# Поддержка отладки в RISC-V
Версия 1.0.0-STABLE
2794e83f020233b6bb8dfee6f641377c8eaa01a1

Редакторы оригинала:
Ernie Edgar <ernie.edgar@sifive.com>, SiFive, Inc.
Tim Newsome <tim@sifive.com>, SiFive, Inc.
Перевод:
Ivan Kuzmenko <https://github.com/rndtrash>

Mon Oct 25 15:13:51 2021 +0300

Люди, внёсшие свой вклад во все версии данной спецификации, в алфавитном порядке (пожалуйста, свяжитесь с редакторами для внесения поправок): Bruce Ableidinger, Krste Asanović, Peter Ashenden, Allen Baum, Mark Beal, Alex Bradbury, Chuanhua Chang, Zhong-Ho Chen, Monte Dalrymple, Paul Donahue, Vyacheslav Dyachenko, Peter Egold, Marc Gauthier, Markus Goehrle, Robert Golla, John Hauser, Richard Herveille, Yung-ching Hsiao, Po-wei Huang, Scott Johnson, L. J. Madar, Grigorios Magklis, Jan Matyas, Kai Meinhard, Jean-Luc Nagel, Aram Nahidipour, Rishiyur Nikhil, Gajinder Panesar, Deepak Panwar, Antony Pavlov, Klaus Kruse Pedersen, Ken Pettit, Darius Rad, Joe Rahmeh, Gavin Stark, Ben Staveley, Wesley Terpstra, Megan Wachs, Jan-Willem van de Waerdt, Philipp Wagner, Stefan Wallentowitz, Ray Van De Walker, Andrew Waterman, Thomas Wicki, Andy Wright, и Bryan Wyatt.

# Предисловие

**Внимание! Данная черновая спецификация будет меняться до тех пор, пока её не примут в стандарт, так что реализации, основанные на этом черновике скорее всего не будет удовлетворять будущему стандарту.**

# Оглавление

# Список иллюстраций

# Список таблиц

# Глава 1

# Вступление

Как только проект прогрессирует из симуляции в реализацию на "железе", у пользователя существенно снижается уровень контроля и понимания текущего состояния системы. Чтобы помочь в создании и отладке низкоуровневого ПО и АО, важно иметь хорошую поддержку отладки, встроенную в само ядро. Когда на ядре работает надёжная ОС, ПО может взять на себя множество задач отладки. Однако в большинстве случаев важна поддержка в самом оборудовании.

Данный документ излагает стандартную архитектуру для поддержки отладки на платформах с ядром RISC-V. Данная архитектура допускает множество реализаций и компромиссов, что приветствуется в широком спектре реализаций RISC-V. В то же время эта спецификация определяет общие интерфейсы, позволяющие отладчикам и компонентам поддерживать множество аппаратных платформ на основе архитектуры набора команд RISC-V.

Разработчики систем могут добавить дополнительную аппаратную поддержку отладки, но эта спецификация определяет только стандартный интерфейс для базового функционала.

## 1.1 Терминология

**AMO**
: Atomic Memory Operation - атомарная операция над памятью.

**BYPASS**
: Инструкция JTAG, предназначенная для выбора регистра обхода в качестве регистра данных, также называемая BYPASS.

**компонент**
: Ядро RISC-V или любая другая часть аппаратной платформы. Как правило, все компоненты будут подключены к одной системной шине.

**CSR**
: Control and Status Register - регистр контроля/статуса.

**DM** Debug Module - модуль отладки (см. Раздел 3.8).

**DMI**
Debug Module Interface - интерфейс модуля отладки (см. Раздел 3.1).

**DR** JTAG Data Register - регистр данных JTAG.

**DTM**
Debug Transport Module - модуль передачи отладочной информации (см. Раздел 6).

**DXLEN**
Debug XLEN - XLEN Отладчика, являющийся самым широким XLEN, поддерживаемым hart, игнорируя текущее значение MXL в `misa`.

**GPR**
General Purpose Register - регистр общего назначения.

**аппаратная платформа**
Система, состоящая из одного или нескольких *компонентов*.

**hart**
HARdware Thread - аппаратный поток в ядре RISC-V.

**IDCODE**
32-bit Identification CODE - 32-битный идентифицирующий код, а также инструкция JTAG, возвращающая значение IDCODE.

**IR** JTAG Instruction Register - регистр инструкции JTAG.

**JTAG**
Имеется ввиду работа, проделанная IEEE Joint Test Action Group, описанная в IEEE 1149.1.

**NAPOT**
Naturally Aligned Powers-Of-Two - выровненный по степени двойки.

**NMI**
Non-Maskable Interrupt - немаскируемое прерывание.

**физический адрес**
Адрес, непосредственно используемый в системной шине.

**SBA**
System Bus Access - доступ к системной шине (см. Раздел 3.10).

**TAP**
Test Access Port - порт тестирования. Определение дано в IEEE 1149.1.

**TM** Trigger Module - модуль триггера (см. Раздел 5).

**виртуальный адрес**
Адрес с точки зрения hart. Если hart использует преобразование адресов, то виртуальный адрес может отличаться от физического. Если не выполнены никакие преобразования, то адреса будут одинаковыми.

`xepc`
CSR со счётчиком команд исключения (например `mepc`), применимый в режиме, в который вводит отладочная ловушка.

## 1.2 Контекст

Документ использует термины, описанные в:

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 1.12 (the Privileged Spec)

### 1.2.1 Версии

Версия 0.13 данного документа была утверждена комиссией фонда RISC-V. Версии 0.13.$x$ являются исправлениями ошибок в той утверждённой спецификации.

Версия 0.14 была черновиком, который никогда не был официально ратифицирован.

Версия 1.0.0 почти что полностью прямо и обратно совместима с версией 0.13.

#### 1.2.1.1 Исправления ошибок с 0.13 по 1.0

Изменения, исправляющие баги в спецификации:

1. Fix order of operations described in `sbdata0`. #392
2. Resume ack is set after resume, in Section 3.5. #400
3. sselect applies to svalue. #402
4. mte only applies when action=0. #411
5. aamsize does not affect Argument Width. #420
6. Clarify that harts halt out of reset if haltreq =1. #419

#### 1.2.1.2 Несовместимые изменения с 0.13 по 1.0

Изменения, нарушающие обратную совместимость. Отладчики или аппаратные реализации, созданные по спецификации версии 0.13, должны быть изменены для совместимости с версией 1.0:

1. Make haltsum0 optional if there is only one hart. #505
2. System bus autoincrement only happens if an access actually takes place. (`sbdata0`) #507
3. Bump version to 3. #512
4. Require debugger to poll dmactive after lowering it. #566
5. Add pending to `icount`. #574

#### 1.2.1.3 Небольшие изменения с 0.13 по 1.0

Изменения, незначительно изменяющие поведение. Технически обратно несовместимы, но скорее всего незаметны:

1. stopcount only applies to hart-local counters. #405
2. version may be invalid when dmactive =0. #414
3. Address triggers (mcontrol) may fire on any accessed address. #421
4. All trigger registers (Section 5.3) are optional. #431
5. When extending IR, bypass still is all ones. #437
6. ebreaks and ebreaku are WARL. #458
7. NMI are disabled by stepie. #465
8. R/W1C fields should be cleared by writing every bit high. #472
9. Specify trigger priorities in Table 5.2 relative to exceptions. #478
10. Time may pass before dmactive becomes high. #500
11. Clear MPRV when resuming into lower privilege mode. #503
12. Halt state may not be preserved across reset. #504
13. Hardware should clear trigger action when dmode is cleared and action is 1. #501
14. Change quick access exceptions to halt the target in Section 3.7.1.2. #585
15. Writing 0 to tdata1 forces a state where tdata2 and tdata3 are writable. #598

#### 1.2.1.4 Новый функционал с 0.13 по 1.0

Новый обратно совместимый функционал, не существовавший ранее:

1. Add halt groups and external triggers in Section 3.6. #404
2. Reserve some DMI space for non-standard use. See custom, and custom0 through custom15. #406
3. Reserve trigger type values for non-standard use. #417
4. Add nmi bit to etrigger. #408
5. Recommend matching on every accessed address. #449
6. Add resume groups in Section 3.6. #506
7. Add relaxedpriv. #536
8. Move scontext, renaming original to mscontext, and create hcontext. #535
9. Add mcontrol6, deprecating mcontrol. #538
10. Add hypervisor support: ebreakvs, ebreakvu, v, hcontext, mcontrol, mcontrol6, and priv. #549
11. Optionally make anyunavail and allunavail sticky, controlled by stickyunavail. #520
12. Add tmexttrigger to support trigger module external trigger inputs. #543
13. Describe mcontrol and mcontrol6 behavior with atomic instructions. #561
14. Trigger hit bits must be set on fire, may be set on match. #593
15. Add sbytemask and sbytemask to textra32 and textra64. #588
16. Allow debugger to request harts stay alive with keepalive bit in Section 3.14.2. #592
17. Add ndmresetpending to allow a debugger to determine when ndmreset is complete. #594
18. Add intctl to support triggers from an interrupt controller. #599

## 1.3 Об этом документе

### 1.3.1 Структура

Этот документ состоит из двух частей. Главная часть документа - это спецификация, данная в нескольких главах. Во второй части данного документа находится приложение. Информация в приложении предназначена для уточнения и предоставления примеров, но это не является частью самой спецификации.

### 1.3.2 ISA против не-ISA

Эта спецификация состоит как из частей, входящих в ISA, так и наоборот. Части ISA определяют состояние и поведение hart, а части не-ISA - состояние и поведение всего остального. Главы, в которых содержатся только части, (не)относящиеся к ISA, отмечены как таковые в заголовке. Главы без уточнения в заголовке относятся как к ISA, так и к не-ISA.

### 1.3.3 Формат определений регистров

Все определения регистров в этом документе следуют показанному ниже формату. Простой график показывает положение полей в регистре. Номер верхнего и нижнего бита отмечен соответственно в верхнем левом и верхнем правом углу каждого поля. Суммарное количество битов в поле написано под ним.

После графика следует таблица, обозначающая имя, описание, разрешённый доступ и значение сброса для каждого поля. Разрешённые доступы перечислены в Таблице 1.2. Значение сброса является либо константой, либо “Предустановленно.” Второе обозначение значит, что регистр имеет действительное значение, зависящее от реализации.

Имена регистров и их поля являются гиперссылками на их определение. Также они находятся в перечне на странице 107.

#### 1.3.3.1 Long Name (`shortname`, at 0x123)

| 31 … 8 | 7 … 0 |
|---|---|
| 0 | field |
| 24 | 8 |

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

Таблица 1.2: Аббревиатуры доступа к регистру

| | |
|---|---|
| R | Read-only - только для чтения. |
| R/W | Read/Write - чтение/запись. |
| R/W1C | Read/Write Ones to Clear - чтение/запись, единицы для очистки. Запись нуля в каждый бит не имеет какого либо эффекта. Запись единицы в каждый бит очищает поле. Результат остальных операций не определён. |
| WARZ | Write any, read zero - любая запись, чтение - ноль. Отладчик может записать любое значение. При чтении это поле возвращает 0. |
| W1 | Write-only - только для записи. Только запись единиц влияет на значение. При чтении возвращённое значение должно быть равным нулю. |
| WARL | Write any, read legal - любая запись, чтение допустимого [значения]. Отладчик может записать любое значение. Если значение не поддерживается, реализация преобразовывает значение в поддерживаемое. |

## 1.4 Предпосылки

Существует множество применений для отдельного аппаратного отладчика, как встроенного в ядро процессора, так и подключаемого извне. Эта спецификация адресована сценариям использования, перечисленным ниже. Имплементации могут не поддерживать каждую возможность, так что некоторые варианты использования могут не поддерживаться.

- Отладка низкоуровневого ПО при отсутствии ОС или других программ.
- Отладка проблем в самой ОС.
- Инициализация аппаратной платформы для тестирования, конфигурации, а также программных компонентов до того, как код будет исполнен на аппаратной платформе.
- Получение доступа к компонентам аппаратной платформы без работающего ЦП.

Также даже без аппаратного интерфейса отладки архитектурная поддержка в процессоре RISC-V поможет в отладке приложений и анализе производительности с помощью аппаратных триггеров и точек останова.

## 1.5 Возможности

Отладочный интерфейс, описанный в данной спецификации, поддерживает следующие функции:

1. Все регистры hart (включая CSR) могут быть прочитаны/перезаписаны.
2. Имеется доступ к памяти как с точки зрения hart, так и напрямую через системную шину.
3. Существует поддержка RV32, RV64 и будущего RV128.
4. Любой hart в аппаратной платформе может быть отдельно отлажен.
5. Отладчик способен найти почти что[1] всё, что ему нужно знать сам по себе, без конфигурации пользователем.
6. Каждый hart может быть отлажен начиная с самой первой выполненной инструкции.
7. Hart в RISC-V можно приостановить, когда исполнена инструкция программной точки останова.
8. Железо может исполнить инструкции пошагово.
9. Функционал отладчика независим от используемого интерфейса передачи отладочных данных.
10. Отладчику не нужно что-либо знать о микроархитектуре отлаживаемого hart.
11. Работа произвольного набора hart может быть одновременно приостановлена или возобновлена. (Опционально)
12. Можно выполнить произвольные инструкции на приостановленном hart. Это значит, что не требуется никакого нового функционала отладчика, когда ядро имеет дополнительные или особые инструкции или состояния, если существует программы, способные перемещать это состояние в GPR. (Опционально)
13. Возможно получить доступ к регистрам без приостановки исполнения кода. (Опционально)
14. Работающий hart может быть адресован для исполнения короткого набора инструкций с небольшими издержками. (Опционально)
15. Управляющий системной шиной (`нужна помощь с переводом “system bus master”`) допускает доступ к памяти без использования каких либо hart. (Опционально)
16. Hart в RISC-V может быть приостановлен, когда выполняется условие триггера на счётчик инструкций, запись/чтение адреса/данных, или особый опкод инструкции. (Опционально)
17. Hart-ы могут быть сгруппированы, и все hart-ы, принадлежащие одной группе приостановятся, когда хотя бы один из них был приостановлен. Эти группы могут вызывать или реагировать на внешние триггеры. (Опционально)

[1] Примечательным исключением является информация о схеме распределения памяти и периферии.

Этот документ не предлагает стратегию или реализацию техник для тестирования, отладки или отлавливания ошибок в АО. Сканирование, встроенная самодиагностика (built-in self test, BIST) и прочее не рассматриваются в этой спецификации, но она не ограничивает их использование в системах на основе RISC-V.

Существует возможность отладки кода, использующего программные потоки, но для них нет особой поддержки отладчиком.

# Глава 2

# Обзор системы

Рисунок 2.1 показывает главные компоненты поддержки отладки. Блоки, обозначенные пунктиром, являются опциональными.

Пользователь взаимодействует с узлом отладки (например ноутбуком), на котором запущен отладчик (например gdb). Отладчик работает с отладочным переводчиком (например OpenOCD, к которому может быть подключён аппаратный драйвер) для связи с устройством передачи отладочной информации (например Olimex USB-JTAG adapter). Данное устройство подключает узел отладки к DTM аппаратной платформы. DTM предоставляет доступ к одному или нескольким DM, используя DMI.

Каждый hart в аппаратной платформе управляется только одним DM. Hart-ы могут быть гетерогенными. Лимита по ассоциации hart-ов с DM не существует, но обычно все hart-ы в пределах одного ядра управляются одним и тем же DM. В большинстве аппаратных платформ будет только один DM, управляющий всеми hart-ами в аппаратной платформе.

DM-ы предоставляют контроль запуска их hart-ов в аппаратной платформе. Абстрактные команды дают доступ GPR-ам. Дополнительные регистры доступны через абстрактные команды или запись программ в опциональный программный буфер.

Программный буфер позволяет отладчику запускать произвольные инструкции на hart-е. Этот механизм также может использовать память. Необязательный блок доступа к системной шине допускает доступ к памяти без использования для этих целей hart-а RISC-V.

