following:</p> <p>0: The previous operation completed successfully.<br/>           1: Reserved.<br/>           2: A previous operation failed. The data scanned into <code>dmi</code> in this access will be ignored. This status is sticky and can be cleared by writing <code>dmireset</code> in <code>dtmcs</code>.<br/>           This indicates that the DM itself responded with an error. There are no specified cases in which the DM would respond with an error, and DMI is not required to support returning errors.<br/>           3: An operation was attempted while a DMI request is still in progress. The data scanned into <code>dmi</code> in this access will be ignored. This status is sticky and can be cleared by writing <code>dmireset</code> in <code>dtmcs</code>. 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.</p> | R/W    | 0     |

### 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.

0  
0  
 1



### 6.1.7 Recommended JTAG Connector

To make it easy to acquire debug hardware, this spec recommends a connector that is compatible with the MIPI-10 .05 inch connector specification, as described in the MIPI Alliance Recommendation for Debug and Trace Connectors, Version 1.10.00, 16 March 2011.

The connector has .05 inch spacing, gold-plated male header with .016 inch thick hardened copper or beryllium bronze square posts (SAMTEC FTSH 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 6.5. The function of each pin is described in Table 6.7.

Table 6.5: MIPI-10 Connector Diagram

|            |   |    |        |
|------------|---|----|--------|
| VREF DEBUG | 1 | 2  | TMS    |
| GND        | 3 | 4  | TCK    |
| GND        | 5 | 6  | TDO    |
| GND or KEY | 7 | 8  | TDI    |
| GND        | 9 | 10 | nRESET |

If a platform requires nTRST then it is permissible to reuse the nRESET pin as the nTRST signal. If a platform requires both system reset and TAP reset, the MIPI-20 connector should be used. Its physical connector is virtually identical to MIPI-10, except that it's twice as long, supporting twice as many pins. Its connector is show in Table 6.6.

Table 6.6: MIPI-20 Connector Diagram

|            |    |    |          |
|------------|----|----|----------|
| VREF DEBUG | 1  | 2  | TMS      |
| GND        | 3  | 4  | TCK      |
| GND        | 5  | 6  | TDO      |
| GND or KEY | 7  | 8  | TDI      |
| GND        | 9  | 10 | nRESET   |
| GND        | 11 | 12 | RTCK     |
| GND        | 13 | 14 | nTRST_PD |
| GND        | 15 | 16 | nTRST    |
| GND        | 17 | 18 | DBGRRQ   |
| GND        | 19 | 20 | DBGACK   |

The same connectors can be used for 2-wire cJTAG. In that case TMS is used for TMS0, and TCK is used for TCK0.

Table 6.7: JTAG Connector Pinout

|    |            |   |
|----|------------|---|
| 1  | VREF DEBUG | Reference voltage for logic high.   |
| 2  | TMS        | JTAG TMS signal, driven by the debug adapter.   |
| 4  | TCK        | JTAG TCK signal, driven by the debug adapter.   |
| 6  | TDO        | JTAG TDO signal, driven by the target.  |
| 7  | GND or KEY | This pin may be cut on the male and plugged on the female header to ensure the header is always plugged in correctly. It is, however, recommended to use this pin as an additional ground, to allow for fastest TCK speeds. A shrouded connector should be used to prevent the cable from being plugged in incorrectly.   |
| 8  | TDI        | JTAG TDI signal, driven by the debug adapter.   |
| 10 | nRESET     | 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. This pin is optional but strongly encouraged.<br><br>If necessary, this pin could be used as nTRST instead. nRESET should never be connected to the TAP reset, otherwise the debugger might not be able to debug through a reset to discover the cause of a crash or to maintain execution control after the reset. |
| 12 | RTCK       | Return test clock, driven by the target. A target may relay the TCK signal here once it has processed it, allowing a debugger to adjust its TCK frequency in response.  |
| 14 | nTRST_PD   | Test reset pull-down (optional), driven by the debug adapter. Same function as nTRST, but with pull-down resistor on target.  |
| 16 | nTRST      | Test reset (optional), driven by the debug adapter. Used to reset the JTAG TAP Controller.  |
| 18 | TRIGIN     | Not used, driven low by the debug adapter.  |
| 20 | TRIGOUT    | Not used, driven by the target.   |

# 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 hart execution pipeline.

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

Memory is accessed using the Abstract Access Memory command or through System Bus Access.

This implementation could allow a debugger to collect information from the hart even when that hart is unable to execute instructions.

### 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. It uses the hart's existing pipeline and ability to execute from arbitrary memory locations to avoid modifications to a hart's datapath.

When the halt request bit is set, the Debug Module raises a special interrupt to the selected harts. 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 DM 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, the DM populates these instructions as `nops`. If `execute` is set, execution continues to the debugger-controlled Program Buffer, otherwise the DM 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 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. This address is considered I/O for `fence` instructions (see #9 on page 41). 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 hart 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 Section 6.1. 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 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 Checking for Halted Harts

A user will want to know as quickly as possible when a hart is halted (e.g. due to a breakpoint). To efficiently determine which harts are halted when there are many harts, the debugger uses the `haltsum` registers. Assuming the maximum number of harts exist, first it checks `haltsum3`. For each bit set there, it writes `hartsel`, and checks `haltsum2`. This process repeats through `haltsum1` and `haltsum0`. Depending on how many harts exist, the process should start at one of the lower `haltsum` registers.

## 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. Then it can clear `haltreq` to 0, or leave it high to catch a hart that resets while halted.

## B.4 Running

First, the debugger should restore any registers that it has overwritten. Then 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`. Harts might halt very quickly after resuming (e.g. 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 hart run. The hart 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>aarsize = 2, transfer, regno = 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>aarsize = 2, transfer, write, regno = 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>aarsize = 2, postexec, transfer, write, regno = 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, regno = 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

With system bus access, addresses are physical system bus addresses.

Read a word from memory using system bus access:

| Op    | Address                 | Value                                   | Comment                |
|-------|-------------------------|---|------------------------|
| Write | <code>sbcs</code>       | <code>sbaccess = 2, sbreadonaddr</code> | Setup                  |
| Write | <code>sbaddress0</code> | address                                 |                        |
| Read  | <code>sbdata0</code>    | -                                       | Value read from memory |

Read block of memory using system bus access:

| Op    | Address                 | Value  | Comment                                     |
|-------|-------------------------|--|---|
| Write | <code>sbc</code>        | <code>sbaccess = 2, sbreadonaddr, sbreadondata, sbautoincrement</code> | Turn on autoread and autoincrement          |
| Write | <code>sbaddress0</code> | address  | Writing address triggers read and increment |
| Read  | <code>sbd</code>        | -  | Value read from memory                      |
| Read  | <code>sbd</code>        | -  | Next value read from memory                 |
| ...   | ...                     | ...  | ...   |
| Write | <code>sbc</code>        | 0  | Disable autoread                            |
| Read  | <code>sbd</code>        | -  | Get last value read from memory.            |

### B.7.2 Using Program Buffer

Through the Program Buffer, the hart performs the memory accesses. Addresses are physical or virtual (depending on `mprven` and other system configuration).