Каждый hart RISC-V может реализовывать TM. Когда выполнены условия триггера, hart-ы остановятся и проинформируют DM о своей остановке.

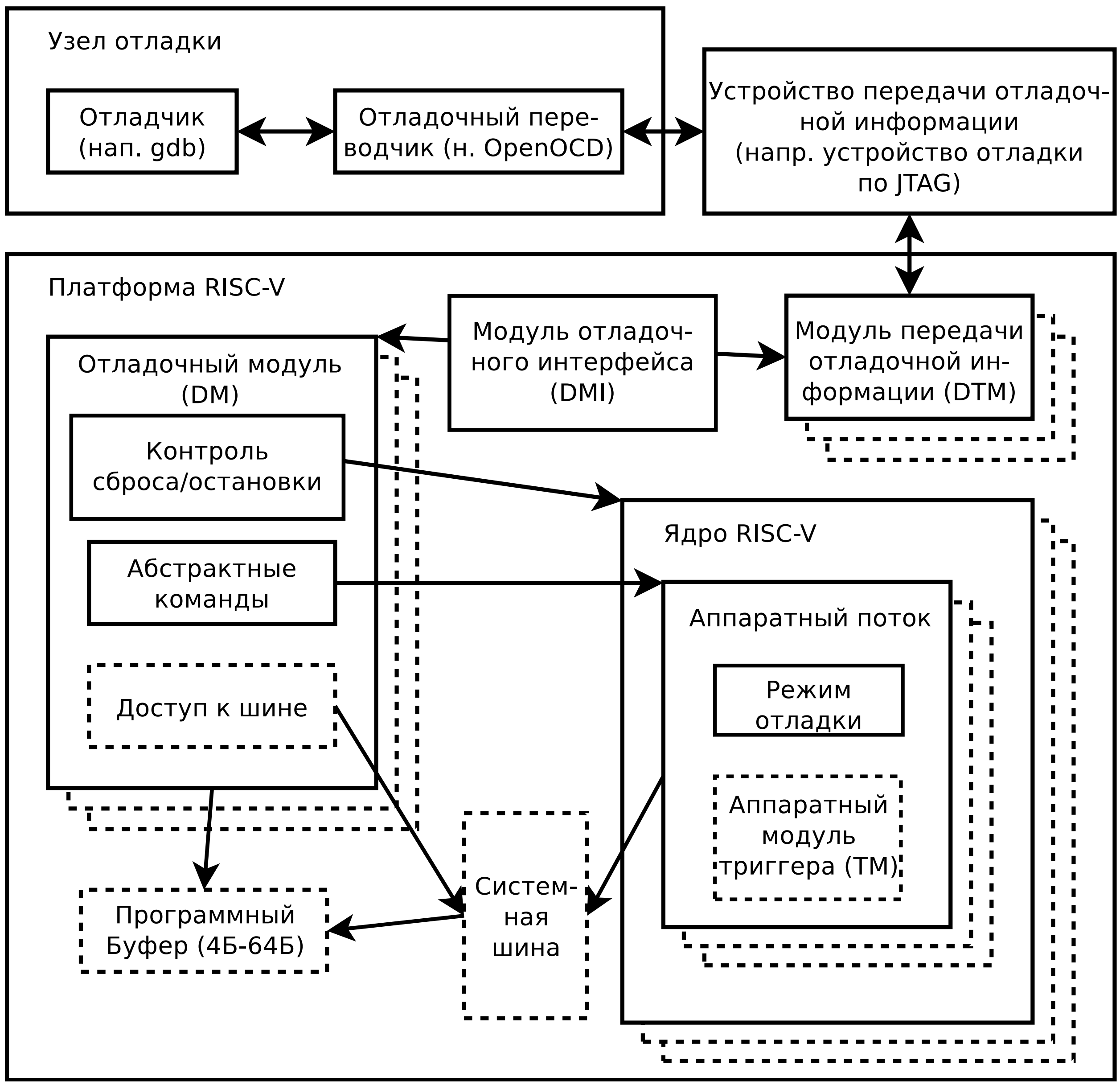


Рис. 2.1: Обзор системы отладки в RISC-V

# Глава 3

# Модуль отладки, не-ISA

Модуль отладки реализует интерфейс перевода между абстрактными отладочными операциями и их конкретными реализациями. Он может поддерживать следующие операции:

1. Дать отладчику необходимую информацию о реализации.
2. Позволить любому индивидуальному hart-у приостановить или возобновить работу.
3. Предоставить список приостановленных hart-ов.
4. Предоставить абстрактный доступ к чтению и записи в GPR-ы приостановленного hart-а.
5. Предоставить доступ к сигналу сброса, позволяющий начать отладку с самой первой инструкции после сброса.
6. Предоставить механизм для разрешения отладки hart-ов сразу же после сброса (независимо от причины сброса). (Опционально)
7. Предоставить абстрактный доступ к не-GPR регистрам `(FIXME: тавтология?)` hart-а. (Опционально)
8. Предоставить программный буфер для принуждения hart-а к выполнению произвольных инструкций. (Опционально)
9. Позволить приостановить, возобновить, и/или сбросить несколько hart-ов одновременно. (Опционально)
10. Разрешить доступ к памяти с точки зрения hart-а. (Опционально)
11. Разрешить прямой доступ к системной шине. (Опционально)
12. Группирование hart-ов. Когда хотя бы один из hart-ов приостанавливается, все остальные также останавливаются. (Опционально)
13. Ответить на внешние триггеры, останавливая каждый hart в настроенной группе. (Опционально)
14. Отправить сигнал внешнему триггеру, когда hart в группе приостанавливается. (Опционально)

Чтобы соответствовать этой спецификации, реализация должна:

1. Реализовать все операции из списка выше без пометки “(Опционально)”.
2. Реализовать хотя бы один из командных механизмов программного буфера, доступа к системной шине или памяти абстрактного доступа.
3. Реализовать хотя бы одно из перечисленного:

(a) Программный буфер.
(b) Абстрактный доступ ко всем регистрам, видимым ПО, работающим на hart-е, в том числе все регистры, имеющиеся на hart-е и перечисленным в Таблице 3.3.
(c) Абстрактный доступ ко всем GPR-ам, `dcsr` и `dpc`, а также объявите, что ваша реализация следует спецификации "Минимальная спецификация отладчика RISC-V 1.0.0-STABLE" вместо "Спецификация отладчика RISC-V 1.0.0-STABLE".

Один модуль отладки может отлаживать до $2^{20}$ hart-ов.

## 3.1 Интерфейс модуля отладки

Модули отладки являются slave по отношению к шине под названием интерфейс модуля отладки (DMI). Master этой шины – DTM. DMI может быть тривиальной шиной с одним master и одним slave (см. A.3) или использовать более полноценную шину, например TileLink или AMBA Advanced Peripheral Bus. Подробности оставим проектировщикам системы.

DMI использует от 7 до 32 битов адреса. Он поддерживает операции чтения и записи. Начало адресного пространства используется для первого (и зачастую единственного) модуля отладки. Дополнительное пространство можно использовать для произвольных устройств отладки, других ядер, дополнительных модулей отладки и т.д. Если этот DMI имеет дополнительные модули отладки, то базовый адрес следующего модуля в адресном пространстве DMI записан в `nextdm`.

Модуль отладки управляется через регистровый доступ к адресному пространству его DMI.

## 3.2 Reset Control

There are two methods that allow a debugger to reset harts. ndmreset resets all the harts in the hardware platform, as well as all other parts of the hardware platform except for the Debug Modules, Debug Transport Modules, and Debug Module Interface. Exactly what is affected by this reset is implementation dependent, but it must be possible to debug programs from the first instruction executed. hartreset resets all the currently selected harts. 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.

To perform either of these resets, the debugger first asserts the bit, and then clears it. The actual reset may start as soon as the bit is asserted, but may start an arbitrarily long time after the bit is deasserted. The reset itself may also take an arbitrarily long time. While the reset is on-going, harts are either in the running state, indicating it's possible to perform some abstract commands during this time, or in the unavailable state, indicating it's not possible to perform any abstract commands during this time. Once a hart's reset is complete, `havereset` becomes set. When a hart comes out of reset and haltreq or resethaltreq are set, the hart will immediately enter Debug Mode (halted state). Otherwise, if the hart was initially running it will execute normally (running state) and if the hart was initially halted it should now be running but may be halted.

*There is no general, reliable way for the debugger to know when reset has actually begun.*

The Debug Module's own state and registers should only be reset at power-up and while dmactive in dmcontrol is 0. If there is another mechanism to reset the DM, this mechanism must also reset all the harts accessible to the DM.

Due to clock and power domain crossing issues, it might not be possible to perform arbitrary DMI accesses across hardware platform 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.

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 might or might not be cleared when dmactive is low.

## 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 hardware platform'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 hawindowsel 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.

Execution of Abstract Commands ignores this mechanism and only applies to the hart selected by hartsel.

## 3.4 Hart DM States

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

Harts are nonexistent if they will never be part of this hardware platform, no matter how long a user waits. E.g. in a simple single-hart hardware platform only one hart exists, and all others are nonexistent. Debuggers may assume that a hardware platform 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 hardware platform. That means harts might become available or unavailable at any time, although these events should be rare in hardware platforms built to be easily debugged. There are no guarantees about the state of the hart when it becomes available.

Hardware platforms 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.

If the DM is reset while a hart is halted, it is UNSPECIFIED whether that hart resumes. Debuggers should use resumereq to explicitly resume harts before clearing dmactive and disconnecting.

## 3.6 Halt Groups, Resume Groups, and External Triggers

An optional feature allows a debugger to place harts into two kinds of groups: halt groups and resume groups. It is also possible to add external triggers to a halt and resume groups.

When any hart in a halt group halts, or an external trigger that's a member of the halt group fires:

1. All the harts in that group will quickly halt, even if they are currently in the process of resuming.
2. Any external triggers in that group are notified.

Adding a hart to a halt group does not automatically halt that hart, even if other harts in the group are already halted.

When any hart in a resume group resumes, or an external trigger that's a member of the resume group fires:

1. All the other harts in that group will quickly resume as soon as any currently executing

abstract commands have completed, except for the harts that are in the process of halting.
2. Any external triggers in that group are notified.

Adding a hart to a resume group does not automatically resume that hart, even if other harts in the group are currently running.

External triggers are abstract concepts that can signal the DM and/or receive signals from the DM. This configuration is done through `dmcs2`, where external triggers are referred to by a number. Commonly, external triggers are capable of sending a signal from the hardware platform into the DM, as well as receiving a signal from the DM to take their own action on. It is also allowable for an external trigger to be input-only or output-only. By convention external triggers 0–7 are bidirectional, triggers 8–11 are input-only, and triggers 12–15 are output-only but this is not required.

---

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

In both halt and resume groups, group 0 is special. Harts in group 0 halt/resume as if groups aren't implemented at all.

When the DM is reset, all harts must be placed in the lowest-numbered halt and resume groups that they can be in. (This will usually be group 0.)

Some designs may choose to hardcode hart groups to a group other than group 0, meaning it is never possible to halt or resume just a single hart. This is explicitly allowed. In that case it must be possible to discover the groups by using `dmcs2` even if it's not possible to change the configuration.

## 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 (running, halted, or held in reset) 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 or if bits that are defined as 0 aren't 0, then the DM must set `cmderr` to 2 (not supported).

---

*Example: Every DM must support the Access Register command, but might 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`. If the debugger starts a new command while `busy` is set, `cmderr` becomes 1 (busy), the currently executing command still gets to run to completion, but any error generated by the currently executing command is lost. 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.

Таблица 3.1: Use of Data Registers

| Argument Width | arg0/return value | arg1 | arg2 |
|---|---|---|---|
| 32 | data0 | data1 | data2 |
| 64 | data0, data1 | data2, data3 | data4, data5 |
| 128 | data0–data3 | data4–data7 | data8–data11 |

---

*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 debugger can try to reset the hart (using hartreset or ndmreset). If that doesn't clear busy, then it can try resetting 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.

Таблица 3.2: Meaning of cmdtype

| cmdtype | Command | Page |
|---|---|---|
| 0 | Access Register Command | 18 |
| 1 | Quick Access | 19 |
| 2 | Access Memory Command | 20 |

#### 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 and transfer are set, increment regno. regno may also be incremented if aarpostincrement is set and transfer is clear.
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. It is recommended that if one register in a group is accessible, then all registers in that group are accessible, but each individual register (aside from GPRs) may be supported differently across read, write, and halt status.

Registers might not be accessible if they wouldn't be accessible by M mode code currently running. (E.g. `fflags` might not be accessible when `mstatus`.FS is 0.) If this is the case, the debugger is responsible for changing state to make the registers accessible. The Core Debug Registers (Section 4.8) should be accessible if abstract CSR access is implemented.

Таблица 3.3: Abstract Register Numbers

| Numbers | Group Description |
|---|---|
| 0x0000 – 0x0fff | CSRs. The “PC” can be accessed here through dpc. |
| 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 sbcs.*

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 aarsize specifies a size larger than the register's actual size, then the access must fail. If a register is accessible, then reads of aarsize 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, regno is incremented. Incrementing past the highest supported value causes regno to become UNSPECIFIED. Supporting this variant is optional. It is undefined whether the increment happens when transfer is 0. |
| postexec | 0: No effect. This variant must be supported, and is the only supported one if progbufsize 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 write.<br>1: Do the operation specified by write.<br>This bit can be used to just execute the Program Buffer without having to worry about placing valid values into aarsize or regno. |
| write | When transfer is set: 0: Copy data from the specified register into `arg0` portion of `data`.<br>1: Copy data from `arg0` portion of `data` into the specified register. |
| regno | Number of the register to access, as described in Table 3.3. dpc 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," the Program Buffer execution ends, and the hart is halted with cause set to 3.
4. If the Program Buffer executed without an exception, then resume the hart.

Implementing this command is optional.

This command does not touch the `data` registers.

| 31–24 | 23–0 |
|---|---|
| cmdtype | 0 |
| 8 | 24 |

| 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* sbcs.

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

| 31–24 | 23 | 22–20 | 19 |
|---|---|---|---|
| cmdtype | aamvirtual | aamsize | aampostincrement |
| 8 | 1 | 3 | 1 |

| 18–17 | 16 | 15–14 | 13–0 |
|---|---|---|---|
| 0 | write | target-specific | 0 |
| 2 | 1 | 2 | 14 |

| Field | Description |
|---|---|
| cmdtype | This is 2 to indicate Access Memory Command. |
| aamvirtual | 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.<br>Debug Modules on systems without address translation (i.e. virtual addresses equal physical) may optionally allow aamvirtual set to 1, which would produce the same result as that same abstract command with aamvirtual cleared. |
| aamsize | 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. |
| aampostincrement | After a memory access has completed, if this bit is 1, increment `arg1` (which contains the address used) by the number of bytes encoded in aamsize.<br>Supporting this variant is optional, but highly recommended for performance reasons. |
| write | 0: Copy data from the memory location specified in `arg1` into the low bits of `arg0`. Any remaining bits of `arg0` now have an undefined value.<br>1: Copy data from the low bits of `arg0` into the memory location specified in `arg1`. |
| target-specific | 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. DMs 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 implicit `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 cause the value of dpc to become UNSPECIFIED. 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 dpc to become UNSPECIFIED upon Program Buffer execution 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 Hart Debug 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

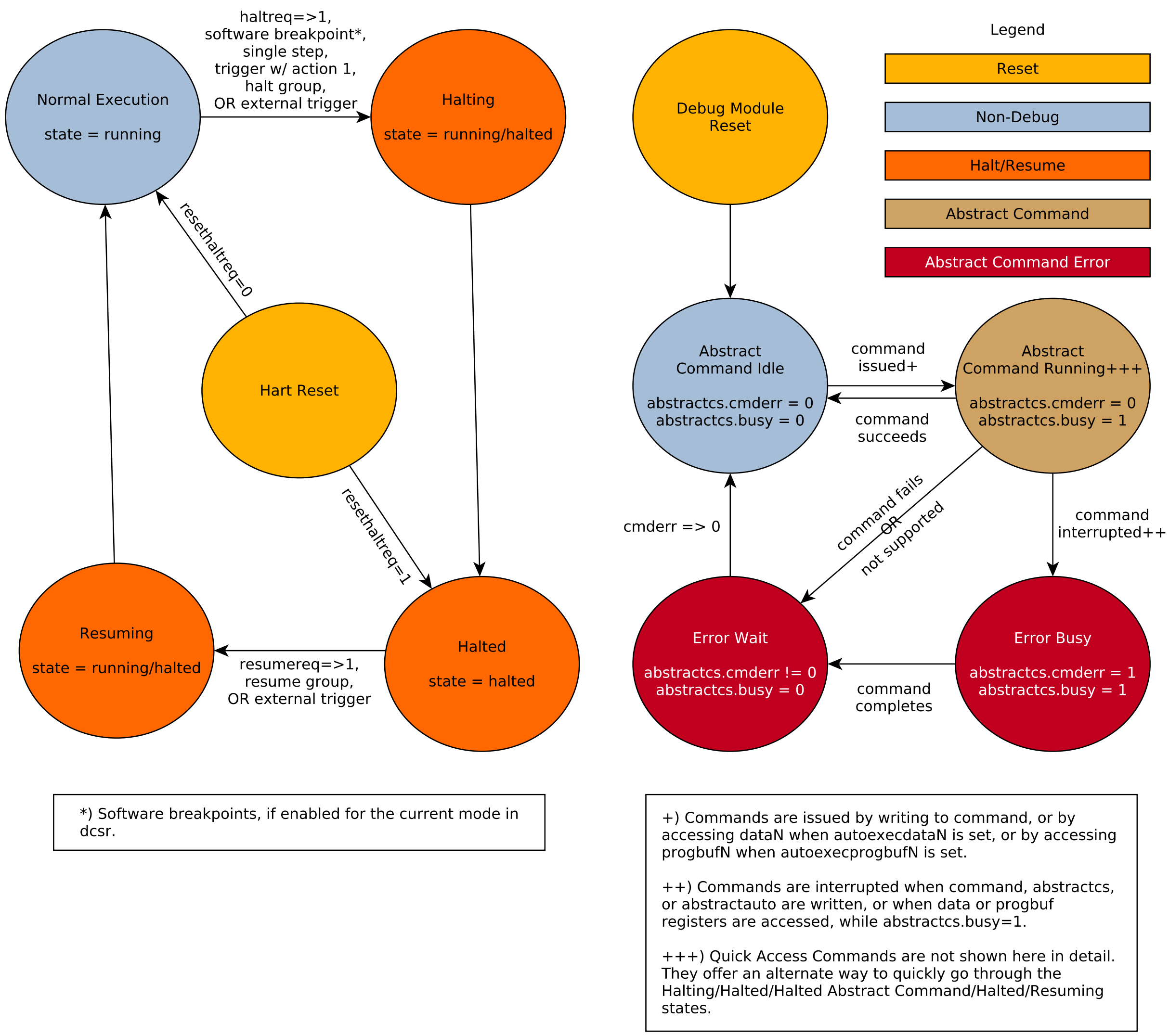


Рис. 3.1: Run/Halt Debug State Machine for single-hart hardware platforms. As only a small amount of state is visible to the debugger, the states and transitions are conceptual.

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 `sbdata` are used for each access size.

Таблица 3.7: System Bus Data Bits

| Access Size | Data Bits |
|---|---|
| 8 | `sbdata0` bits 7:0 |
| 16 | `sbdata0` bits 15:0 |
| 32 | `sbdata0` |
| 64 | `sbdata1`, `sbdata0` |
| 128 | `sbdata3`, `sbdata2`, `sbdata1`, `sbdata0` |

Depending on the microarchitecture, data accessed through System Bus Access might 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.

---

*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 `hartinfo`, this can be used to quickly perform a memory or register access. For some hardware platforms this will be too intrusive, but many hardware platforms 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 hardware platform, nor expose details about the harts connected to the DM. All DM registers should read 0, while writes should be ignored, with the following mandatory exceptions:

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

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

## 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 dmcontrol until dmactive is high.
4. 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 hardware platform 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 or non-existent 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. `sbcs`), or by checking bits in another register (e.g. progbufsize).

Таблица 3.8: Debug Module Debug Bus Registers

| Address | Name | Page |
|---|---|---|
| 0x04 | Abstract Data 0 (`data0`) | 39 |
| 0x05 | Abstract Data 1 (`data1`) | |
| 0x06 | Abstract Data 2 (`data2`) | |
| 0x07 | Abstract Data 3 (`data3`) | |
| 0x08 | Abstract Data 4 (`data4`) | |
| 0x09 | Abstract Data 5 (`data5`) | |
| 0x0a | Abstract Data 6 (`data6`) | |
| 0x0b | Abstract Data 7 (`data7`) | |
| 0x0c | Abstract Data 8 (`data8`) | |
| 0x0d | Abstract Data 9 (`data9`) | |
| 0x0e | Abstract Data 10 (`data10`) | |
| 0x0f | Abstract Data 11 (`data11`) | |
| 0x10 | Debug Module Control (`dmcontrol`) | 30 |
| 0x11 | Debug Module Status (`dmstatus`) | 28 |
| 0x12 | Hart Info (`hartinfo`) | 33 |
| 0x13 | Halt Summary 1 (`haltsum1`) | 41 |
| 0x14 | Hart Array Window Select (`hawindowsel`) | 34 |
| 0x15 | Hart Array Window (`hawindow`) | 35 |
| 0x16 | Abstract Control and Status (`abstractcs`) | 35 |
| 0x17 | Abstract Command (`command`) | 36 |
| 0x18 | Abstract Command Autoexec (`abstractauto`) | 37 |
| 0x19 | Configuration String Pointer 0 (`confstrptr0`) | 37 |
| 0x1a | Configuration String Pointer 1 (`confstrptr1`) | 38 |
| 0x1b | Configuration String Pointer 2 (`confstrptr2`) | 38 |
| 0x1c | Configuration String Pointer 3 (`confstrptr3`) | 38 |
| 0x1d | Next Debug Module (`nextdm`) | 38 |
| 0x1f | Custom Features (`custom`) | 48 |
| 0x20 | Program Buffer 0 (`progbuf0`) | 39 |
| 0x21 | Program Buffer 1 (`progbuf1`) | |

*Continued on next page*

Таблица 3.8: Debug Module Debug Bus Registers

| Address | Name | Page |
|---|---|---|
| 0x22 | Program Buffer 2 (`progbuf2`) | |
| 0x23 | Program Buffer 3 (`progbuf3`) | |
| 0x24 | Program Buffer 4 (`progbuf4`) | |
| 0x25 | Program Buffer 5 (`progbuf5`) | |
| 0x26 | Program Buffer 6 (`progbuf6`) | |
| 0x27 | Program Buffer 7 (`progbuf7`) | |
| 0x28 | Program Buffer 8 (`progbuf8`) | |
| 0x29 | Program Buffer 9 (`progbuf9`) | |
| 0x2a | Program Buffer 10 (`progbuf10`) | |
| 0x2b | Program Buffer 11 (`progbuf11`) | |
| 0x2c | Program Buffer 12 (`progbuf12`) | |
| 0x2d | Program Buffer 13 (`progbuf13`) | |
| 0x2e | Program Buffer 14 (`progbuf14`) | |
| 0x2f | Program Buffer 15 (`progbuf15`) | |
| 0x30 | Authentication Data (`authdata`) | 39 |
| 0x32 | Debug Module Control and Status 2 (`dmcs2`) | 40 |
| 0x34 | Halt Summary 2 (`haltsum2`) | 42 |
| 0x35 | Halt Summary 3 (`haltsum3`) | 42 |
| 0x37 | System Bus Address 127:96 (`sbaddress3`) | 45 |
| 0x38 | System Bus Access Control and Status (`sbcs`) | 42 |
| 0x39 | System Bus Address 31:0 (`sbaddress0`) | 44 |
| 0x3a | System Bus Address 63:32 (`sbaddress1`) | 45 |
| 0x3b | System Bus Address 95:64 (`sbaddress2`) | 45 |
| 0x3c | System Bus Data 31:0 (`sbdata0`) | 46 |
| 0x3d | System Bus Data 63:32 (`sbdata1`) | 47 |
| 0x3e | System Bus Data 95:64 (`sbdata2`) | 47 |
| 0x3f | System Bus Data 127:96 (`sbdata3`) | 47 |
| 0x40 | Halt Summary 0 (`haltsum0`) | 41 |
| 0x70 | Custom Features 0 (`custom0`) | 48 |
| 0x71 | Custom Features 1 (`custom1`) | |
| 0x72 | Custom Features 2 (`custom2`) | |
| 0x73 | Custom Features 3 (`custom3`) | |
| 0x74 | Custom Features 4 (`custom4`) | |
| 0x75 | Custom Features 5 (`custom5`) | |
| 0x76 | Custom Features 6 (`custom6`) | |
| 0x77 | Custom Features 7 (`custom7`) | |
| 0x78 | Custom Features 8 (`custom8`) | |
| 0x79 | Custom Features 9 (`custom9`) | |
| 0x7a | Custom Features 10 (`custom10`) | |
| 0x7b | Custom Features 11 (`custom11`) | |
| 0x7c | Custom Features 12 (`custom12`) | |
| 0x7d | Custom Features 13 (`custom13`) | |
| 0x7e | Custom Features 14 (`custom14`) | |
| 0x7f | Custom Features 15 (`custom15`) | |

### 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.

| 31 25 | 24 | 23 | 22 | 21 20 | 19 |
|---|---|---|---|---|---|
| 0 | ndmresetpending | stickyunavail | impebreak | 0 | allhavereset |
| 7 | 1 | 1 | 1 | 2 | 1 |

| 18 | 17 | 16 | 15 | 14 | 13 |
|---|---|---|---|---|---|
| anyhavereset | allresumeack | anyresumeack | allnonexistent | anynonexistent | allunavail |
| 1 | 1 | 1 | 1 | 1 | 1 |

| 12 | 11 | 10 | 9 | 8 | 7 |
|---|---|---|---|---|---|
| anyunavail | allrunning | anyrunning | allhalted | anyhalted | authenticated |
| 1 | 1 | 1 | 1 | 1 | 1 |

| 6 | 5 | 4 | 3 0 |
|---|---|---|---|
| authbusy | hasresethaltreq | confstrptrvalid | version |
| 1 | 1 | 1 | 4 |

| Field | Description | Access | Reset |
|---|---|---|---|
| ndmresetpending | 0: Unimplemented, or ndmreset is zero and no ndmreset is currently in progress.<br>1: ndmreset is currently nonzero, or there is an ndmreset in progress. | R | - |
| stickyunavail | 0: The per-hart `unavail` bits reflect the current state of the hart.<br>1: The per-hart `unavail` bits are sticky. Once they are set, they will not clear until the debugger acknowledges them using ackunavail. | R | Preset |
| impebreak | If 1, then there is an implicit `ebreak` instruction at the non-existent word immediately after the Program Buffer. This saves the debugger from having to write the `ebreak` itself, and allows the Program Buffer to be one word smaller.<br>This must be 1 when progbufsize 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 | - |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| allnonexistent | This field is 1 when all currently selected harts do not exist in this hardware platform. | R | - |
| anynonexistent | This field is 1 when any currently selected hart does not exist in this hardware platform. | R | - |
| allunavail | This field is 1 when all currently selected harts are unavailable, or (if stickyunavail is 1) were unavailable without that being acknowledged. | R | - |
| anyunavail | This field is 1 when any currently selected hart is unavailable, or (if stickyunavail is 1) was unavailable without that being acknowledged. | R | - |
| allrunning | This field is 1 when all currently selected harts are running. | R | - |
| anyrunning | This field is 1 when any currently selected hart is running. | R | - |
| allhalted | This field is 1 when all currently selected harts are halted. | R | - |
| anyhalted | This field is 1 when any currently selected hart is halted. | R | - |
| authenticated | 0: 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 `authdata`.<br>1: The authentication module is busy. Accessing `authdata` results in unspecified behavior.<br>authbusy only becomes set in immediate response to an access to `authdata`. | R | 0 |
| hasresethaltreq | 1 if this Debug Module supports halt-on-reset functionality controllable by the setresethaltreq and clrresethaltreq bits. 0 otherwise. | R | Preset |
| confstrptrvalid | 0: `confstrptr0`–`confstrptr3` hold information which is not relevant to the configuration string.<br>1: `confstrptr0`–`confstrptr3` 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>3: There is a Debug Module and it conforms to version 1.0 of this specification.<br>15: There is a Debug Module but it does not conform to any available version of this spec. | R | 3 |

### 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.

keepalive is an optional internal bit of per-hart state. When it is set, it suggests that the hardware should attempt to keep the hart available for the debugger, e.g. by keeping it from entering a low-power state once powered on. Even if the bit is implemented, hardware might not be able to keep a hart available. The bit is written through setkeepalive and clrkeepalive.

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

| 31 | 30 | 29 | 28 | 27 |
|---|---|---|---|---|
| haltreq | resumereq | hartreset | ackhavereset | ackunavail |
| 1 | 1 | 1 | 1 | 1 |

| 26 | 25 16 | 15 6 | 5 | 4 |
|---|---|---|---|---|
| hasel | hartsello | hartselhi | setkeepalive | clrkeepalive |
| 1 | 10 | 10 | 1 | 1 |

| 3 | 2 | 1 | 0 |
|---|---|---|---|
| setresethaltreq | clrresethaltreq | ndmreset | dmactive |
| 1 | 1 | 1 | 1 |

| Field | Description | Access | Reset |
|---|---|---|---|
| haltreq | 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 hartsel and hasel. | WARZ | - |
| resumereq | 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>resumereq is ignored if haltreq is set.<br>Writes apply to the new value of hartsel and hasel. | W1 | - |
| hartreset | This optional field writes the reset bit for all the currently selected harts. To perform a reset the debugger writes 1, and then writes 0 to deassert the reset signal.<br>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 hartsel and hasel. | WARL | 0 |
| ackhavereset | 0: No effect.<br>1: Clears `havereset` for any selected harts.<br>Writes apply to the new value of hartsel and hasel. | W1 | - |
| ackunavail | 0: No effect.<br>1: Clears `unavail` for any selected harts.<br>Writes apply to the new value of hartsel and hasel. | W1 | - |
| hasel | Selects the definition of currently selected harts.<br>0: There is a single currently selected hart, that is selected by hartsel.<br>1: There may be multiple currently selected harts – the hart selected by hartsel, 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. | WARL | 0 |
| hartsello | The low 10 bits of hartsel: the DM-specific index of the hart to select. This hart is always part of the currently selected harts. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| hartselhi | The high 10 bits of hartsel: the DM-specific index of the hart to select. This hart is always part of the currently selected harts. | WARL | 0 |
| setkeepalive | This optional field sets keepalive for all currently selected harts, unless clrkeepalive is simultaneously set to 1.<br>Writes apply to the new value of hartsel and hasel. | W1 | - |
| clrkeepalive | This optional field clears keepalive for all currently selected harts.<br>Writes apply to the new value of hartsel and hasel. | W1 | - |
| setresethaltreq | This optional field writes the halt-on-reset request bit for all currently selected harts, unless clrresethaltreq 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 clrresethaltreq to clear it.<br>Writes apply to the new value of hartsel and hasel.<br>If hasresethaltreq 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 hartsel and hasel. | W1 | - |
| ndmreset | This bit controls the reset signal from the DM to the rest of the hardware platform. The signal should reset every part of the hardware platform, including every hart, except for the DM and any logic required to access the DM. To perform a hardware platform 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 | This bit serves as a reset signal for the Debug Module itself. After changing the value of this bit, the debugger must poll dmcontrol until dmactive has taken the requested value before performing any action that assumes the requested dmactive state change has completed. Hardware may take an arbitrarily long time to complete activation or deactivation and will indicate completion by setting dmactive to the requested value.<br>0: The module's state, including authentication mechanism, takes its reset values (the dmactive bit is the only bit which can be written to something other than its reset value). Any accesses to the module may fail. Specifically, version might not return correct data.<br>1: The module functions normally.<br>No other mechanism should exist that may result in resetting the Debug Module after power up.<br>To place the Debug Module into a known state, a debugger may write 0 to dmactive, poll until dmactive is observed 0, write 1 to dmactive, and poll until dmactive is observed 1.<br>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. | 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.