Read a word from memory using program buffer:

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

Read block of memory using program buffer:



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

### B.7.3 Using Abstract Memory Access

Abstract memory accesses act as if they are performed by the hart, although the actual implementation may differ.

Read a word from memory using abstract memory access:

| Op    | Address              | Value                              | Comment                |
|-------|----------------------|------------------------------------|------------------------|
| Write | <code>data1</code>   | address                            |                        |
| Write | <code>command</code> | <code>cmdtype=2, aamsize =2</code> |                        |
| Read  | <code>data0</code>   | -                                  | Value read from memory |

Read block of memory using abstract memory access:

| Op    | Address                   | Value   | Comment  |
|-------|---------------------------|---|--|
| Write | <code>abstractauto</code> | 1   | Re-execute the command when <code>data0</code> is accessed |
| Write | <code>data1</code>        | address   |  |
| Write | <code>command</code>      | <code>cmdtype=2, aamsize =2, aampostincrement =1</code> |  |
| Read  | <code>data0</code>        | -   | Read value, and trigger reading of next address            |
| ...   | ...                       | ...   | ...  |
| Write | <code>abstractauto</code> | 0   | Disable auto-exec  |
| Read  | <code>data0</code>        | -   | Get last value read from memory.                           |

## B.8 Writing Memory

### B.8.1 Using System Bus Access

With system bus access, addresses are physical system bus addresses.

Write a word to memory using system bus access:

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

Write a block of memory using system bus access:

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

### B.8.2 Using Program Buffer

Through the Program Buffer, the hart performs the memory accesses. Addresses are physical or virtual (depending on `mprven` and other system configuration).

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>    | address                                      |   |
| Write | <code>command</code>  | <code>write, regno = 0x1008</code>           | Write <code>s0</code>                               |
| Write | <code>data0</code>    | value  |   |
| Write | <code>command</code>  | <code>write, postexec, regno = 0x1009</code> | Write <code>s1</code> , then execute program buffer |

Write block of memory using program buffer:

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

### B.8.3 Using Abstract Memory Access

Abstract memory accesses act as if they are performed by the hart, although the actual implementation may differ.

Write a word to memory using abstract memory access:

| Op    | Address              | Value                                       | Comment |
|-------|----------------------|---|---------|
| Write | <code>data1</code>   | address                                     |         |
| Write | <code>data0</code>   | value                                       |         |
| Write | <code>command</code> | <code>cmdtype=2, aamsize =2, write=1</code> |         |

Write a block of memory using abstract memory access:

| Op    | Address                   | Value  | Comment  |
|-------|---------------------------|--|--|
| Write | <code>data1</code>        | address  |  |
| Write | <code>data0</code>        | value0   |  |
| Write | <code>command</code>      | <code>cmdtype=2, aamsize =2, write=1, aampostincrement =1</code> |  |
| Write | <code>abstractauto</code> | 1  | Re-execute the command when <code>data0</code> is accessed |
| Write | <code>data0</code>        | value1   |  |
| Write | <code>data0</code>        | value2   |  |
| ...   | ...                       | ...  | ...  |
| Write | <code>data0</code>        | valueN   |  |
| Write | <code>abstractauto</code> | 0  | Disable auto-exec  |

## B.9 Triggers

A debugger can use hardware triggers to halt a hart when a certain event occurs. Below are some examples, but as there is no requirement on the number of features of the triggers implemented by a hart, these examples may not be applicable to all implementations. When a debugger wants to set a trigger, it writes the desired configuration, and then reads back to see if that configuration is supported.

Enter Debug Mode just before the instruction at 0x80001234 is executed, to be used as an instruction breakpoint in ROM:

|                     |            |   |
|---------------------|------------|---|
| <code>tdata1</code> | 0x105c     | action=1, match=0, m=1, s=1, u=1, execute=1 |
| <code>tdata2</code> | 0x80001234 | address                                     |

Enter Debug Mode right after the value at 0x80007f80 is read:

|                     |            |   |
|---------------------|------------|---|
| <code>tdata1</code> | 0x4159     | timing=1, action=1, match=0, m=1, s=1, u=1,<br>load=1 |
| <code>tdata2</code> | 0x80007f80 | address   |

Enter Debug Mode right before a write to an address between 0x80007c80 and 0x80007cef (inclusive):

|                       |            |   |
|-----------------------|------------|---|
| <code>tdata1</code> 0 | 0x195a     | action=1, chain=1, match=2, m=1, s=1, u=1,<br>store=1 |
| <code>tdata2</code> 0 | 0x80007c80 | start address (inclusive)                             |
| <code>tdata1</code> 1 | 0x11da     | action=1, match=3, m=1, s=1, u=1, store=1             |
| <code>tdata2</code> 1 | 0x80007cf0 | end address (exclusive)                               |

Enter Debug Mode right before a write to an address between 0x81230000 and 0x8123fff (inclusive):

|                     |            |  |
|---------------------|------------|--|
| <code>tdata1</code> | 0x10da     | action=1, match=1, m=1, s=1, u=1, store=1        |
| <code>tdata2</code> | 0x81237fff | 16 bits to match exactly, then 0, then all ones. |

Enter Debug Mode right after a read from an address between 0x86753090 and 0x8675309f or between 0x96753090 and 0x9675309f (inclusive):

|                       |            |  |
|-----------------------|------------|--|
| <code>tdata1</code> 0 | 0x41a59    | timing=1, action=1, chain=1, match=4, m=1, s=1,<br>u=1, load=1 |
| <code>tdata2</code> 0 | 0xfff03090 | Mask for low half, then match for low half                     |
| <code>tdata1</code> 1 | 0x412d9    | timing=1, action=1, match=5, m=1, s=1, u=1,<br>load=1          |
| <code>tdata2</code> 1 | 0xefff8675 | Mask for high half, then match for high half                   |

## B.10 Handling Exceptions

Generally the debugger can avoid exceptions by being careful with the programs it writes. Sometimes they are unavoidable though, e.g. 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.11 Quick Access

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 examples here use the pseudo-op `transfer dest, src` to represent all these options.