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

| Field | Description | Access | Reset |
|---|---|---|---|
| nscratch | Number of `dscratch` registers available for the debugger to use during program buffer execution, starting from `dscratch0`. The debugger can make no assumptions about the contents of these registers between commands. | R | Preset |
| dataaccess | 0: The `data` 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 `data` registers are shadowed in the hart's memory map. Each register takes up 4 bytes in the memory map. | R | Preset |
| datasize | If dataaccess is 0: Number of CSRs dedicated to shadowing the `data` registers.<br>If dataaccess is 1: Number of 32-bit words in the memory map dedicated to shadowing the `data` registers.<br>Since there are at most 12 `data` registers, the value in this register must be 12 or smaller. | R | Preset |
| dataaddr | If dataaccess is 0: The number of the first CSR dedicated to shadowing the `data` registers.<br>If dataaccess is 1: Address of RAM where the data registers are shadowed. This address is sign extended giving a range of -2048 to 2047, easily addressed with a load or store using `x0` as the address register. | R | Preset |

### 3.14.4 Hart Array Window Select (`hawindowsel`, 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`.

| 31 15 | 14 0 |
|---|---|
| 0 | hawindowsel |
| 17 | 15 |

| Field | Description | Access | Reset |
|---|---|---|---|
| hawindowsel | The high bits of this field may be tied to 0, depending on how large the array mask register is. E.g. on a hardware platform 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 `hawindowsel`. I.e. bit 0 refers to hart `hawindowsel` $*32$, while bit 31 refers to hart `hawindowsel` $*32+31$.

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 hawindowsel.

| 31–0 |
|---|
| maskdata |
| 32 |

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

Writing this register while an abstract command is executing causes cmderr to become 1 (busy) once the command completes (busy becomes 0).

---

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

| 31–29 | 28–24 | 23–13 | 12 | 11 | 10–8 | 7–4 | 3–0 |
|---|---|---|---|---|---|---|---|
| 0 | progbufsize | 0 | busy | relaxedpriv | cmderr | 0 | datacount |
| 3 | 5 | 11 | 1 | 1 | 3 | 4 | 4 |

| Field | Description | Access | Reset |
|---|---|---|---|
| progbufsize | Size of the Program Buffer, in 32-bit words. Valid sizes are 0 - 16. | R | Preset |
| busy | 1: An abstract command is currently being executed.<br>This bit is set as soon as `command` is written, and is not cleared until that command has completed. | R | 0 |
| relaxedpriv | This optional bit controls whether program buffer and abstract memory accesses are performed with the exact and full set of permission checks that apply based on the current architectural state of the hart performing the access, or with a relaxed set of permission checks (e.g. PMP restrictions are ignored). The details of the latter are implementation-specific. When set to 0, full permissions apply; when set to 1, relaxed permissions apply. | WARL | Preset |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| cmderr | Gets set if an abstract command fails. The bits in this field remain set until they are cleared by writing 1 to them. No abstract command is started until the value is reset to 0.<br>This field only contains a valid value if busy is 0.<br>0 (none): No error.<br>1 (busy): An abstract command was executing while command, abstractcs, or abstractauto was written, or when one of the data or progbuf registers was read or written. This status is only written if cmderr contains 0.<br>2 (not supported): The command in command 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>6: Reserved for future use.<br>7 (other): The command failed for another reason. | R/W1C | 0 |
| datacount | Number of data 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 become 1 (busy) once the command completes (busy becomes 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.*

| 31 24 | 23 0 |
|---|---|
| cmdtype | control |
| 8 | 24 |

| Field | Description | Access | Reset |
|---|---|---|---|
| cmdtype | The type determines the overall functionality of this abstract command. | WARZ | 0 |
| control | This field is interpreted in a command-specific manner, described for each abstract command. | WARZ | 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 supported by setting bits and reading them back.

If this register is implemented then bits corresponding to implemented progbuf and data registers must be writable. Other bits must be hard-wired to 0.

Writing this register while an abstract command is executing causes cmderr to become 1 (busy) once the command completes (busy becomes 0).

| 31–16 | 15–12 | 11–0 |
|---|---|---|
| autoexecprogbuf | 0 | autoexecdata |
| 16 | 4 | 12 |

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

| 31 | 0 |
|---|---|
| addr | |
| 32 | |

### 3.14.10 Configuration String Pointer 1 (`confstrptr1`, at 0x1a)

When confstrptrvalid is set, reading this register returns bits 63:32 of the configuration string pointer. See `confstrptr0` for more details.

This entire register is read-only.

| 31 | 0 |
|---|---|
| addr | |
| 32 | |

### 3.14.11 Configuration String Pointer 2 (`confstrptr2`, at 0x1b)

When confstrptrvalid is set, reading this register returns bits 95:64 of the configuration string pointer. See `confstrptr0` for more details.

This entire register is read-only.

| 31 | 0 |
|---|---|
| addr | |
| 32 | |

### 3.14.12 Configuration String Pointer 3 (`confstrptr3`, at 0x1c)

When confstrptrvalid is set, reading this register returns bits 127:96 of the configuration string pointer. See `confstrptr0` for more details.

This entire register is read-only.

| 31 | 0 |
|---|---|
| addr | |
| 32 | |

### 3.14.13 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.

| 31 | 0 |
| --- | --- |
| addr | |
| 32 | |

### 3.14.14 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 might 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.

| 31 | 0 |
| --- | --- |
| data | |
| 32 | |

### 3.14.15 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.

| 31 | 0 |
| --- | --- |
| data | |
| 32 | |

### 3.14.16 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.

| 31 … 0 |
|---|
| data |
| 32 |

### 3.14.17 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.

If halt groups are not implemented, then group will always be 0 when grouptype is 0.

If resume groups are not implemented, then grouptype will remain 0 even after 1 is written there.

The DM external triggers available to add to halt groups may be the same as or distinct from the DM external triggers available to add to resume groups.

| 31 … 12 | 11 | 10 … 7 | 6 … 2 | 1 | 0 |
|---|---|---|---|---|---|
| 0 | grouptype | dmexttrigger | group | hgwrite | hgselect |
| 20 | 1 | 4 | 5 | 1 | 1 |

| Field | Description | Access | Reset |
|---|---|---|---|
| grouptype | 0: The remaining fields in this register configure halt groups.<br>1: The remaining fields in this register configure resume groups. | WARL | 0 |
| dmexttrigger | This field contains the currently selected DM 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 DM external triggers exist. | WARL | 0 |
| group | When hgselect is 0, contains the group of the hart specified by hartsel.<br>When hgselect is 1, contains the group of the DM external trigger selected by dmexttrigger.<br>Writes only have an effect if hgwrite is also written 1.<br>Group numbers are contiguous starting at 0, with the highest number being implementation-dependent, and possibly different between different group types. Debuggers should read back this field after writing to confirm they are using a hart group that is supported.<br>If groups aren't implemented, then this entire field is 0. | WARL | preset |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| hgwrite | When 1 is written and hgselect is 0, for every selected hart the DM will change its group to the value written to group, if the hardware supports that group for that hart. Implementations may also change the group of a minimal set of unselected harts in the same way, if that is necessary due to a hardware limitation.<br>When 1 is written and hgselect is 1, the DM will change the group of the DM external trigger selected by dmexttrigger to the value written to group, if the hardware supports that group for that trigger.<br>Writing 0 has no effect. | W1 | - |
| hgselect | 0: Operate on harts.<br>1: Operate on DM external triggers.<br>If there are no DM external triggers, this field must be tied to 0. | WARL | 0 |

### 3.14.18 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.

This register might not be present if fewer than 2 harts are connected to this DM.

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.

| 31 — 0 |
|---|
| haltsum0 |
| 32 |

### 3.14.19 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 might not be present if fewer than 33 harts are connected to this DM.

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.

| 31 0 |
|---|
| haltsum1 |
| 32 |

### 3.14.20 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 might not be present if fewer than 1025 harts are connected to this DM.

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

This entire register is read-only.

| 31 0 |
|---|
| haltsum2 |
| 32 |

### 3.14.21 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 might not be present if fewer than 32769 harts are connected to this DM.

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'hfffff.

This entire register is read-only.

| 31 0 |
|---|
| haltsum3 |
| 32 |

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

| 31 29 | 28 23 | 22 | 21 | 20 |
|---|---|---|---|---|
| sbversion | 0 | sbbusyerror | sbbusy | sbreadonaddr |
| 3 | 6 | 1 | 1 | 1 |

| 19 17 | 16 | 15 | 14 12 | 11 5 |
|---|---|---|---|---|
| sbaccess | sbautoincrement | sbreadondata | sberror | sbasize |
| 3 | 1 | 1 | 3 | 7 |

| 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| sbaccess128 | sbaccess64 | sbaccess32 | sbaccess16 | sbaccess8 |
| 1 | 1 | 1 | 1 | 1 |

| 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 sbbusy 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 sbcs while sbbusy is high result in undefined behavior. A debugger must not write to sbcs until it reads sbbusy as 0. | R | 0 |
| sbreadonaddr | When 1, every write to sbaddress0 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 sbaccess has an unsupported value when the DM starts a bus access, the access is not performed and sberror is set to 4. | R/W | 2 |
| sbautoincrement | When 1, `sbaddress` is incremented by the access size (in bytes) selected in sbaccess after every system bus access. | R/W | 0 |
| sbreadondata | When 1, every read from sbdata0 automatically triggers a system bus read at the (possibly auto-incremented) address. | R/W | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| sberror | 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 |
| sbasize | Width of system bus addresses in bits. (0 indicates there is no bus access support.) | R | Preset |
| sbaccess128 | 1 when 128-bit system bus accesses are supported. | R | Preset |
| sbaccess64 | 1 when 64-bit system bus accesses are supported. | R | Preset |
| sbaccess32 | 1 when 32-bit system bus accesses are supported. | R | Preset |
| sbaccess16 | 1 when 16-bit system bus accesses are supported. | R | Preset |
| sbaccess8 | 1 when 8-bit system bus accesses are supported. | R | Preset |

### 3.14.23 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.

| 31 … 0 |
|---|
| address |
| 32 |

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

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

If sbasize 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.

| 31 0 |
|---|
| address |
| 32 |

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

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

If sbasize 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.

| 31 0 |
|---|
| address |
| 32 |

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

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

If sbasize 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.

| 31 0 |
|---|
| address |
| 32 |

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

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

If all of the sbaccess bits in sbcs 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:
   (a) Perform a system bus read from the address contained in sbaddress, placing the result in sbdata.
   (b) If sbautoincrement is set and the read was successful, increment sbaddress.
4. Clear sbbusy.

Only sbdata0 has this behavior. The other sbdata registers have no side effects. On systems that have buses wider than 32 bits, a debugger should access sbdata0 after accessing the other sbdata registers.

| 31 … 0 |
|---|
| data |
| 32 |

| Field | Description | Access | Reset |
|---|---|---|---|
| data | Accesses bits 31:0 of sbdata. | R/W | 0 |

### 3.14.28 System Bus Data 63:32 (`sbdata1`, 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.

| 31 0 |
|---|
| data |
| 32 |

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

### 3.14.29 System Bus Data 95:64 (`sbdata2`, 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.

| 31 0 |
|---|
| data |
| 32 |

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

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

This register only exists if sbaccess128 is 1.

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

| 31 0 |
|---|
| data |
| 32 |

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

### 3.14.31 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.

### 3.14.32 Custom Features 0 (custom0, at 0x70)

The optional custom0 through custom15 registers may be used for non-standard features. Future versions of the debug spec will not use these addresses.

# Глава 4

# RISC-V Debug, ISA

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. Because the hart is halted, there is no forward progress in the normal instruction stream. How Debug Mode is implemented is not specified here.

When executing code due to an abstract command, 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. Full permission checks, or a relaxed set of permission checks, will apply according to relaxedpriv.
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. If stopcount is 0 then counters continue. If it is 1 then counters are stopped.
6. If stoptime is 0 then timers continue. If it is 1 then timers are stopped.
7. The `wfi` instruction acts as a `nop`.
8. Almost all instructions that change the privilege level have UNSPECIFIED behavior. This includes `ecall`, `mret`, `sret`, and `uret`. (To change the privilege level, the debugger can write prv and v in dcsr). The only exception is `ebreak`, which ends execution of the Program Buffer when executed.
9. 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 illegal instructions.

10. 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 illegal instructions.
11. Instructions that depend on the value of the PC (e.g. `auipc`) may act as illegal instructions.
12. Effective XLEN is DXLEN.
13. Forward progress is guaranteed.

---

*When mprven =1, the external debugger can set MPRV and MPP appropriately to have hardware perform memory accesses with the appropriate endianness, address translation, permission checks, and PMP/PMA checks (subject to relaxedpriv). This is also the only way to access all of physical memory when 34-bit physical addresses are supported on a Sv32 hart. If hardware ties mprven to 0 then the external debugger is expected to simulate all the effects of MPRV, including any extensions that affect memory accesses. For these reasons it is recommended to tie mprven to 1.*

## 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

### 4.4.1 Step Bit In Dcsr

This method is only available to external debuggers, and is the preferred way to single step.

An external debugger can cause a halted hart to execute a single instruction or trap and then re-enter Debug Mode by setting step before resuming. If step is set when a hart resumes then it will single step, regardless of the reason for resuming.

If control is transferred to a trap handler while executing the instruction, then Debug Mode is re-entered immediately after the PC is changed to the trap handler, and the appropriate `tval` and `cause` registers are updated. In this case none of the trap handler is executed, and if the cause was a pending interrupt no instructions might be executed at all.

If executing or fetching the instruction causes a trigger to fire with action=1, 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`.

### 4.4.2 Icount Trigger

Native debuggers won't have access to `dcsr`, but can use the `icount` trigger by setting `count` to 1.

This approach does have some limitations:

1. Interrupts will fire as usual. Debuggers that want to disable interrupts while stepping must disable them by changing `mstatus`, and specially handle instructions that read `mstatus`.
2. `wfi` instructions are not treated specially and might take a very long time to complete.

## 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 Resume

When a hart resumes:

1. `pc` changes to the value stored in `dpc`.
2. The current privilege mode and virtualization mode are changed to that specified by prv and v.
3. If the new privilege mode is less privileged than M-mode, MPRV in `mstatus` is cleared.

4. The hart is no longer in debug mode.

## 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 `misa`) 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.

Таблица 4.1: Core Debug Registers

| Address | Name | Page |
|---|---|---|
| 0x7b0 | Debug Control and Status (`dcsr`) | 52 |
| 0x7b1 | Debug PC (`dpc`) | 55 |
| 0x7b2 | Debug Scratch Register 0 (`dscratch0`) | 56 |
| 0x7b3 | Debug Scratch Register 1 (`dscratch1`) | 56 |

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

---

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

This CSR is read/write.

| 31 28 | 27 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 |
|---|---|---|---|---|---|---|---|---|
| debugver | 0 | ebreakvs | ebreakvu | ebreakm | 0 | ebreaks | ebreaku | stepie |
| 4 | 10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| 10 | 9 | 8 6 | 5 | 4 | 3 | 2 | 1 0 |
|---|---|---|---|---|---|---|---|
| stopcount | stoptime | cause | v | mprven | nmip | step | prv |
| 1 | 1 | 3 | 1 | 1 | 1 | 1 | 2 |

| Field | Description | Access | Reset |
|---|---|---|---|
| debugver | 0: There is no debug support.<br>4: Debug support exists as it is described in this document.<br>15: There is debug support, but it does not conform to any available version of this spec. | R | Preset |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| ebreakvs | 0: `ebreak` instructions in VS-mode behave as described in the Privileged Spec.<br>1: `ebreak` instructions in VS-mode enter Debug Mode.<br>This bit is hardwired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| ebreakvu | 0: `ebreak` instructions in VU-mode behave as described in the Privileged Spec.<br>1: `ebreak` instructions in VU-mode enter Debug Mode.<br>This bit is hardwired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| ebreakm | 0: `ebreak` instructions in M-mode behave as described in the Privileged Spec.<br>1: `ebreak` instructions in M-mode enter Debug Mode. | R/W | 0 |
| ebreaks | 0: `ebreak` instructions in S-mode behave as described in the Privileged Spec.<br>1: `ebreak` instructions in S-mode enter Debug Mode.<br>This bit is hardwired to 0 if the hart does not support S-mode. | WARL | 0 |
| ebreaku | 0: `ebreak` instructions in U-mode behave as described in the Privileged Spec.<br>1: `ebreak` instructions in U-mode enter Debug Mode.<br>This bit is hardwired to 0 if the hart does not support U-mode. | WARL | 0 |
| stepie | 0: Interrupts (including NMI) are disabled during single stepping.<br>1: Interrupts (including NMI) 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 |
|---|---|---|---|
| stopcount | 0: Increment counters as usual.<br>1: Don't increment any hart-local counters while in Debug Mode or on `ebreak` instructions that cause entry into Debug Mode. These counters include the `instret` CSR. On single-hart cores `cycle` 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 |
| stoptime | 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 |
| cause | Explains why Debug Mode was entered.<br>When there are multiple reasons to enter Debug Mode in a single cycle, hardware should set cause to the cause with the highest priority.<br>1: An `ebreak` instruction was executed. (priority 3)<br>2: A Trigger Module trigger fired with action=0. (priority 4)<br>3: The debugger requested entry to Debug Mode using haltreq. (priority 1)<br>4: The hart single stepped because step was set. (priority 0, lowest)<br>5: The hart halted directly out of reset due to resethaltreq. 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 |
| v | Extends the prv field with the virtualization mode 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 virtualization mode when exiting Debug Mode. This bit is hardwired to 0 on harts that do not support virtualization mode. | WARL | 0 |
| mprven | 0: MPRV in `mstatus` is ignored in Debug Mode.<br>1: MPRV in `mstatus` takes effect in Debug Mode.<br>Implementing this bit is optional. It may be tied to either 0 or 1. | WARL | Preset |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| nmip | 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 |
| step | When set and not in Debug Mode, the hart will only execute a single instruction and then enter Debug Mode. See Section 4.4.1 for details.<br>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. | WARL | 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.

Таблица 4.3: Virtual address in DPC upon Debug Mode Entry

| Cause | Virtual Address in DPC |
|---|---|
| `ebreak` | Address of the `ebreak` instruction |
| single step | Address of the instruction that would be executed next if no debugging was going on. Ie. `pc` + 4 for 32-bit instructions that don't change program flow, the destination PC on taken jumps/branches, etc. |
| trigger module | The address of the next instruction to be executed at the time that debug mode was entered. If the trigger is `mcontrol` or `mcontrol6` and `timing` is 0, this corresponds to the address of the instruction which caused the trigger to fire. |
| 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.

This CSR is read/write.

| DXLEN-1 … 0 |
|---|
| dpc |
| DXLEN |

### 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.

Таблица 4.4: Virtual Core Debug Registers

| Address | Name | Page |
|---|---|---|
| virtual | Privilege Level (`priv`) | 56 |

### 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` and `v` 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.

| 2 | 1 … 0 |
|---|---|
| v | prv |
| 1 | 2 |

Таблица 4.5: Privilege Level and Virtualization Mode Encoding

| H extension supported | v | prv | Abbreviation | Name |
|---|---|---|---|---|
| No | 0 | 0 | U-mode | User mode |
| No | 0 | 1 | S-mode | Supervisor mode |
| No | 0 | 3 | M-mode | Machine mode |
| Yes | 0 | 0 | U-mode | User mode |
| Yes | 0 | 1 | HS-mode | Hypervisor-enabled supervisor mode |
| Yes | 0 | 3 | M-mode | Machine mode |
| Yes | 1 | 0 | VU-mode | Virtual user mode |
| Yes | 1 | 1 | VS-mode | Virtual supervisor mode |

| Field | Description | Access | Reset |
|---|---|---|---|
| v | Contains the virtualization mode the hart was operating in when Debug Mode was entered. The encoding is described in Table 4.5, and matches the virtualization mode encoding from the Privileged Spec. A user can write this value to change the hart's virtualization mode when exiting Debug Mode. | WARL | 0 |
| prv | 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 |

## Глава 5

# Trigger Module (TM), ISA

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 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 might not exist at all and accessing them results in an illegal instruction exception.

Triggers do not fire while in Debug Mode.

## 5.1 Enumeration

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.*

## 5.2 Actions

Triggers can be configured to take one of several actions when they fire. Table 5.1 lists all options.

Таблица 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.) `xepc` must contain the virtual address of the next instruction that must be executed to preserve the program flow. |
| 1 | Enter Debug Mode. dpc must contain the virtual address of the next instruction that must be executed to preserve the program flow.<br>This action is only legal when the trigger's dmode is 1. Since the `tdata` registers are WARL, hardware should clear the action field whenever the action field is 1, dmode is cleared, and the new value of the action field would also be 1. |
| 2 – 5 | Reserved for use by the trace specification. |
| 8 – 9 | Signal the firing of the trigger to other blocks within the hart (e.g. as countable events to hpmcounters). Use external debug trigger output 0 or 1 (respectively). |
| other | Reserved for future use. |

## 5.3 Priority

Table 5.2 lists the synchronous exceptions from the Privileged Spec, and where the various types of triggers fit in. The first 3 columns come from the Privileged Spec, and the final column shows where triggers fit in. Priorities in the table are separated by horizontal lines, so e.g. etrigger and itrigger have the same priority. If this table contradicts the table in the Privileged Spec, then the latter takes precedence.

This table only applies if triggers are precise. Otherwise triggers will fire some indeterminate time after the event, and the priority is irrelevant. When triggers are chained, the priority is the lowest priority of the triggers in the chain.

| Priority | Exception Code | Description | Trigger |
|---|---|---|---|
| *Highest* | 3 | | etrigger |
| | 3 | | icount |
| | 3 | | itrigger |
| | 3 | | mcontrol/mcontrol6 after (on previous instruction) |
| | 3 | Instruction address breakpoint | mcontrol/mcontrol6 execute address before |
| | 12 | Instruction page fault | |
| | 1 | Instruction access fault | |
| | 3 | | mcontrol/mcontrol6 execute data before |
| | 2 | Illegal instruction | |
| | 0 | Instruction address misaligned | |
| | 8, 9, 11 | Environment call | |
| | 3 | Environment break | |
| | 3 | Load/Store/AMO address breakpoint | mcontrol/mcontrol6 load/store address before |
| | 3 | | mcontrol/mcontrol6 store data before |
| | 6 | Store/AMO address misaligned | |
| | 4 | Load address misaligned | |
| | 15 | Store/AMO page fault | |
| | 13 | Load page fault | |
| | 7 | Store/AMO access fault | |
| | 5 | Load access fault | |
| *Lowest* | 3 | | mcontrol/mcontrol6 load data before |

Таблица 5.2: Synchronous exception priority in decreasing priority order.

When multiple triggers in the same priority fire at once, hit (if implemented) is set for all of them. If one of these triggers has the “enter Debug Mode” action (1) and another trigger has the “raise a breakpoint exception” action (0), 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.

## 5.4 Native Triggers

*Triggers can be used for native debugging when* action *=0. If supported by the hart and desired by the debugger, triggers will often be programmed to have* m *=0 so that when they fire they cause a breakpoint exception to trap to a more privileged mode. That breakpoint exception can either be taken in M-mode or it can be delegated to a less privileged mode. However, it is possible for triggers to fire in the same mode that the resulting exception will be handled in.*

*In particular, when* action *=0:*

1. *mcontrol and mcontrol6 triggers with* m *=1 can cause a breakpoint exception that is taken from M-mode to M-mode (regardless of delegation).*
2. *mcontrol and mcontrol6 triggers with* s *=1 can cause a breakpoint exception that is taken from S-mode to S-mode if* `medeleg` *[3]=1.*
3. *mcontrol6 triggers with* vs *=1 can cause a breakpoint exception that is taken from VS-mode to VS-mode if* `medeleg` *[3]=1 and* `hedeleg` *[3]=1.*
4. *icount triggers with* m *=1 can cause a breakpoint exception that is taken from M-mode to M-mode (regardless of delegation).*
5. *icount triggers with* s *=1 can cause a breakpoint exception that is taken from S-mode to S-mode if* `medeleg` *[3]=1.*
6. *icount triggers with* vs *=1 can cause a breakpoint exception that is taken from VS-mode to VS-mode if* `medeleg` *[3]=1 and* `hedeleg` *[3]=1.*
7. *etrigger and itrigger triggers will always be taken from a trap handler before the first instruction of the handler. If etrigger/itrigger is set to trigger on exception/interrupt X and if X is delegated to mode Y then the trigger will cause a breakpoint exception that is taken from mode Y to mode Y unless breakpoint exceptions are delegated to a more privileged mode than Y.*
8. *tmexttrigger triggers are asynchronous and may occur in any mode and at any time.*

*In these cases such a trigger may cause a breakpoint exception while already in a trap handler. This might leave the hart unable to resume normal execution because state such as* `mcause` *and* `mepc` *would be overwritten.*

Harts that support triggers with action =0 should implement one of the following two solutions to solve the problem of reentrancy:

1. The hardware prevents triggers with action =0 from firing while in M-mode and while MIE in `mstatus` is 0. If `medeleg` [3]=1 then it prevents triggers with action =0 from firing while in S-mode and while SIE in `sstatus` is 0. If `medeleg` [3]=1 and `hedeleg` [3]=1 then it prevents triggers with action =0 from firing while in VS-mode and while SIE in `vsstatus` is 0.
2. mte and mpte in `tcontrol` is implemented. `medeleg` [3] is hard-wired to 0.

---

*The first option has the limitation that interrupts might be disabled at times when a user still might want triggers to fire. It has the benefit that breakpoints are not required to be handled in M-mode.*

*The second option has the benefit that it only disables triggers during the trap handler, though it requires specific software support for this debug feature in the M-mode trap handlers. It can only work if breakpoints are not delegated to less privileged modes and therefore targets primarily implementations without S-mode.*

*Because* `tcontrol` *is not accessible to S-mode, the second option can not be extended to accommodate delegation without adding additional S-mode and VS-mode CSRs.*

*Both options prevent etrigger and itrigger from having any effect on exceptions and interrupts that are handled in M-mode. They also prevent triggering during some initial portion of each handler. Debuggers should use other mechanisms to debug these cases, such as patching the handler or setting a breakpoint on the instruction after* MIE *is cleared.*

## 5.5 Trigger Registers

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

Almost all 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 must not modify the contents of other `tdata` registers, nor the configuration of any trigger besides the one that is currently selected.

The combination of these rules means that a debugger cannot simply set a trigger by writing `tdata1`, then `tdata2`, etc. The current value of `tdata2` might not be legal with the new value of `tdata1`. To help with this situation, it is guaranteed that writing 0 to `tdata1` disables the trigger, and leaves it in a state where `tdata2` and `tdata3` can be written with any value that makes sense for any trigger type supported by this trigger.

As a result, a debugger can write any supported trigger as follows:

1. Write 0 to `tdata1`.
2. Write desired values to `tdata2` and `tdata3`.
3. Write desired value to `tdata1`.

The trigger registers, except `scontext` and `hcontext`, 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. On systems where those values of XLEN can differ, this is handled as follows. Fields retain their values regardless of XLEN, which only affects where in the register these fields appear (e.g. `type`). Some fields are wider when XLEN is 64 than when it is 32 (e.g. `svalue`). The high bits in such fields retain their value but are not readable when XLEN is 32. A modification of a register when XLEN is 32 clears any inaccessible bits in that register.

Таблица 5.3: Trigger Registers

| Address | Name | Page |
|---|---|---|
| 0x5a8 | Supervisor Context (`scontext`) | 67 |
| 0x6a8 | Hypervisor Context (`hcontext`) | 66 |
| 0x7a0 | Trigger Select (`tselect`) | 63 |
| 0x7a1 | Trigger Data 1 (`tdata1`) | 63 |
| 0x7a1 | Match Control (`mcontrol`) | 68 |
| 0x7a1 | Match Control Type 6 (`mcontrol6`) | 74 |
| 0x7a1 | Instruction Count (`icount`) | 81 |
| 0x7a1 | Interrupt Trigger (`itrigger`) | 82 |
| 0x7a1 | Exception Trigger (`etrigger`) | 83 |
| 0x7a1 | External Trigger (`tmexttrigger`) | 84 |
| 0x7a2 | Trigger Data 2 (`tdata2`) | 65 |
| 0x7a3 | Trigger Data 3 (`tdata3`) | 65 |
| 0x7a3 | Trigger Extra (RV32) (`textra32`) | 85 |
| 0x7a3 | Trigger Extra (RV64) (`textra64`) | 86 |
| 0x7a4 | Trigger Info (`tinfo`) | 65 |

*Continued on next page*

Таблица 5.3: Trigger Registers

| Address | Name | Page |
|---|---|---|
| 0x7a5 | Trigger Control (`tcontrol`) | 66 |
| 0x7a8 | Machine Context (`mcontext`) | 67 |
| 0x7aa | Machine Supervisor Context (`mscontext`) | 67 |

### 5.5.1 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 external debugger must restore this register if it modifies it.

This CSR is read/write.

| XLEN-1 0 |
|---|
| index |
| XLEN |

### 5.5.2 Trigger Data 1 (`tdata1`, at 0x7a1)

This register is optional if no triggers are implemented.

This CSR is read/write.

| XLEN-1 XLEN-4 | XLEN-5 | XLEN-6 0 |
|---|---|---|
| type | dmode | data |
| 4 | 1 | XLEN - 5 |

| Field | Description | Access | Reset |
|---|---|---|---|
| type | 0: There is no trigger at this tselect.<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 mcontrol.<br>3: The trigger is an instruction count trigger. The remaining bits in this register act as described in icount.<br>4: The trigger is an interrupt trigger. The remaining bits in this register act as described in itrigger.<br>5: The trigger is an exception trigger. The remaining bits in this register act as described in etrigger.<br>6: The trigger is an address/data match trigger. The remaining bits in this register act as described in mcontrol6. This is similar to a type 2 trigger, but provides additional functionality and should be used instead of type 2 in newer implementations.<br>7: The trigger is a trigger source external to the TM. The remaining bits in this register act as described in tmexttrigger.<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. | WARL | Preset |
| dmode | If type is 0, then this bit is hard-wired to 0.<br>0: Both Debug and M-mode can write the tdata registers at the selected tselect.<br>1: Only Debug Mode can write the tdata registers at the selected tselect. Writes from other modes are ignored.<br>This bit is only writable from Debug Mode. In ordinary use, external debuggers will always set this bit when configuring a trigger. When clearing this bit, debuggers should also clear the action field (whose location depends on type). | WARL | 0 |
| data | If type is 0, then this field is hard-wired to 0.<br>Trigger-specific data. | WARL | Preset |

### 5.5.3 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.

This CSR is read/write.

| XLEN-1    0 |
|---|
| data |
| XLEN |

### 5.5.4 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.

This CSR is read/write.

| XLEN-1    0 |
|---|
| data |
| XLEN |

### 5.5.5 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`.

Writing this read/write CSR has no effect.

| XLEN-1    16 | 15    0 |
|---|---|
| 0 | info |
| XLEN - 16 | 16 |

| Field | Description | Access | Reset |
|---|---|---|---|
| info | One bit for each possible `type` enumerated in `tdata1`. 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 |

### 5.5.6 Trigger Control (`tcontrol`, at 0x7a5)

This optional register is only accessible in M-mode and Debug Mode and provides various control bits related to triggers.