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

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

This shows an example of setting the `m` bit in `mcontrol` to enable a hardware breakpoint in M-mode. Similar quick access instructions could have been used previously to configure the trigger that is being enabled here:

| Op    | Address               | Value                              | Comment                                 |
|-------|-----------------------|------------------------------------|---|
| Write | <code>progbuf0</code> | <code>transfer arg0, s0</code>     | Save <code>s0</code>                    |
| Write | <code>progbuf1</code> | <code>li s0, (1 &lt;&lt; 6)</code> | Form the mask for <code>m</code> bit    |
| Write | <code>progbuf2</code> | <code>csrrs x0, tdata1, s0</code>  | Apply the mask to <code>mcontrol</code> |
| Write | <code>progbuf3</code> | <code>transfer s0, arg2</code>     | Restore <code>s0</code>                 |
| Write | <code>progbuf4</code> | <code>ebreak</code>                |   |
| Write | <code>command</code>  | <code>0x10000000</code>            | Perform quick access                    |

# Index

aampostincrement, 16  
aamsize, 16  
aamvirtual, 16  
aarpostincrement, 14  
aarsize, 14  
abits, 67  
abstractauto, 30  
abstractcs, 28  
Access Memory, 15  
Access Register, 13  
ackhavereset, 24  
action, 58, 61, 62  
address, 37, 38, 67  
allhalted, 22  
allhavereset, 22  
allnonexistent, 22  
allresumeack, 22  
allrunning, 22  
allunavail, 22  
anyhalted, 22  
anyhavereset, 22  
anynonexistent, 22  
anyresumeack, 22  
anyrunning, 22  
anyunavail, 22  
authbusy, 23  
authdata, 32  
authenticated, 23  
autoexecdata, 30  
autoexecprogbuf, 30  
  
busy, 28  
BYPASS, 68  
  
cause, 45  
chain, 59  
clrresethaltreq, 25  
cmderr, 29  
cmdtype, 14–16, 30  
command, 29  
  
confstrptr0, 30  
confstrptrvalid, 23  
control, 30  
count, 60  
custom, 40  
custom0, 40  
custom15, 40  
  
data, 39, 40, 52, 55, 67  
data0, 31  
dataaccess, 27  
dataaddr, 27  
datacount, 29  
datasize, 27  
dcsr, 44  
dmactive, 26  
dmcontrol, 23  
dmcs2, 32  
dmi, 67  
dmihardreset, 66  
dmireset, 66  
dmistat, 67  
dmode, 52  
dmstatus, 21  
dpc, 46  
dscratch0, 47  
dscratch1, 47  
dtmcs, 66  
  
ebreakm, 44  
ebreaks, 44  
ebreaku, 44  
etrigger, 62  
execute, 60  
exttrigger, 32  
  
field, 2  
  
haltgroup, 32  
haltreq, 24  
haltsum0, 33

haltsum1, 33  
 haltsum2, 34  
 haltsum3, 34  
 hartinfo, 26  
 hartreset, 24  
 hartsel, 23  
 hartselhi, 25  
 hartsello, 25  
 hasel, 25  
 hasresethaltreq, 23  
 hawindow, 27  
 hawindowssel, 27  
 hgselect, 33  
 hgwrite, 33  
 hit, 56, 60–62  
  
 icount, 60  
 IDCODE, 65  
 idle, 66  
 impebreak, 22  
 info, 53  
 itrigger, 61  
  
 load, 60  
  
 m, 59–62  
 ManuId, 66  
 maskmax, 56  
 match, 59  
 mcontext, 54  
 mcontrol, 55  
 mprven, 45  
 mpte, 54  
 mselect, 63  
 mte, 54  
 mvalue, 63  
  
 ndmreset, 25  
 nextdm, 31  
 nmi, 62  
 nmip, 45  
 nscratch, 26  
  
 op, 68  
  
 PartNumber, 66  
 postexec, 14  
 priv, 47  
 progbuf0, 31  
 progbufsize, 28  
 prv, 46, 48  
  
 Quick Access, 14  
  
 regno, 14  
 resethaltreq, 23  
 resumereq, 24  
  
 s, 60–62  
 sbaccess, 35  
 sbaccess128, 36  
 sbaccess16, 36  
 sbaccess32, 36  
 sbaccess64, 36  
 sbaccess8, 36  
 sbaddress0, 36  
 sbaddress1, 37  
 sbaddress2, 37  
 sbaddress3, 38  
 sbasize, 36  
 sbautoincrement, 35  
 sbbusy, 35  
 sbbusyerror, 35  
 sbcs, 34  
 sbdata0, 38  
 sbdata1, 39  
 sbdata2, 39  
 sbdata3, 40  
 sberror, 36  
 sbreadonaddr, 35  
 sbreadondata, 35  
 sbversion, 35  
 scontext, 54  
 select, 56  
 setresethaltreq, 25  
 shortname, 2  
 sizehi, 56  
 sizelo, 58  
 sselect, 63  
 step, 46  
 stepie, 44  
 stopcount, 45  
 stoptime, 45  
 store, 60  
 svalue, 63  
  
 target-specific, 16  
 tcontrol, 53

tdata1, [52](#)  
tdata2, [52](#)  
tdata3, [53](#)  
textra32, [62](#)  
textra64, [63](#)  
timing, [57](#)  
tinfo, [53](#)  
transfer, [14](#)  
tselect, [51](#)  
type, [52](#)

u, [60–62](#)

Version, [66](#)  
version, [23](#), [67](#)

write, [14](#), [16](#)

xdebugver, [44](#)



# Appendix C

## Change Log

| Revision | Date       | Author(s)        | Description   |
|----------|------------|------------------|---|
| 034d0d6  | 2019-04-30 | Tim Newsome      | Clarify that debuggers should honor maskmax. (#440)                                 |
| 4369eb8  | 2019-04-30 | pdonahue-ventana | Finesse ligatures to work with Adobe Acrobat Reader search and cut-and-paste (#442) |
| d125b9b  | 2019-04-30 | pdonahue-ventana | sberror and sdbusyerror don't both have to be non-zero to prevent (#447)            |
| 859e167  | 2019-04-30 | Tim Newsome      | Tweak address matches. (#449)   |
| 96b2b28  | 2019-04-25 | Tim Newsome      | Clarify not supported cmderr. (#446)  |
| 658417f  | 2019-04-16 | Tim Newsome      | When extending IR, BYPASS still is all ones. (#437)                                 |
| 2e24bab  | 2019-04-16 | Tim Newsome      | JTAG does not suggest any specific IDCODE encoding (#439)                           |
| c50efcb  | 2019-04-09 | Tim Newsome      | Rebuild PDF.  |
| 281e4ad  | 2019-03-21 | Tim Newsome      | Don't run text off a page when longtable is used. (#434)                            |
| 76874e9  | 2019-03-20 | Tim Newsome      | Explain how to detect the version. (#433)   |
| a543b76  | 2019-03-12 | Tim Newsome      | Rebuild PDF.  |
| a686747  | 2019-02-21 | Tim Newsome      | All trigger registers are optional (#431)   |
| d6e4cd8  | 2019-02-19 | Josh Scheid      | Fix typo. (#426)  |
| e773936  | 2019-02-19 | Tim Newsome      | Try to get travis to build the release branch. (#430)                               |
| 3621456  | 2019-02-19 | Tim Newsome      | Abstract memory accesses use the low bits of arg0. (#429)                           |
| 94a5f9c  | 2019-02-12 | Tim Newsome      | Clarify that harts halt out of reset if haltreq=1 (#419)                            |
| 518e732  | 2019-02-12 | Tim Newsome      | Rebuild PDF.  |
| 62f36e1  | 2019-02-11 | Tim Newsome      | Errata go in 0.13.x, this is 0.14. (#424)   |
| 66c3117  | 2019-01-31 | Tim Newsome      | Address triggers may fire on any accessed address. (#421)                           |
| 6102412  | 2019-01-31 | Tim Newsome      | \Faamsize does not affect Argument Width. (#420)                                    |
| 1ea1a9b  | 2019-01-09 | Tim Newsome      | Add nmi bit to etrigger. (#408)   |
| d1c7a3f  | 2019-01-09 | Tim Newsome      | Reserve trigger types for non-standard use. (#417)                                  |
| 83b12fb  | 2019-01-08 | Tim Newsome      | Rebuild PDF.  |

|         |            |             |   |
|---------|------------|-------------|---|
| b4b3b5c | 2019-01-07 | Tim Newsome | \Fversion may be invalid when \Factive=0 (#414)             |
| 800450f | 2019-01-01 | Tim Newsome | mte only applies when action=0 (#411)                       |
| 67c7fe2 | 2018-12-13 | Tim Newsome | Add pre-built PDF of the 0.13 release.                      |
| 5e7cb72 | 2018-12-12 | Tim Newsome | Stopcount only applies to hart-local counters. (#405)       |
| e5902fc | 2018-12-12 | Tim Newsome | Reserve some DMI space for non-standard use. (#406)         |
| 3c0dc6a | 2018-12-11 | Tim Newsome | Rebuild PDFs.   |
| aeec8f3 | 2018-12-04 | Tim Newsome | Add halt groups and external triggers. (#404)               |
| 814406d | 2018-11-13 | Tim Newsome | Clarify what the 4 states are. (#403)                       |
| cb64db0 | 2018-11-06 | Tim Newsome | Rebuild PDFs.   |
| 70da60c | 2018-11-05 | Tim Newsome | sselect applies to svalue. (#402)                           |
| 66fe38e | 2018-11-05 | Tim Newsome | Fix trigger example value. (#401)                           |
| 688ccaf | 2018-11-05 | Tim Newsome | Resume ack is set after resume. (#400)                      |
| 553dda7 | 2018-11-05 | Tim Newsome | Fix sbdata0 read order of operations. (#392)                |
| b864f54 | 2018-10-31 | Tim Newsome | Add Compatibility section to the introduction. (#399)       |
| 0b205b1 | 2018-10-31 | Tim Newsome | Create errata document. (#398)                              |
| 5390063 | 2018-10-26 | Tim Newsome | Bump version to 0.13.1. (#391)                              |
| e46c2db | 2018-10-08 | bdwyatt     | Fix link to PDF (#387)                                      |
| ed66f39 | 2018-10-02 | Tim Newsome | Rebuild PDF.  |
| f2873e7 | 2018-10-02 | Tim Newsome | Run/Halt figure applies only to single-hart systems. (#385) |
| a79945f | 2018-10-02 | Tim Newsome | Add ASID and context compare for triggers (#363)            |
| 9bb7da6 | 2018-10-02 | Tim Newsome | Clean up language of #383. (#384)                           |
| fce4da5 | 2018-10-02 | Tim Newsome | Make haltreq and resumereq proper write-only. (#383)        |
| e5da11e | 2018-10-02 | Tim Newsome | Minimal implementations can't access all registers (#381)   |
| e1be8f4 | 2018-10-02 | Tim Newsome | Format quotes correctly. (#382)                             |
| e9103ba | 2018-10-02 | Tim Newsome | Change from AVR debug connector to MIPI-10,20. (#375)       |
| 8841a7a | 2018-10-02 | Tim Newsome | Abstract reg access is independent of run/halt. (#380)      |
| 71c54bb | 2018-10-02 | Tim Newsome | Explicitly state what's required for compliance. (#379)     |
| 4edb285 | 2018-10-01 | Tim Newsome | Rebuild PDF.  |
| b0420b3 | 2018-10-01 | Tim Newsome | Final cleanups! Mostly table formatting. (#377)             |
| d43f5a4 | 2018-10-01 | Tim Newsome | Clarify W1. (#372)  |
| 72618f3 | 2018-10-01 | Tim Newsome | Leave space for trace, but don't specify anything. (#376)   |
| b7db4ce | 2018-10-01 | Tim Newsome | Add dcsr.cause for being halted out of reset. (#370)        |
| 42ab2a1 | 2018-09-28 | Tim Newsome | Clean up language, formatting, consistency. (#371)          |
| 7801874 | 2018-09-28 | Tim Newsome | Little language and formatting cleanups. (#366)             |
| 38ae12f | 2018-09-27 | Tim Newsome | Reset dmi.op to 0 instead of 2. (#369)                      |
| b50dc0d | 2018-09-27 | Tim Newsome | Formatting, language, consistency. (#373)                   |
| 425e9b1 | 2018-09-27 | Tim Newsome | Distinguish draft and release builds. (#364)                |
| c7b4e1c | 2018-09-26 | Tim Newsome | Stepping over wfi does not enter wait state. (#368)         |

|         |            |                |   |
|---------|------------|----------------|---|
| 4725879 | 2018-09-25 | Tim Newsome    | Language, formatting, and abstract cmd arguments. (#367)  |
| 62bf89d | 2018-09-25 | Tim Newsome    | Rebuild PDF.  |
| 10dfa65 | 2018-09-24 | Tim Newsome    | Allow global reset to reset the DM. (#350)  |
| 84ec8a5 | 2018-09-18 | Tim Newsome    | Harts can be in exactly 1 of 4 states. (#354)   |
| 308eaf6 | 2018-09-17 | Tim Newsome    | Mostly match "official" style for credits. (#362)   |
| b6187ff | 2018-09-17 | Tim Newsome    | Specify ackhavereset as W1. (#361)  |
| 41d9f06 | 2018-09-14 | Tim Newsome    | Abstract commands might work on a hung hart. (#360)   |
| fa561bd | 2018-09-14 | Tim Newsome    | Can't change harts during operations, and the current hart becoming unavailable may terminate the abstract command with error. (#322) |
| 900cdbf | 2018-09-11 | Tim Newsome    | Rebuild PDF.  |
| 514ef6f | 2018-09-07 | Tim Newsome    | Clarify lack of notification for other reset harts (#349)   |
| e0ff31e | 2018-09-07 | Tim Newsome    | Clarify postexec when there is no Program Buffer (#352)   |
| 3dacc00 | 2018-09-07 | Florian Zaruba | Move regno table to the actual access reg command (#345)  |
| 5d25cd5 | 2018-09-06 | Tim Newsome    | don't set most bits of DMCONTROL during abstract commands (#324)  |
| 12655e0 | 2018-09-06 | Tim Newsome    | Document breakpoint exception + enter debug mode (#299)   |
| 6894f4b | 2018-09-05 | Tim Newsome    | Define DXLEN as the widest supported XLEN. (#298)   |
| 114a208 | 2018-09-04 | Tim Newsome    | Restrict how many bits may be set in dmcontrol. (#348)  |
| 4cd1563 | 2018-09-03 | Tim Newsome    | Don't change selected harts during hart reset. (#337)   |
| 1529c26 | 2018-09-03 | Tim Newsome    | On trigger chains, only the last action is taken. (#341)  |
| 18a3531 | 2018-08-31 | Tim Newsome    | Authdata is bidirectional. (#347)   |
| 7d14f95 | 2018-08-27 | Tommy Thorn    | m "LaTeX/english issues: eg. -i, e.g., etc" (#342)  |
| 0fb41b9 | 2018-08-27 | Tim Newsome    | Don't change step/stepie while running. (#340)  |
| ff09418 | 2018-08-21 | Tim Newsome    | Rebuild PDF.  |
| 6bd15ac | 2018-08-20 | Tim Newsome    | Be more clear about running signal. (#338)  |
| e967b3b | 2018-08-20 | Tim Newsome    | mprven may be tied high or low. (#339)  |
| 0f120c0 | 2018-08-20 | Tim Newsome    | Solution to native triggers in M mode only systems (#309)   |
| 13d5c08 | 2018-08-17 | Tim Newsome    | Thank John Hauser.  |
| b52d9fe | 2018-08-17 | Tim Newsome    | Allow control xfers in progbuf to act as illegal. (#331)  |
| 19058ef | 2018-08-17 | Tim Newsome    | Clarify that resumereq is not level-sensitive. (#321)   |
| 497352c | 2018-08-16 | Tim Newsome    | Side effects happen for abstract register accesses (#334)   |
| fd5cf62 | 2018-08-15 | Tim Newsome    | Triggers do not fire in Debug Mode. (#335)  |
| 762d308 | 2018-08-15 | Tim Newsome    | Add aarpostincrement to abstract register access. (#333)  |
| 45b7636 | 2018-08-14 | Tim Newsome    | Clearing hasel does not clear the ha mask reg. (#327)   |