This CSR is read/write.

| XLEN-1 … 10 | 9 | 8 | 7 | 6 … 4 | 3 | 2 … 0 |
|---|---|---|---|---|---|---|
| 0 | hcxe | scxe | mpte | 0 | mte | 0 |
| XLEN - 10 | 1 | 1 | 1 | 3 | 1 | 3 |

| Field | Description | Access | Reset |
|---|---|---|---|
| hcxe | `hcontext` enable.<br>0: `hcontext` is set to 0 and writes are ignored.<br>1: `hcontext` may be written and read. | WARL | 0 |
| scxe | `scontext` enable.<br>0: `scontext` is set to 0 and writes are ignored.<br>1: `scontext` may be written and read.<br>Enabling `scontext` can be a security risk in a virtualized system with a hypervisor that does not swap `scontext`. | WARL | 0 |
| mpte | M-mode previous trigger enable field.<br>mpte and mte provide one solution to a problem regarding triggers with action=0 firing in M-mode trap handlers. See Section 5.4 for more details.<br>When a trap into M-mode is taken, mpte is set to the value of mte. | WARL | 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, mte is set to 0. When `mret` is executed, mte is set to the value of mpte. | WARL | 0 |

### 5.5.7 Hypervisor Context (`hcontext`, at 0x6a8)

This optional register is only accessible in S/HS-mode, M-mode and Debug Mode.

If the H extension is not implemented then this register is not implemented, though the underlying state may be accessible via the optional `mcontext` alias.

This CSR is read/write.

| XLEN-1 | 0 |
|---|---|
| hcontext | |
| XLEN | |

| Field | Description | Access | Reset |
|---|---|---|---|
| hcontext | Hypervisor 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. If the H extension is not implemented, it's recommended to implement no more than 6 bits on RV32 and 13 on RV64 (as visible through the `mcontext` register). If the H extension is implemented, it's recommended to implement no more than 7 bits on RV32 and 14 on RV64. | WARL | 0 |

### 5.5.8 Supervisor Context (`scontext`, at 0x5a8)

This optional register is only accessible in S/HS-mode, VS-mode, M-mode and Debug Mode.

This CSR is read/write.

| XLEN-1 | 0 |
|---|---|
| data | |
| XLEN | |

| 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.<br>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. | WARL | 0 |

### 5.5.9 Machine Context (`mcontext`, at 0x7a8)

This optional register is an alias for `hcontext` and is only accessible in M-mode and Debug mode.

### 5.5.10 Machine Supervisor Context (`mscontext`, at 0x7aa)

This optional register is an alias for `scontext` included for backward compatibility (if desired).

### 5.5.11 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 5.10 suggests timings for the best user experience.

A chain of triggers that don't all have the same `timing` value will never fire. That means to implement the suggestions in Table 5.10, both timings should be supported on load address triggers.

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.

If the A extension is supported, then trigger behavior is as follows for the load and store bits:

1. `lr` instructions are loads
2. successful `sc` instructions are stores
3. it is UNSPECIFIED whether failing `sc` instructions are stores or not
4. Each AMO instruction is a load for the read portion of the operation. The address is always available to trigger on, although the value loaded might not be, depending on the hardware implementation.
5. Each AMO instruction is a store for the write portion of the operation. The address is always available to trigger on, although the value stored might not be, depending on the hardware implementation.

If the destination register of any load or AMO is `zero` then it is UNSPECIFIED whether a load trigger with `select` =1 will match. Whether store triggers with `select` =1 match on AMOs is UNSPECIFIED.

This CSR is read/write.

| XLEN-1 XLEN-4 | XLEN-5 | XLEN-6 XLEN-11 | XLEN-12 23 | 22 21 | 20 | 19 |
|---|---|---|---|---|---|---|
| type | dmode | maskmax | 0 | sizehi | hit | select |
| 4 | 1 | 6 | XLEN - 34 | 2 | 1 | 1 |

| 18 | 17 16 | 15 12 | 11 | 10 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| timing | sizelo | action | chain | match | m | 0 | s | u | execute | store | load |
| 1 | 2 | 4 | 1 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Field | Description | Access | Reset |
|---|---|---|---|
| maskmax | Specifies the largest naturally aligned powers-of-two (NAPOT) range supported by the hardware when match is 1. The value is the logarithm base 2 of the number of bytes in that range. A value of 0 indicates match 1 is not supported. A value of 63 corresponds to the maximum NAPOT range, which is $2^{63}$ bytes in size. | R | Preset |
| sizehi | This field only exists when XLEN is at least 64. It contains the 2 high bits of the access size. The low bits come from sizelo. See sizelo for how this is used. | WARL | 0 |
| hit | If this bit is implemented then it must become set when this trigger fires and may become set 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. | WARL | 0 |
| select | This bit determines the contents of the XLEN-bit compare values.<br>0: There is at least one compare value and it contains the lowest virtual address of the access. It is recommended that there are additional compare values for the other accessed virtual addresses. (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: There is exactly one compare value and it contains the data value loaded or stored, or the instruction executed. Any bits beyond the size of the data access will contain 0. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| timing | 0: The action for this trigger will be taken just before the instruction that triggered it is committed, but after all preceding instructions are committed. `xepc` or dpc (depending on action) must be set to the virtual address of the instruction that matched.<br>If this is combined with load and select =1 then a memory access will be performed (including any side effects of performing such an access) even though the load will not update its destination register. Debuggers should consider this when setting such breakpoints on, for example, memory-mapped I/O addresses.<br>1: The action for this trigger will be taken after the instruction that triggered it is committed. It should be taken before the next instruction is committed, but it is better to implement triggers imprecisely than to not implement them at all. `xepc` or dpc (depending on action) must be set to the virtual address of the next instruction that must be executed to preserve the program flow.<br>Most hardware will only implement one timing or the other, possibly dependent on select, execute, load, and store. This bit primarily exists for the hardware to communicate to the debugger what will happen. Hardware may implement the bit fully writable, in which case the debugger has a little more control.<br>Data load triggers with timing 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 timing of 1.<br>If a trigger with timing of 0 matches, it is implementation-dependent whether that prevents a trigger with timing of 1 matching as well. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| sizelo | This field contains the 2 low bits of the access size. The high bits come from sizehi. The combined value is interpreted as follows:<br>0: The trigger will attempt to match against an access of any size. The behavior is only well-defined if select = 0, or if the access size is XLEN.<br>1: The trigger will only match against 8-bit memory accesses.<br>2: The trigger will only match against 16-bit memory accesses or execution of 16-bit instructions.<br>3: The trigger will only match against 32-bit memory accesses or execution of 32-bit instructions.<br>4: The trigger will only match against execution of 48-bit instructions.<br>5: The trigger will only match against 64-bit memory accesses or execution of 64-bit instructions.<br>6: The trigger will only match against execution of 80-bit instructions.<br>7: The trigger will only match against execution of 96-bit instructions.<br>8: The trigger will only match against execution of 112-bit instructions.<br>9: The trigger will only match against 128-bit memory accesses or execution of 128-bit instructions.<br>An implementation must support the value of 0, but all other values are optional. When an implementation supports address triggers (select =0), it is recommended that those triggers support every access size that the hart supports, as well as for every instruction size that the hart supports.<br>Implementations such as RV32D or RV64V are able to perform loads and stores that are wider than XLEN. Custom extensions may also support instructions that are wider than XLEN. Because `tdata2` is of size XLEN, there is a known limitation that data value triggers (select =1) can only be supported for access sizes up to XLEN bits. When an implementation supports data value triggers (select =1), it is recommended that those triggers support every access size up to XLEN that the hart supports, as well as for every instruction length up to XLEN that the hart supports. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| action | The action to take when the trigger fires. The values are explained in Table 5.1. | WARL | 0 |
| chain | 0: When this trigger matches, the configured action is taken.<br>1: While this trigger does not match, it prevents the trigger with the next index from matching.<br>A trigger chain starts on the first trigger with chain = 1 after a trigger with chain = 0, or simply on the first trigger if that has chain = 1. It ends on the first trigger after that which has chain = 0. 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.<br>Debuggers should not terminate a chain with a trigger with a different type. It is undefined when exactly such a chain fires.<br>Because chain affects the next trigger, hardware must zero it in writes to mcontrol that set dmode to 0 if the next trigger has dmode of 1. In addition hardware should ignore writes to mcontrol that set dmode to 1 if the previous trigger has both dmode of 0 and chain of 1. Debuggers must avoid the latter case by checking chain on the previous trigger if they're writing mcontrol.<br>Implementations that wish to limit the maximum length of a trigger chain (eg. to meet timing requirements) may do so by zeroing chain in writes to mcontrol that would make the chain too long. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| match | 0: Matches when any compare value equals tdata2.<br>1: Matches when the top $M$ bits of any compare value match the top $M$ bits of tdata2. $M$ is $\mathsf{XLEN} - 1$ minus the index of the least-significant bit containing 0 in tdata2. Debuggers should only write values to tdata2 such that $M+$maskmax $\geq \mathsf{XLEN}$ and $M > 0$ , otherwise it's undefined on what conditions the trigger will match.<br>2: Matches when any compare value is greater than (unsigned) or equal to tdata2.<br>3: Matches when any compare value is less than (unsigned) tdata2.<br>4: Matches when $\frac{\mathsf{XLEN}}{2}-1$:0 of any compare value equals $\frac{\mathsf{XLEN}}{2} - 1$:0 of tdata2 after $\frac{\mathsf{XLEN}}{2} - 1$:0 of the compare value is ANDed with $\mathsf{XLEN}-1$:$\frac{\mathsf{XLEN}}{2}$ of tdata2.<br>5: Matches when $\mathsf{XLEN} - 1$:$\frac{\mathsf{XLEN}}{2}$ of any compare value equals $\frac{\mathsf{XLEN}}{2} - 1$:0 of tdata2 after $\mathsf{XLEN} - 1$:$\frac{\mathsf{XLEN}}{2}$ of the compare value is ANDed with $\mathsf{XLEN} - 1$:$\frac{\mathsf{XLEN}}{2}$ of tdata2.<br>8: Matches when match $= 0$ would not match.<br>9: Matches when match $= 1$ would not match.<br>12: Matches when match $= 4$ would not match.<br>13: Matches when match $= 5$ would not match.<br>Other values are reserved for future use.<br>All comparisons only look at the lower XLEN (in the current mode) bits of the compare values and of tdata2. When select $=1$ and access size is N, this is further reduced, and comparisons only look at the lower N bits of the compare values and of tdata2. | WARL | 0 |
| m | When set, enable this trigger in M-mode. | WARL | 0 |
| s | When set, enable this trigger in S/HS-mode. This bit is hard-wired to 0 if the hart does not support S-mode. | WARL | 0 |
| u | When set, enable this trigger in U-mode. This bit is hard-wired to 0 if the hart does not support U-mode. | WARL | 0 |
| execute | When set, the trigger fires on the virtual address or opcode of an instruction that is executed. | WARL | 0 |
| store | When set, the trigger fires on the virtual address or data of any store. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| load | When set, the trigger fires on the virtual address or data of any load. | WARL | 0 |

### 5.5.12 Match Control Type 6 (`mcontrol6`, at 0x7a1)

This register is accessible as `tdata1` when `type` is 6.

This replaces mcontrol in newer implementations and serves to provide additional functionality.

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 5.10 suggests timings for the best user experience.

Таблица 5.10: Suggested Trigger 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 |

A chain of triggers that don't all have the same `timing` value will never fire. That means to implement the suggestions in Table 5.10, both timings should be supported on load address triggers.

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.

In implementations that support `match` mode 1 (NAPOT), not all NAPOT ranges may be supported. All NAPOT ranges between $2^1$ and $2^{maskmax6}$ are supported where maskmax6 $\geq 1$. The value of maskmax6 can be determined by the debugger via the following sequence:

1. Set `match` =1.
2. Read `match`. If it is not 1 then NAPOT matching is not supported.
3. Write all ones to `tdata2`.
4. Read `tdata2`. The value of maskmax6 is the index of the most significant 0 bit plus 1.

If the A extension is supported, then trigger behavior is as follows for the load and store bits:

1. `lr` instructions are loads
2. successful `sc` instructions are stores
3. it is UNSPECIFIED whether failing `sc` instructions are stores or not
4. Each AMO instruction is a load for the read portion of the operation. The address is always available to trigger on, although the value loaded might not be, depending on the hardware implementation.
5. Each AMO instruction is a store for the write portion of the operation. The address is always available to trigger on, although the value stored might not be, depending on the hardware implementation.

If the destination register of any load or AMO is `zero` then it is UNSPECIFIED whether a load trigger with select =1 will match. Whether store triggers with select =1 match on AMOs is UNSPECIFIED.

This CSR is read/write.

| XLEN-1 XLEN-4 | XLEN-5 | XLEN-6 25 | 24 | 23 | 22 | 21 |
|---|---|---|---|---|---|---|
| type | dmode | 0 | vs | vu | hit | select |
| 4 | 1 | XLEN - 30 | 1 | 1 | 1 | 1 |