|         |            |             |  |
|---------|------------|-------------|--|
| 2ca20aa | 2018-08-13 | Tim Newsome | clrresethaltreq trumps setresethaltreq (#332)                          |
| 57df3f3 | 2018-08-10 | Tim Newsome | \Rcommand is not readable. (#328)                                      |
| 81df032 | 2018-08-10 | Tim Newsome | Explain what we mean by Preset. (#323)                                 |
| b51c6db | 2018-08-10 | Tim Newsome | Clarify ebreak behavior when ebreak* are 0. (#311)                     |
| a14d868 | 2018-08-10 | Tim Newsome | Allow extra harts to be reset. (#330)                                  |
| 6d60ad9 | 2018-08-07 | Tim Newsome | Rebuild PDF  |
| f4bd15f | 2018-08-02 | Tim Newsome | Define cmderr for non-existent register access. (#325)                 |
| 2d7d3d0 | 2018-07-20 | Tim Newsome | Fix typo in data0 definition.  |
| c8a64d1 | 2018-07-19 | Tim Newsome | Rebuild PDF.   |
| 9d2944f | 2018-07-18 | Tim Newsome | Add size to mcontrol. (#310)   |
| 6bd1a4c | 2018-07-16 | Tim Newsome | Put the description of dmstatus first. (#303)                          |
| 25e81e5 | 2018-07-12 | Tim Newsome | Fix typo in trigger example. (#308)                                    |
| 8462c94 | 2018-07-09 | Tim Newsome | Rebuild pdf.   |
| 38fde94 | 2018-07-09 | Tim Newsome | datacount cannot be 0 (#286)   |
| 800ca8d | 2018-07-06 | Tim Newsome | Clarifications requested by Jeremy Bennett (#280)                      |
| b363afa | 2018-07-06 | Tim Newsome | Add missing .tex file to dependencies. (#302)                          |
| 93340e4 | 2018-07-06 | Tim Newsome | Clarify that trigger registers are WARL. (#306)                        |
| 95af58a | 2018-07-06 | Tim Newsome | Force the register-address in place. (#304)                            |
| d83039d | 2018-07-06 | Tim Newsome | \Fcause priority numbers: higher means higher (#307)                   |
| 921c6a3 | 2018-07-03 | Tim Newsome | Completing progbuf exec is I/O for fence insts. (#305)                 |
| 99e01fa | 2018-06-27 | Tim Newsome | Add target-specific bits to abstract access memory. (#295)             |
| 4a0152d | 2018-06-19 | Tim Newsome | Only write busy to \Fcmderr if \Fcmderr is 0. (#296)                   |
| b0dc615 | 2018-06-16 | Tim Newsome | Rebuild the PDF.   |
| 90873eb | 2018-06-16 | Tim Newsome | Fix typo in abstract access memory examples. (#297)                    |
| 5fe8e08 | 2018-06-16 | Tim Newsome | dret is a section, not a subsection of reset (#294)                    |
| abfd8a0 | 2018-06-14 | Tim Newsome | Revert "Only write busy to \Fcmderr if \Fcmderr is 0."                 |
| 7c66968 | 2018-06-14 | Tim Newsome | Only write busy to \Fcmderr if \Fcmderr is 0.                          |
| 0f28f27 | 2018-06-08 | Tim Newsome | Abstract memory (#283)   |
| 7c840dd | 2018-06-08 | Tim Newsome | Specify an Exception Trigger (#266)                                    |
| 9d0d8af | 2018-06-06 | Tim Newsome | Clarify what address space these registers are in (#281)               |
| a7f293d | 2018-06-03 | Tim Newsome | Add missing dependency to Makefile (#285)                              |
| 37893aa | 2018-05-30 | Tim Newsome | Make trigger types writable. (#279)                                    |
| 6730cc0 | 2018-05-29 | Tim Newsome | Explain priority assignment rationale. (#277)                          |
| b6d5d66 | 2018-05-25 | Tim Newsome | Prevent M mode triggers affecting D mode ones (#282)                   |
| 08ee84f | 2018-05-22 | Tim Newsome | Reading tselect doesn't guarantee a valid trigger. (#271)              |
| 6dfe375 | 2018-04-18 | Megan Wachs | Debug Module should be capitalized                                     |
| dac2120 | 2018-04-11 | Megan Wachs | resethaltreq: Proposal for forcing a hart into debug mode out of reset |

|         |            |             |   |
|---------|------------|-------------|---|
| 3b6442f | 2018-05-16 | Tim Newsome | tdata2 need only hold valid addresses if select=0 (#278)                  |
| 68501cb | 2018-04-26 | mwachs5     | mprven: Add a bit to enable MPRV to take effect in debug mode             |
| 9fcabe0 | 2018-05-03 | Megan Wachs | Appendix: correct and clarify what debugger vs DM does                    |
| 30773fd | 2018-05-03 | Tim Newsome | Debuggers must not write sbcs while sbbusy is set (#270)                  |
| 50d8cd8 | 2018-05-03 | Megan Wachs | Remove merge commits from the changelog                                   |
| 3b7a296 | 2018-05-02 | Tim Newsome | Fix typo.   |
| b26072b | 2018-05-02 | Tim Newsome | Explain that 1 in hart array mask means selected                          |
| 41f6026 | 2018-05-02 | Megan Wachs | Examples: Give an example of CSR access with Quick Access (#268)          |
| 675bb14 | 2018-05-01 | Tim Newsome | Replace XLEN with MXLEN. #257   |
| 848cca1 | 2018-04-30 | Megan Wachs | Overview Diagram: increase number of Progbuf words (#267)                 |
| a719ee6 | 2018-04-25 | Megan Wachs | fix misspelled name   |
| 097c701 | 2018-04-23 | Tim Newsome | Fix typo.   |
| 01dabd5 | 2018-04-23 | Tim Newsome | Incorporate review feedback.  |
| ca7a9d0 | 2018-04-18 | Tim Newsome | Add trigger examples for match types 1, 4, and 5                          |
| cd5a15c | 2018-04-16 | Tim Newsome | Give a few trigger examples.  |
| 4375927 | 2018-04-12 | Tim Newsome | Clarify that maskmax applies only to NAPOT trigger                        |
| acadfe9 | 2018-04-13 | Megan Wachs | NMI: debugging may not be possible if an NMI happens                      |
| 8fb190c | 2018-04-12 | Tim Newsome | Another attempt at SBA errors.  |
| 714c5d1 | 2018-04-11 | Megan Wachs | Core Debug: all interrupts are masked includes NMI                        |
| 56fbd9d | 2018-04-11 | Megan Wachs | DCSR: add nmip bit to indicate NMI is pending                             |
| ffe3c2  | 2018-04-10 | Tim Newsome | Clarify SBA unsupported access size error.                                |
| b4006ac | 2018-04-10 | Tim Newsome | Clarify high bits of sbdata in narrow reads.                              |
| 4ca83dd | 2018-03-28 | Tim Newsome | Clarify progbuf=1 some more   |
| 3b62243 | 2018-03-26 | Tim Newsome | Clarify debugger requirements when progbufsize=1                          |
| ffba4d0 | 2018-03-26 | Tim Newsome | Explain why progbufsize=1 is special                                      |
| 6b88905 | 2018-03-19 | Megan Wachs | haltsum1: correct its address to be BWC and not overlap with ABSTRACTAUTO |
| 2382e2e | 2018-03-06 | Megan Wachs | Correct some inaccuracies in the chisel generated files                   |
| 3e88e11 | 2018-03-06 | Megan Wachs | travis: add 'make chisel' target to regression                            |
| 32cbb9b | 2018-03-19 | Tim Newsome | Nonexistent/unavailable harts are not halted.                             |
| f8a7bb7 | 2018-03-19 | Tim Newsome | More clarification.   |
| e21ae4c | 2018-03-16 | Tim Newsome | Allow any bit in hart array mask to be tied to 0                          |
| efb7e45 | 2018-03-15 | Tim Newsome | Change dcsr.prv reset value to 3  |
| f19946b | 2018-03-15 | Tim Newsome | Clarify hart array mask register size.                                    |
| ddec145 | 2018-03-14 | Tim Newsome | Be more precise about core vs hart  |
| 4e5f4ad | 2018-03-14 | Tim Newsome | Review feedback.  |
| 8ac9273 | 2018-03-14 | Tim Newsome | Be more precise about processor vs hart                                   |
| 83c9774 | 2018-03-14 | Tim Newsome | Clarify abstract command errors.  |
| 4ebc177 | 2018-03-14 | Tim Newsome | hawindowssel can be smaller, depends on # of harts                        |
| 11e1b5c | 2018-03-14 | Tim Newsome | Split future ideas section into a notes doc                               |
| bafeeaa | 2018-03-13 | Tim Newsome | Rebuild PDF   |

|         |            |                   |  |
|---------|------------|-------------------|--|
| 6a85d53 | 2018-03-13 | Tim Newsome       | Incorporate review feedback.   |
| f213315 | 2018-03-09 | Tim Newsome       | Clarify user responsibilities when debugging lr/sc   |
| 3641305 | 2018-03-09 | Tim Newsome       | Remove implemented features from Future Ideas.   |
| 1135bf3 | 2018-03-06 | Tim Newsome       | Incorporate feedback.  |
| 8f35e7e | 2018-03-05 | Megan Wachs       | gt_1024: Clarify that some registers may not be present for small numbers of harts                     |
| 683ae37 | 2018-02-14 | Megan Wachs       | hartsum- $\hat{}$ haltsum  |
| ee51758 | 2018-02-14 | Megan Wachs       | Modification of $\hat{}$ 1024 hart proposal that maintains backwards compatibility                     |
| 370d222 | 2018-03-05 | Tim Newsome       | Rephrase description of hit bit.   |
| eee5e0c | 2018-03-05 | Tim Newsome       | Clarify multiple DMs/harts   |
| 4d5acef | 2018-02-28 | Tim Newsome       | Clarify what happens when \Fauthenticated is clear   |
| 6a0c9ec | 2018-02-27 | Tim Newsome       | Move hit bit per review feedback.  |
| 097bd8e | 2018-02-21 | Tim Newsome       | Fix link to pre-built pdf  |
| d21774b | 2018-02-21 | Omer Faruk IR-MAK | Python interpreter to be used should default to Python2  |
| a8c10cf | 2018-02-20 | Tim Newsome       | Incorporate review feedback.   |
| a0f947c | 2018-02-20 | Tim Newsome       | Make trigger hit bit optional.   |
| 77e4634 | 2018-02-08 | Tim Newsome       | Add hit bit to hardware triggers.  |
| 140390a | 2018-02-05 | Tim Newsome       | Better wording.  |
| e35b1ff | 2018-02-05 | Tim Newsome       | Move Reg Access Abbrev table after sample register   |
| e887433 | 2018-02-05 | Tim Newsome       | Use longtable instead of xtabular.   |
| 5c84437 | 2018-01-31 | Tim Newsome       | Abstract Command data usage depends on the command   |
| 3d508ea | 2018-01-25 | Tim Newsome       | HARTSELBITS- $\hat{}$ HARTSELLEN and other feedback  |
| eb653f7 | 2018-01-24 | Tim Newsome       | Be explicit about the size of \Fhartsel.   |
| 822bd81 | 2018-01-24 | Tim Newsome       | Revert incrementing version number.  |
| 4c755af | 2018-01-24 | Tim Newsome       | \Fsbbusyerror also inhibits new accesses.  |
| 457413d | 2018-01-24 | Tim Newsome       | Update how to enumerate all harts.   |
| 2180801 | 2018-01-18 | Tim Newsome       | Fix ambiguity in busy error reporting.   |
| 3140efa | 2018-01-09 | Tim Newsome       | Re-apply e698a5001aa4583d31dde484d78f4f10e4e3148f . No need to list out all the consecutive registers. |
| 390daa7 | 2018-01-18 | mwachs5           | sbadress: Only writes to address will actually cause an error. Reads while busy are permitted.         |
| 5c820f3 | 2018-01-18 | Megan Wachs       | Remove reference to "caches"   |
| 4533648 | 2018-01-18 | Megan Wachs       | correct access spelling  |
| d37c1ac | 2018-01-16 | Tim Newsome       | Fix table column overruns by going full manual   |
| e9100ea | 2018-01-16 | Tim Newsome       | Correct when sbbusy error is set for being busy.   |
| c029cc7 | 2018-01-16 | Tim Newsome       | Complete partial sentence.   |
| 494338a | 2018-01-15 | Tim Newsome       | Add clarifications about error handling.   |
| e14c34e | 2018-01-15 | Tim Newsome       | Incorporate review feedback.   |
| 68720e5 | 2018-01-15 | Tim Newsome       | Remove H bits from triggers.   |
| b8eb62a | 2018-01-15 | Tim Newsome       | Clarify when sbaccess is checked for validity  |
| 8b50d29 | 2018-01-12 | Tim Newsome       | Add \Fsbbusy, to avoid race clearing \Fsbberror  |
| 50b1b41 | 2018-01-12 | Tim Newsome       | Clarify: writes to \Rsbdata0 write the new data  |
| 7f26759 | 2018-01-12 | Tim Newsome       | Clarify exactly which bits are used for SB access.   |
| 47a019c | 2018-01-11 | Tim Newsome       | Fix typo.  |