| 20 | 19 16 | 15 12 | 11 | 10 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| timing | size | action | chain | match | m | 0 | s | u | execute | store | load |
| 1 | 4 | 4 | 1 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Field | Description | Access | Reset |
|---|---|---|---|
| vs | When set, enable this trigger in VS-mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| vu | When set, enable this trigger in VU-mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| hit | If this 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. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| select | This bit determines the contents of the XLEN-bit compare values.<br>0: There is at least one compare value and it contains the lowest virtual address of the access. In addition, it is recommended that there are additional compare values for 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: There is exactly one compare value and it contains the data value loaded or stored, or the instruction executed. Any bits beyond the size of the data access will contain 0. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| timing | 0: The action for this trigger will be taken just before the instruction that triggered it is committed, but after all preceding instructions are committed. `xepc` or dpc (depending on action) must be set to the virtual address of the instruction that matched.<br>If this is combined with load and select =1 then a memory access will be performed (including any side effects of performing such an access) even though the load will not update its destination register. Debuggers should consider this when setting such breakpoints on, for example, memory-mapped I/O addresses.<br>1: The action for this trigger will be taken after the instruction that triggered it is committed. It should be taken before the next instruction is committed, but it is better to implement triggers imprecisely than to not implement them at all. `xepc` or dpc (depending on action) must be set to the virtual address of the next instruction that must be executed to preserve the program flow.<br>Most hardware will only implement one timing or the other, possibly dependent on select, execute, load, and store. This bit primarily exists for the hardware to communicate to the debugger what will happen. Hardware may implement the bit fully writable, in which case the debugger has a little more control.<br>Data load triggers with timing 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 timing of 1.<br>If a trigger with timing of 0 matches, it is implementation-dependent whether that prevents a trigger with timing of 1 matching as well. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| size | 0: The trigger will attempt to match against an access of any size. The behavior is only well-defined if select $= 0$, or if the access size is XLEN.<br>1: The trigger will only match against 8-bit memory accesses.<br>2: The trigger will only match against 16-bit memory accesses or execution of 16-bit instructions.<br>3: The trigger will only match against 32-bit memory accesses or execution of 32-bit instructions.<br>4: The trigger will only match against execution of 48-bit instructions.<br>5: The trigger will only match against 64-bit memory accesses or execution of 64-bit instructions.<br>6: The trigger will only match against execution of 80-bit instructions.<br>7: The trigger will only match against execution of 96-bit instructions.<br>8: The trigger will only match against execution of 112-bit instructions.<br>9: The trigger will only match against 128-bit memory accesses or execution of 128-bit instructions.<br>An implementation must support the value of 0, but all other values are optional. When an implementation supports address triggers (select $=0$), it is recommended that those triggers support every access size that the hart supports, as well as for every instruction size that the hart supports.<br>Implementations such as RV32D or RV64V are able to perform loads and stores that are wider than XLEN. Custom extensions may also support instructions that are wider than XLEN. Because tdata2 is of size XLEN, there is a known limitation that data value triggers (select $=1$) can only be supported for access sizes up to XLEN bits. When an implementation supports data value triggers (select $=1$), it is recommended that those triggers support every access size up to XLEN that the hart supports, as well as for every instruction length up to XLEN that the hart supports. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| action | The action to take when the trigger fires. The values are explained in Table 5.1. | WARL | 0 |
| chain | 0: When this trigger matches, the configured action is taken.<br>1: While this trigger does not match, it prevents the trigger with the next index from matching.<br>A trigger chain starts on the first trigger with chain $= 1$ after a trigger with chain $= 0$, or simply on the first trigger if that has chain $= 1$. It ends on the first trigger after that which has chain $= 0$. 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.<br>Debuggers should not terminate a chain with a trigger with a different type. It is undefined when exactly such a chain fires.<br>Because chain affects the next trigger, hardware must zero it in writes to `mcontrol6` that set dmode to 0 if the next trigger has dmode of 1. In addition hardware should ignore writes to `mcontrol6` that set dmode to 1 if the previous trigger has both dmode of 0 and chain of 1. Debuggers must avoid the latter case by checking chain on the previous trigger if they're writing `mcontrol6`.<br>Implementations that wish to limit the maximum length of a trigger chain (eg. to meet timing requirements) may do so by zeroing chain in writes to `mcontrol6` that would make the chain too long. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| match | 0: Matches when any compare value equals tdata2.<br>1: Matches when the top $M$ bits of any compare value match the top $M$ bits of tdata2. $M$ is $\mathsf{XLEN} - 1$ minus the index of the least-significant bit containing 0 in tdata2. tdata2 is WARL and bit $\mathsf{maskmax6} - 1$ will be set to 0 if no less significant bits are written with 0. Legal values for tdata2 require $M + \mathsf{maskmax6} \geq \mathsf{XLEN}$ and $M > 0$. See above for how to determine maskmax6.<br>2: Matches when any compare value is greater than (unsigned) or equal to tdata2.<br>3: Matches when any compare value is less than (unsigned) tdata2.<br>4: Matches when $\frac{\mathsf{XLEN}}{2} - 1{:}0$ of any compare value equals $\frac{\mathsf{XLEN}}{2} - 1{:}0$ of tdata2 after $\frac{\mathsf{XLEN}}{2} - 1{:}0$ of the compare value is ANDed with $\mathsf{XLEN} - 1{:}\frac{\mathsf{XLEN}}{2}$ of tdata2.<br>5: Matches when $\mathsf{XLEN} - 1{:}\frac{\mathsf{XLEN}}{2}$ of any compare value equals $\frac{\mathsf{XLEN}}{2} - 1{:}0$ of tdata2 after $\mathsf{XLEN} - 1{:}\frac{\mathsf{XLEN}}{2}$ of the compare value is ANDed with $\mathsf{XLEN} - 1{:}\frac{\mathsf{XLEN}}{2}$ of tdata2.<br>8: Matches when match $= 0$ would not match.<br>9: Matches when match $= 1$ would not match.<br>12: Matches when match $= 4$ would not match.<br>13: Matches when match $= 5$ would not match.<br>Other values are reserved for future use.<br>All comparisons only look at the lower XLEN (in the current mode) bits of the compare values and of tdata2. When select $=1$ and access size is N, this is further reduced, and comparisons only look at the lower N bits of the compare values and of tdata2. | WARL | 0 |
| m | When set, enable this trigger in M-mode. | WARL | 0 |
| s | When set, enable this trigger in S/HS-mode. This bit is hard-wired to 0 if the hart does not support S-mode. | WARL | 0 |
| u | When set, enable this trigger in U-mode. This bit is hard-wired to 0 if the hart does not support U-mode. | WARL | 0 |
| execute | When set, the trigger fires on the virtual address or opcode of an instruction that is executed. | WARL | 0 |
| store | When set, the trigger fires on the virtual address or data of any store. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| load | When set, the trigger fires on the virtual address or data of any load. | WARL | 0 |

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

This register is accessible as tdata1 when type is 3.

If count is not 0, then every instruction completed or trap taken from a mode where the trigger is enabled decrements count by 1. When count is decremented from 1 to 0 then pending becomes set. When pending is set, the trigger fires just before any further instructions are executed in a mode where the trigger is enabled. As the trigger fires, pending is cleared.

*The intent of this design is to cleanly handle the case where action is 0, m is 0, u is 1, count is 1, and the U-mode instruction being executed causes a trap into M-mode. In that case we want the entire M-mode handler to be executed, and the debug trap to be taken before the next U-mode instruction.*

*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.*

This CSR is read/write.

| XLEN-1 — XLEN-4 | XLEN-5 | XLEN-6 — 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|
| type | dmode | 0 | vs | vu | hit |
| 4 | 1 | XLEN - 32 | 1 | 1 | 1 |

| 23 — 10 | 9 | 8 | 7 | 6 | 5 — 0 |
|---|---|---|---|---|---|
| count | m | pending | s | u | action |
| 14 | 1 | 1 | 1 | 1 | 6 |

| Field | Description | Access | Reset |
|---|---|---|---|
| vs | When set, enable this trigger in VS-mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| vu | When set, enable this trigger in VU-mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| hit | If this 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. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| count | When count is decremented to 0, the trigger fires. Instead of changing count from 1 to 0, it is also acceptable for hardware to clear m, s, u, vs, and vu. This allows count to be hard-wired to 1 if this register just exists for single step. | WARL | 1 |
| m | When set, enable this trigger in M-mode. | WARL | 0 |
| pending | This bit becomes set when count is decremented from 1 to 0. It is cleared when the trigger fires. | R/W | 0 |
| s | When set, enable this trigger in S/HS-mode. This bit is hard-wired to 0 if the hart does not support S-mode. | WARL | 0 |
| u | When set, enable this trigger in U-mode. This bit is hard-wired to 0 if the hart does not support U-mode. | WARL | 0 |
| action | The action to take when the trigger fires. The values are explained in Table 5.1. | WARL | 0 |

### 5.5.14 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 trap handler is executed.

This CSR is read/write.

| XLEN-1 … XLEN-4 | XLEN-5 | XLEN-6 | XLEN-7 … 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 … 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| type | dmode | hit | 0 | vs | vu | 0 | m | 0 | s | u | action |
| 4 | 1 | 1 | XLEN - 19 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 |

| Field | Description | Access | Reset |
|---|---|---|---|
| hit | If this 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. | WARL | 0 |
| vs | When set, enable this trigger for interrupts that are taken from VS mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| vu | When set, enable this trigger for interrupts that are taken from VU mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| m | When set, enable this trigger for interrupts that are taken from M mode. | WARL | 0 |
| s | When set, enable this trigger for interrupts that are taken from S/HS mode. This bit is hard-wired to 0 if the hart does not support S-mode. | WARL | 0 |
| u | When set, enable this trigger for interrupts that are taken from U mode. This bit is hard-wired to 0 if the hart does not support U-mode. | WARL | 0 |
| action | The action to take when the trigger fires. The values are explained in Table 5.1. | WARL | 0 |

### 5.5.15 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 trap handler is executed.

This CSR is read/write.

| XLEN-1 XLEN-4 | XLEN-5 | XLEN-6 | XLEN-7 13 | 12 | 11 |
|---|---|---|---|---|---|
| type | dmode | hit | 0 | vs | vu |
| 4 | 1 | 1 | XLEN - 19 | 1 | 1 |

| 10 | 9 | 8 | 7 | 6 | 5 0 |
|---|---|---|---|---|---|
| nmi | m | 0 | s | u | action |
| 1 | 1 | 1 | 1 | 1 | 6 |

| Field | Description | Access | Reset |
|---|---|---|---|
| hit | If this 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. | WARL | 0 |
| vs | When set, enable this trigger for exceptions that are taken from VS mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| vu | When set, enable this trigger for exceptions that are taken from VU mode. This bit is hard-wired to 0 if the hart does not support virtualization mode. | WARL | 0 |
| nmi | When set, non-maskable interrupts cause this trigger to fire, regardless of the values of m, s, u, vs, and vu. | WARL | 0 |
| m | When set, enable this trigger for exceptions that are taken from M mode. | WARL | 0 |
| s | When set, enable this trigger for exceptions that are taken from S/HS mode. This bit is hard-wired to 0 if the hart does not support S-mode. | WARL | 0 |
| u | When set, enable this trigger for exceptions that are taken from U mode. This bit is hard-wired to 0 if the hart does not support U-mode. | WARL | 0 |
| action | The action to take when the trigger fires. The values are explained in Table 5.1. | WARL | 0 |

### 5.5.16 External Trigger (`tmexttrigger`, at 0x7a1)

This register is accessible as `tdata1` when `type` is 7.

This trigger fires when any selected TM external trigger input signals. Up to 16 TM external trigger inputs coming from other blocks outside the TM, (e.g. signaling an hpmcounter overflow) can be selected. Hardware may support none or just a few TM external trigger inputs (starting with TM external trigger input 0 and continuing sequentially). Unsupported inputs are hardwired to be inactive.

This CSR is read/write.

| XLEN-1 … XLEN-4 | XLEN-5 | XLEN-6 | XLEN-7 … 23 | 22 | 21 … 6 | 5 … 0 |
|---|---|---|---|---|---|---|
| type | dmode | hit | 0 | intctl | select | action |
| 4 | 1 | 1 | XLEN - 29 | 1 | 16 | 6 |

| Field | Description | Access | Reset |
|---|---|---|---|
| hit | If this 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. | WARL | 0 |
| intctl | This optional bit, when set, causes this trigger to fire whenever an attached interrupt controller signals a trigger. | WARL | 0 |
| select | Selects any combination of up to 16 external debug trigger inputs that cause this trigger to fire. | WARL | 0 |
| action | The action to take when the trigger fires. The values are explained in Table 5.1. | WARL | 0 |

### 5.5.17 Trigger Extra (RV32) (`textra32`, at 0x7a3)

This register is accessible as `tdata3` when type is 2, 3, 4, 5, or 6 and XLEN=32.

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).

Byte-granular comparison of `scontext` to svalue allows `scontext` to be defined to include more than one element of comparison. For example, software instrumentation can program the `scontext` value to be the concatenation of different ID contexts such as process ID and thread ID. The user can then program byte compares based on sbytemask to include one or more of the contexts in the compare.

Byte masking only applies to `scontext` comparison; i.e when sselect is 1.

*Note that sselect and mhselect filtering apply in all modes, including M-mode and S-mode. If desired, debuggers can use a trigger's mode filtering bits to restrict the matching to modes where it considers ASID/VMID/scontext/hcontext to be active.*

This CSR is read/write.

| 31–26 | 25–23 | 22–20 | 19–18 | 17–2 | 1–0 |
|---|---|---|---|---|---|
| mhvalue | mhselect | 0 | sbytemask | svalue | sselect |
| 6 | 3 | 3 | 2 | 16 | 2 |

| Field | Description | Access | Reset |
|---|---|---|---|
| mhvalue | Data used together with mhselect. | WARL | 0 |

*Continued on next page*

| Field | Description | Access | Reset |
|---|---|---|---|
| mhselect | 0: Ignore mhvalue.<br>4: This trigger will only match if the low bits of `mcontext`/`hcontext` equal mhvalue.<br>1, 5: This trigger will only match if the low bits of `mcontext`/`hcontext` equal {mhvalue, mhselect[2]}.<br>2, 6: This trigger will only match if VMID in hgatp equals the lower VMIDMAX (defined in the Privileged Spec) bits of {mhvalue, mhselect[2]}.<br>3, 7: Reserved.<br>If the H extension is not supported, the only legal values are 0 and 4. | WARL | 0 |
| sbytemask | When the least significant bit of this field is 1, it causes bits 7:0 in the comparison to be ignored, when sselect =1. When the next most significant bit of this field is 1, it causes bits 15:8 to be ignored in the comparison, when sselect =1. | WARL | 0 |
| svalue | Data used together with sselect.<br>This field should be tied to 0 when S-mode is not supported. | WARL | 0 |
| sselect | 0: Ignore svalue.<br>1: This trigger will only match if the low bits of `scontext` equal svalue.<br>2: This trigger will only match if:<br>• the mode is VS-mode or VU-mode and ASID in `vsatp` equals the lower ASIDMAX (defined in the Privileged Spec) bits of svalue.<br>• in all other modes, ASID in `satp` equals the lower ASIDMAX (defined in the Privileged Spec) bits of svalue.<br>This field should be tied to 0 when S-mode is not supported. | WARL | 0 |

### 5.5.18 Trigger Extra (RV64) (`textra64`, at 0x7a3)

This register is accessible as `tdata3` when type is 2, 3, 4, 5, or 6 and XLEN=64. The fields are defined above, in `textra32`.

Byte-granular comparison of `scontext` to svalue in `textra64` allows `scontext` to be defined to include more than one element of comparison. For example, software instrumentation can program the `scontext` value to be the concatenation of different ID contexts such as process ID and thread ID. The user can then program byte compares based on sbytemask to include one or more of the contexts in the compare.

Byte masking only applies to `scontext` comparison; i.e when sselect is 1.

This CSR is read/write.

| 63–51 | 50–48 | 47–41 | 40–36 | 35–2 | 1–0 |
|---|---|---|---|---|---|
| mhvalue | mhselect | 0 | sbytemask | svalue | sselect |
| 13 | 3 | 7 | 5 | 34 | 2 |

| Field | Description | Access | Reset |
|---|---|---|---|
| sbytemask | When the least significant bit of this field is 1, it causes bits 7:0 in the comparison to be ignored, when sselect =1. Likewise, the second bit controls the comparison of bits 15:8, third bit controls the comparison of bits 23:16, fourth bit controls the comparison of bits 31:24, and fifth bit controls the comparison of bits 33:32. | WARL | 0 |

# Глава 6

# Debug Transport Module (DTM), non-ISA

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 hardware platform. Ideally every component that communicates with the outside world includes a DTM, allowing a hardware platform to be debugged through every transport it supports. For instance a USB component could include a DTM. This would trivially allow any hardware 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 1.0.0-STABLE, with custom DTM." If the JTAG DTM described here is implemented, it must be advertised as conforming to the "RISC-V Debug Specification 1.0.0-STABLE, 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 component's 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.

Таблица 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 (`dtmcs`) | For Debugging | 90 |
| 0x11 | Debug Module Interface Access (`dmi`) | For Debugging | 91 |
| 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 | ManufId | 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 |
| ManufId | 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.

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

| Field | Description | Access | Reset |
|---|---|---|---|
| dmihardreset | Writing 1 to this bit does a hard reset of the DTM, causing the DTM to forget about any outstanding DMI transactions, and returning all registers and internal state to their reset value. 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, but does not affect outstanding DMI transactions. | 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 (dmistat of 3). A debugger must still check dmistat 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 op of 2).<br>3: An operation was attempted while a DMI access was still in progress (resulted in op of 3). | R | 0 |
| abits | The size of address in dmi. | R | Preset |
| version | 0: Version described in spec version 0.11.<br>1: Version described in spec versions 0.13 and 1.0.<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 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.*

| abits+33 … 34 | 33 … 2 | 1 … 0 |
|---|---|---|
| address | data | op |
| abits | 32 | 2 |

| Field | Description | Access | Reset |
|---|---|---|---|
| address | Address used for DMI access. In Update-DR this value is used to access the DM over the DMI. | R/W | 0 |
| data | 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 | When the debugger writes this field, it has the following meaning:<br>0: Ignore data and address. (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.<br>1: Read from address. (read)<br>2: Write data to address. (write)<br>3: Reserved.<br>When the debugger reads this field, it means the following:<br>0: The previous operation completed successfully.<br>1: Reserved.<br>2: A previous operation failed. The data scanned into dmi in this access will be ignored. This status is sticky and can be cleared by writing dmireset in dtmcs.<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 dmi in this access will be ignored. This status is sticky and can be cleared by writing dmireset in dtmcs. 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. | 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.