|         |            |             |   |
|---------|------------|-------------|---|
| a49d6ad | 2018-01-11 | Tim Newsome | sbreadonaddr is R/W                                       |
| 42195c2 | 2018-01-11 | Tim Newsome | Fix cut-and-paste error.                                  |
| 6c95235 | 2018-01-11 | Tim Newsome | Add sbaddress3, for future proofing.                      |
| e3345ea | 2018-01-11 | Tim Newsome | Incorporate review feedback.                              |
| 6da48f8 | 2018-01-11 | Tim Newsome | Remove dmerr.   |
| e99c092 | 2018-01-10 | Tim Newsome | Add system bus version field.                             |
| a6aa531 | 2018-01-10 | Tim Newsome | Talk about all data and progbuf regs in first reg         |
| af272db | 2018-01-09 | Megan Wachs | Update dret font  |
| 3d579d8 | 2018-01-09 | Tim Newsome | Explicitly list data[1-10] and progbuf[1-15]              |
| c6481ae | 2018-01-09 | Tim Newsome | Revert "Explicitly list data[1-10] and progbuf[1-15]"     |
| e698a50 | 2018-01-09 | Tim Newsome | Explicitly list data[1-10] and progbuf[1-15]              |
| e547ed5 | 2018-01-09 | Tim Newsome | Clarify that we deal in physical addresses only.          |
| b377b89 | 2018-01-09 | Tim Newsome | Revert "Clarify that we deal in physical addresses only." |
| f7da066 | 2018-01-09 | Tim Newsome | Clarify that we deal in physical addresses only.          |
| 99a1599 | 2018-01-09 | Tim Newsome | Clarify that \Fdatasize contains at most 12.              |
| ae6e88a | 2018-01-09 | mwachs5     | dret: Legal only in Debug Mode                            |
| 18f392d | 2017-11-24 | Tim Newsome | Get rid of sbsingleread in favor of sbreadonaddr          |
| 5754a3b | 2018-01-05 | Megan Wachs | Use a different word than "clobbered"                     |
| aca7e0b | 2018-01-03 | Megan Wachs | Add missing "to"s to abstractauto description             |
| d59ddf3 | 2018-01-03 | Megan Wachs | Correct plurality of halted harts in haltsum              |
| 57c53ed | 2017-12-22 | Tim Newsome | Put parens around all macros that need it.                |
| 7ded846 | 2017-12-18 | Tim Newsome | Refer to existing hart instead of "valid"                 |
| 68b8ac8 | 2017-12-15 | Tim Newsome | Make \Fhaltsel WARL.                                      |
| 6a72f45 | 2017-12-18 | Tim Newsome | Mark this as a draft, which it is.                        |
| dd8d871 | 2017-12-18 | Tim Newsome | Properly deal with \ chars in the changelog.              |
| 42f920c | 2017-12-18 | Tim Newsome | Deal with \ chars in the changelog.                       |
| b13891c | 2017-12-15 | Tim Newsome | Revert "Make \Fhaltsel WARL."                             |
| 26d76a0 | 2017-12-15 | Tim Newsome | Make \Fhaltsel WARL.                                      |
| afda8d7 | 2017-11-28 | mwachs5     | update PDF  |
| 134d310 | 2017-11-28 | Megan Wachs | Correct compressed version of ebreak                      |
| caa1258 | 2017-11-27 | Megan Wachs | badaddr -i tval (Priv Spec 1.9 -i 1.9.1)                  |
| 32b0f08 | 2017-11-22 | Tim Newsome | Incorporate feedback.                                     |
| 2f7aa54 | 2017-11-22 | Tim Newsome | Simplify, and explain trigger behavior.                   |
| 3e5887f | 2017-11-21 | Tim Newsome | Clarify some single step corner cases.                    |
| f4b9ae2 | 2017-11-21 | Tim Newsome | Make ackhavereset write-only. (#178)                      |
| efe3dc8 | 2017-11-21 | Tim Newsome | Make hartreset R/W (#177)                                 |
| ce1b359 | 2017-11-17 | Megan Wachs | Reset clarifications (#172)                               |
| 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.                               |
| 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)                 |
| 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   |
| e00625f | 2017-10-18 | Tim Newsome | Include pdf.   |
| c23e729 | 2017-10-18 | Tim Newsome | Clarify more.  |
| 83f9faf | 2017-10-11 | Tim Newsome | Clarify what <code>\Fimpebreak</code> does.  |
| 78082b5 | 2017-10-11 | Tim Newsome | Mention <code>\Fimpebreak</code> in Program Buffer description.                          |
| 0378324 | 2017-10-11 | mwachs5     | Add legend and update some transitions on the Abstract Command State Machine diagram     |
| fa2b600 | 2017-10-11 | Megan Wachs | add missing period   |
| 0610630 | 2017-10-11 | Megan Wachs | Just do simple <code>hmode -i dmode</code> replacement                                   |
| 16e11f3 | 2017-10-11 | Tim Newsome | Remove <code>hmode</code> reference, to fix build.                                       |
| 84b9a6a | 2017-10-11 | Tim Newsome | Add <code>\Fimpebreak</code> , to support of implicit <code>ebreak</code> .              |
| 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'   |
| ea2877d | 2017-10-10 | Tim Newsome | Move how-to-debug into the relevant section.   |
| 486ecc6 | 2017-10-05 | Tim Newsome | Refuse unsupported bus accesses.   |
| 6ca221d | 2017-10-05 | Tim Newsome | <code>haltreq</code> , <code>resumereq</code> , <code>hartreset</code> are per-hart bits |
| d4118ab | 2017-09-30 | Tim Newsome | <code>ndmreset</code> can't reset logic required to access DM.                           |
| c6bd8d1 | 2017-09-29 | Tim Newsome | and <code>-i</code> or   |
| 58c2441 | 2017-09-29 | Tim Newsome | Mention <code>\Fstepie</code> in Single Step   |
| 94c5f78 | 2017-09-29 | Tim Newsome | Clarify <code>ndmreset</code> .  |
| 12810b4 | 2017-09-29 | Tim Newsome | Clarify that <code>sbaddress</code> is physical.   |
| 5862fdf | 2017-09-29 | Tim Newsome | Unify M mode and <code>mprv</code> comment.  |
| aea1bd5 | 2017-09-29 | Tim Newsome | Define behavior when <code>haltreq</code> and <code>resumereq</code> are set             |
| 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 <code>\Fdmerr</code> .   |
| 25685eb | 2017-09-28 | Tim Newsome | Explain that bus master or <code>progbuf</code> 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 | <code>data0</code> should be <code>sbdata0</code> in this table.                         |
| 5858cfe | 2017-09-27 | Tim Newsome | Clarify why <code>\Rpriv</code> exists.  |
| bc3c2aa | 2017-09-27 | Tim Newsome | Mention where <code>priv</code> 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 <code>instret</code> not incrementing on <code>ebreak</code> .                   |
| c163d22 | 2017-09-20 | Tim Newsome | Remove <code>ebreakh</code> .  |
| 9971075 | 2017-09-20 | Tim Newsome | Clarify we're talking about privilege  |
| 3fbe495 | 2017-09-20 | Tim Newsome | Clarify that we're talking about *implementation*  |
| 3684854 | 2017-09-20 | Tim Newsome | Use steps environment in <code>sbdata0</code> .  |
| d4eda18 | 2017-09-20 | Tim Newsome | Explain that only <code>sbdata0</code> has side effects.                                 |
| ae781c6 | 2017-09-20 | Tim Newsome | Don't refer to internal system bus registers.  |
| 875922e | 2017-09-20 | Tim Newsome | Explain <code>sbdata0</code> being stale a bit more.                                     |
| cd44fd5 | 2017-09-20 | Tim Newsome | Clarify <code>autoread</code>  |
| 194484b | 2017-09-20 | Tim Newsome | Clarify <code>hawindow</code> .  |
| 02f1aac | 2017-09-20 | Tim Newsome | Clarify that <code>\Fdataaddr</code> is relative to <code>\Rzero</code> .                |
| 0e9b6ae | 2017-09-20 | Tim Newsome | Clarify nonexistent vs unavailable.  |
| b55ff41 | 2017-09-20 | Tim Newsome | Fix <code>devtreevalid</code> .  |



|         |            |             |   |
|---------|------------|-------------|---|
| 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 <code>\Ftransfer</code> might be used.                           |
| 6b2ee61 | 2017-09-20 | Tim Newsome | Explain where <code>\Fsize</code> encoding came from.                         |
| c9f3b73 | 2017-09-14 | Tim Newsome | Fix typo.   |
| 4b25400 | 2017-09-13 | Tim Newsome | Mention <code>dpc</code> 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 <code>-i</code> operation   |
| 994afdc | 2017-09-13 | Tim Newsome | Account for multiple selected harts.  |
| aa4a297 | 2017-09-13 | Tim Newsome | Halt Control <code>-i</code> Run Control                                      |
| e97c821 | 2017-09-13 | Tim Newsome | continuous <code>-i</code> contiguous   |
| 97f73ff | 2017-09-13 | Tim Newsome | Clarify <code>ndmreset</code> behavior.                                       |
| 6078220 | 2017-09-13 | Tim Newsome | Explain <code>ndmreset</code>   |
| 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.                                  |
| 8967b0a | 2017-08-16 | Megan Wachs | Compressed instructions are <code>c.foo</code> , not <code>foo.c</code>       |
| b5698a9 | 2017-08-16 | Megan Wachs | clarify <code>progbufsize</code> description                                  |
| d221bab | 2017-08-16 | Megan Wachs | Remove <code>progbufsize</code> enums from register description               |
| 0498102 | 2017-08-16 | Megan Wachs | appendix: Use standard assembly format for sw                                 |
| 4456d99 | 2017-08-09 | Tim Newsome | Rename <code>progsz</code> to <code>progbufsize</code> .                      |
| 55d5b66 | 2017-08-09 | Tim Newsome | Clarify that trigger comparisons are unsigned.                                |
| 21e35ef | 2017-08-09 | Tim Newsome | Configuration String <code>-i</code> Device Tree                              |
| 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   |
| 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.        |
| 5ba18f9 | 2017-07-27 | Tim Newsome | Fix more typos.   |
| 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 <code>dpc</code> contents.  |
| 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   |
| c5ab04c | 2017-07-10 | Megan Wachs | Update <code>abstract_commands.xml</code>                                     |
| 6e8cdf1 | 2017-07-10 | Megan Wachs | Update <code>abstract_commands.xml</code>                                     |

|         |            |             |   |
|---------|------------|-------------|---|
| cf6e3f2 | 2017-07-10 | Megan Wachs | clarify DCSR.cause  |
| 79ffb9  | 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  |
| 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              |
| 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   |
| 4688988 | 2017-06-29 | mwachs5     | DCSR: define a 'stepie' bit which may be hard-wired to 0.                             |
| 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   |
| 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.   |
| 89ffe50 | 2017-06-06 | mwachs5     | NDMRESET: Clarify what it may and may not do  |
| 1932da0 | 2017-06-06 | mwachs5     | DPC: Clarifications on its meaning  |
| 6470fdb | 2017-06-06 | mwachs5     | ABSTRACTCS: Correct inconsistency on the number of data words.                        |
| 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   |
| 1347371 | 2017-06-05 | Tim Newsome | Add intdisable to dcsr.   |
| 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.   |
| b8cc523 | 2017-06-02 | Tim Newsome | Add included tex files as dependencies. (#78)   |
| 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 | Remove virtual register from core_registers.xml<br>(Temporary Acct.)  |
| bfc64fb | 2017-05-10 | Megan Wachs | Add missing sw_registers.tex file<br>(Temporary Acct.)  |
| 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.   |
| 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.  |
| 1e5c068 | 2017-03-27 | Megan Wachs  | Add LICENSE  |
| 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   |
| 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.   |
| 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.  |
| c9db98c | 2017-02-22 | Tim Newsome   | Simplify description of op statuses.   |
| bda39cc | 2017-02-22 | mwachs5       | Add explicit type field to Abstract Command.   |
| f83a1ca | 2017-02-22 | mwachs5       | Finish up replacement of ibuf- $\bar{c}$ progbuf   |
| 9666e51 | 2017-02-22 | mwachs5       | IBUF- $\bar{c}$ 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.   |
| 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 - $\bar{c}$ Halt Mode.   |

|         |            |             |   |
|---------|------------|-------------|---|
| d29fc1f | 2017-02-22 | Tim Newsome | Debug Mode -i 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.   |
| 0802d5a | 2017-02-15 | mwachs5     | Use consistent 'Control and Status' naming for CS registers.  |
| 5acc7d  | 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.  |
| 4b62c40 | 2017-02-14 | mwachs5     | SystemBus: Clean up some formatting and error specification notes.  |
| 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.  |
| 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  |
| 23c2648 | 2017-02-11 | mwachs5     | jtag-dtm: ask for clarification on TAP sharing.   |
| 7020d23 | 2017-02-11 | mwachs5     | jtag-dtm: Clarifications, DBUS-iDMI   |
| 292d49c | 2017-02-11 | Megan Wachs | fix indentation   |
| 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.   |
| 170bff1 | 2017-02-10 | Megan Wachs | Allow size 4 for the program buffer   |
| 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 i32-bit registers.   |
| 7f0f09a | 2017-02-08 | Tim Newsome | Reset dbus status to "failure" to avoid confusion.  |
| 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   |

|         |            |             |  |
|---------|------------|-------------|--|
| 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.  |
| 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.                                 |
| 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  |