Таблица 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 hardware platform requires nTRST then it is permissible to reuse the nRESET pin as the nTRST signal. If a hardware platform requires both hardware platform 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.

Таблица 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 | TRIGIN |
| GND | 19 | 20 | TRIGOUT |

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

Таблица 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>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 by this specification, to be driven by debug adapter. (Can be used for extended functions like UART or boot mode selection by some debug adapters). |
| 20 | TRIGOUT | Not used by this specification, driven by the target. |

# Приложение 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 and is only accessible to the harts in Debug Mode. When taking this trap, `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 `nop`s. 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 trap address within the Debug Module. The code there causes the hart to write to the Debug Module indicating an exception. Then the hart jumps back to the park loop. The DM infers from the write that there was an exception, and sets `cmderr` appropriately. Typically the hart will execute a `fence` instruction before entering the park loop, to ensure that any effects from the abstract command, such as a write to `data0`, take effect before the DM returns `busy` to 0.

To resume execution, the debug module sets a flag which causes the hart to execute a `dret`. `dret` is an instruction that only has meaning while in Debug Mode and not executing from the Program Buffer. Its recommended encoding is 0x7b200073. 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.

Note that for debug to be possible, the PMP must not disallow fetches, loads, or stores in the address range associated with the Debug Module when the hart is in Debug Mode.

## A.3 Debug Module Interface Signals

As stated in section 3.1 the details of the DMI are left to the system designer. It is quite often the case that only one DTM and one DM is implemented. In this case it might be useful to comply with the signals suggested in table A.1, which is the implementation used in the open-source rocket-chip RISC-V core.

The DTM can start a request when the DM sets REQ_READY to 1. When this is the case REQ_OP can be set to 1 for a read or 2 for a write request. The desired address is driven with the REQ_ADDRESS signal. Finally REQ_VALID is set high, indicating to the DM that a valid request is pending.

The DM must respond to a request from the DTM when RSP_READY is high. The status of the response is indicated by the RSP_OP signal (see `op`). The data of the response is driven to RSP_DATA. A pending response is signalled by setting RSP_VALID.

| Signal | Width | Source | Description |
|---|---|---|---|
| REQ_VALID | 1 | DTM | Indicates that a valid request is pending |
| REQ_READY | 1 | DM | Indicates that the DM is able to process a request |
| REQ_ADDRESS | abits | DTM | Requested address |
| REQ_DATA | 32 | DTM | Requested data |
| REQ_OP | 2 | DTM | Same meaning as the op field |
| RSP_VALID | 1 | DM | Indicates that a valid respond is pending |
| RSP_READY | 1 | DTM | Indicates that the DTM is able to process a respond |
| RSP_DATA | 32 | DM | Response data |
| RSP_OP | 2 | DM | Same meaning as the op field |

Таблица A.1: Signals for the suggested DMI between one DTM and one DM

# Приложение 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 | `command` | aarsize = 2, transfer, regno = 0x1008 | Read `s0` |
| Read | `data0` | - | Returns value that was in `s0` |

Write `mstatus` using abstract command:

| Op | Address | Value | Comment |
|---|---|---|---|
| Write | data0 | new value | |
| Write | command | aarsize = 2, transfer, write, regno = 0x300 | Write `mstatus` |

### 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 | progbuf0 | `csrw s0, MSTATUS` | |
| Write | progbuf1 | `ebreak` | |
| Write | data0 | new value | |
| Write | command | aarsize = 2, postexec, transfer, write, regno = 0x1008 | Write `s0`, then execute program buffer |

Read `f1` using program buffer:

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

## 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 | sbcs | sbaccess = 2, sbreadonaddr | Setup |
| Write | sbaddress0 | address | |
| Read | sbdata0 | - | Value read from memory |

Read block of memory using system bus access:

| Op | Address | Value | Comment |
|---|---|---|---|
| Write | sbcs | sbaccess = 2, sbreadonaddr, sbreadondata, sbautoincrement | Turn on autoread and autoincrement |
| Write | sbaddress0 | address | Writing address triggers read and increment |
| Read | sbdata0 | - | Value read from memory |
| Read | sbdata0 | - | Next value read from memory |
| ... | ... | ... | ... |
| Write | sbcs | 0 | Disable autoread |
| Read | sbdata0 | - | 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 | progbuf0 | `lw s0, 0(s0)` | |
| Write | progbuf1 | `ebreak` | |
| Write | data0 | address | |
| Write | command | write, postexec, regno = 0x1008 | Write `s0`, then execute program buffer |
| Write | command | regno = 0x1008 | Read `s0` |
| Read | data0 | - | Value read from memory |

Read block of memory using program buffer:

| Op | Address | Value | Comment |
|---|---|---|---|
| Write | progbuf0 | lw s1, 0(s0) | |
| Write | progbuf1 | addi s0, s0, 4 | |
| Write | progbuf2 | ebreak | |
| Write | data0 | address | |
| Write | command | write, postexec, regno = 0x1008 | Write s0, then execute program buffer |
| Write | command | postexec, regno = 0x1009 | Read s1, then execute program buffer |
| Write | abstractauto | autoexecdata [0] | Set autoexecdata [0] |
| Read | data0 | - | Get value read from memory, then execute program buffer |
| Read | data0 | - | Get next value read from memory, then execute program buffer |
| ... | ... | ... | ... |
| Write | abstractauto | 0 | Clear autoexecdata [0] |
| Read | data0 | - | 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 | data1 | address | |
| Write | command | cmdtype=2, aamsize =2 | |
| Read | data0 | - | Value read from memory |

Read block of memory using abstract memory access:

| Op | Address | Value | Comment |
|---|---|---|---|
| Write | abstractauto | 1 | Re-execute the command when data0 is accessed |
| Write | data1 | address | |
| Write | command | cmdtype=2, aamsize =2, aampostincrement =1 | |
| Read | data0 | - | Read value, and trigger reading of next address |
| ... | ... | ... | ... |
| Write | abstractauto | 0 | Disable auto-exec |
| Read | data0 | - | 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 | sbcs | sbaccess = 2 | Configure access size |
| Write | sbaddress0 | address | |
| Write | sbdata0 | value | |

Write a block of memory using system bus access:

| Op | Address | Value | Comment |
|---|---|---|---|
| Write | sbcs | sbaccess = 2, sbautoincrement | Turn on autoincrement |
| Write | sbaddress0 | address | |
| Write | sbdata0 | value0 | |
| Write | sbdata0 | value1 | |
| ... | ... | ... | ... |
| Write | sbdata0 | 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 | progbuf0 | `sw s1, 0(s0)` | |
| Write | progbuf1 | `ebreak` | |
| Write | data0 | address | |
| Write | command | write, regno = 0x1008 | Write `s0` |
| Write | data0 | value | |
| Write | command | write, postexec, regno = 0x1009 | Write `s1`, then execute program buffer |

Write block of memory using program buffer:

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

### 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 | data1 | address | |
| Write | data0 | value | |
| Write | command | cmdtype=2, aamsize =2, write=1 | |

Write a block of memory using abstract memory access:

| Op | Address | Value | Comment |
|---|---|---|---|
| Write | data1 | address | |
| Write | data0 | value0 | |
| Write | command | cmdtype=2, aamsize =2, write=1, aampostincrement =1 | |
| Write | abstractauto | 1 | Re-execute the command when data0 is accessed |
| Write | data0 | value1 | |
| Write | data0 | value2 | |
| ... | ... | ... | ... |
| Write | data0 | valueN | |
| Write | abstractauto | 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 might 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:

| | | |
|---|---|---|
| `tdata1` | 0x105c | action=1, match=0, m=1, s=1, u=1, execute=1 |
| `tdata2` | 0x80001234 | address |

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

| | | |
|---|---|---|
| `tdata1` | 0x4159 | timing=1, action=1, match=0, m=1, s=1, u=1, load=1 |
| `tdata2` | 0x80007f80 | address |

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

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

Enter Debug Mode right before a write to an address between 0x81230000 and 0x8123ffff (inclusive):

| | | |
|---|---|---|
| `tdata1` | 0x10da | action=1, match=1, m=1, s=1, u=1, store=1 |
| `tdata2` | 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):

| | | |
|---|---|---|
| `tdata1` 0 | 0x41a59 | timing=1, action=1, chain=1, match=4, m=1, s=1, u=1, load=1 |
| `tdata2` 0 | 0xfff03090 | Mask for low half, then match for low half |
| `tdata1` 1 | 0x412d9 | timing=1, action=1, match=5, m=1, s=1, u=1, load=1 |
| `tdata2` 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 hardware 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 | `progbuf0` | `transfer arg2, s0` | Save `s0` |
| Write | `progbuf1` | `transfer s0, arg0` | Read first argument (address) |
| Write | `progbuf2` | `transfer arg0, s1` | Save `s1` |
| Write | `progbuf3` | `transfer s1, arg1` | Read second argument (data) |
| Write | `progbuf4` | `sw s1, 0(s0)` | |
| Write | `progbuf5` | `transfer s1, arg0` | Restore `s1` |
| Write | `progbuf6` | `transfer s0, arg2` | Restore `s0` |
| Write | `progbuf7` | `ebreak` | |
| Write | `data0` | address | |
| Write | `data1` | data | |
| Write | `command` | 0x10000000 | 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 | `progbuf0` | `transfer arg0, s0` | Save `s0` |
| Write | `progbuf1` | `li s0, (1 << 6)` | Form the mask for `m` bit |
| Write | `progbuf2` | `csrrs x0, tdata1, s0` | Apply the mask to `mcontrol` |
| Write | `progbuf3` | `transfer s0, arg2` | Restore `s0` |
| Write | `progbuf4` | `ebreak` | |
| Write | `command` | 0x10000000 | Perform quick access |

# Предметный указатель

# Приложение C

# История изменений

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 2794e83 | 2021-10-25 | Ivan Kuzmenko | Исправлена ссылка на Trigger Module |
| 603d0af | 2021-10-25 | Ivan Kuzmenko | Попытка исправить очередной конфликт слияния |
| 42120c1 | 2021-10-12 | Ivan Kuzmenko | Исправил опечатку |
| 667f365 | 2021-10-12 | Ivan Kuzmenko | Перевёл главу 2 и иллюстрацию |
| ff8a6fa | 2021-10-07 | Paul Donahue | Clarify that etrigger.nmi ignores vs and vu (not just m, s, u) (#676) |
| 543e71b | 2021-10-07 | NewPirateOfUASeas | fix |
| 7e83f35 | 2021-10-07 | NewPirateOfUASeas | overview sortof translated |
| 4d40157 | 2021-10-07 | Ivan Kuzmenko | GH Actions |
| 86bebf7 | 2021-10-07 | Ivan Kuzmenko | Исправил копируемость текста и некоторые опечатки |
| 6bb504f | 2021-10-05 | Ivan Kuzmenko | ой |
| 41f4563 | 2021-10-05 | Ivan Kuzmenko | Перевод первой главы на русский |
| 55c492d | 2021-09-17 | Paul Donahue | Clarify ASID filtering (#667) |
| 031ff77 | 2021-09-15 | Tim Newsome | SBA write memory example: configure size (#673) |
| 80563e2 | 2021-09-14 | Tim Newsome | Run ‘apt update‘ (#672) |
| fc52dc5 | 2021-09-14 | Tim Newsome | Don’t call read/write CSRs read-only. (#668) |
| 57c1233 | 2021-09-13 | Tim Newsome | Rebuild PDF. |
| 6e127fc | 2021-08-16 | Paul Donahue | Native triggers and reentrancy (#660) |
| 6a9b355 | 2021-08-13 | Tim Newsome | Clarify unused debug connector pins names/uses. (#664) |
| f77291e | 2021-08-12 | Tim Newsome | Rebuild PDF. |
| 33cd3a6 | 2021-08-10 | Paul Donahue | Clarify dcsr.cause=2 (#663) |
| 8f7873d | 2021-07-28 | Paul Donahue | Clarify dpc for non-mcontrol triggers (#624) |
| 70e3db2 | 2021-07-13 | Daniel Mangum | Update link to mailing list in README.md (#658) |
| 63c985f | 2021-07-12 | Tim Newsome | Rebuild PDF. |
| 9ac6506 | 2021-07-08 | Paul Donahue | xepc instead of exception PC (#654) |
| e6b7b6c | 2021-07-08 | Tim Newsome | Clarify postincrement/transfer behavior. (#655) |
| f393a8d | 2021-07-08 | Daniel Mangum | Fix small grammatical error in suggested DMI signals (#656) |

| | | | |
|---|---|---|---|
| d584f0f | 2021-07-08 | Tim Newsome | Attempt to build the document using github actions. (#657) |
| 022d62f | 2021-06-29 | Paul Donahue | Clarify hstatus when the H extension isn't implemented (#646) |
| c391ffe | 2021-06-16 | Paul Donahue | MPRV is in mstatus, not mcontrol. (#651) |
| a1e05fb | 2021-06-14 | Paul Donahue | Clarify what happens to halted harts upon DM reset (#648) |
| 0f4ea2f | 2021-06-10 | Tim Newsome | Rebuild PDF. |
| 641cd87 | 2021-06-09 | Tim Newsome | Unselected harts may change groups when hgwrite=1 (#642) |
| 55333c4 | 2021-06-09 | Tim Newsome | hasel etc. are only ignored by abstract commands. (#643) |
| 3704486 | 2021-06-09 | Tim Newsome | Mention license in debug_defines.h. (#641) |
| b68c265 | 2021-06-08 | Paul Donahue | Single stepping an instruction that fires a trigger (#622) |
| fdecf04 | 2021-06-08 | Tim Newsome | Labels must come after (or in) their caption. (#640) |
| 5350603 | 2021-06-04 | Paul Donahue | Traps caused by action=0 triggers can be delegated via medeleg (#637) |
| 11bf0db | 2021-05-28 | Tim Newsome | Divide the spec into ISA and non-ISA. (#635) |
| 3334910 | 2021-05-18 | Tim Newsome | Further diagram update. (#632) |
| b344c8f | 2021-05-14 | Tim Newsome | External debuggers should set dmode. (#634) |
| d9434bc | 2021-05-14 | Paul Donahue | Fix #630 (#633) |
| 39fab4b | 2021-05-10 | Tim Newsome | Rebuild PDF. |
| 090eac8 | 2021-05-04 | Tim Newsome | Update Run/Halt Debug State Machine. (#629) |
| 31a5f61 | 2021-04-08 | Tim Newsome | Rebuild PDF. |
| e3e408c | 2021-03-25 | Paul Donahue | Clarify triggers (#628) |
| e0e0b4d | 2021-03-23 | Tim Newsome | Remove latest draft link. (#627) |
| 1b665f3 | 2021-03-22 | Paul Donahue | clarify abstractauto (#625) |
| 7242fe1 | 2021-03-17 | Paul Donahue | Clarify DM behavior for non-existent registers (#621) |
| 9ae7560 | 2021-03-10 | Paul Donahue | Fix minor typos (#620) |
| bd786dc | 2021-02-08 | Tim Newsome | Rebuild PDF. |
| 132ffb8 | 2021-02-02 | Thomas Wicki | Clarify MCONTROL/MCONTROL6 'timing' description (#614) |
| 075de11 | 2021-02-02 | Paul Donahue | Clarify aarpostincrement wrapping (#613) |
| fb15173 | 2021-01-25 | Paul Donahue | hartsel and hasel are WARL (#612) |
| fcc2a33 | 2021-01-11 | Tim Newsome | Rebuild PDF. |
| 1b9caa1 | 2021-01-11 | Tim Newsome | Define some more terms/acronyms. (#610) |
| 822f63b | 2021-01-05 | Tim Newsome | Move dret completely into the appendix. (#611) |
| 57e271d | 2020-12-30 | Tim Newsome | Use A-mode instead of "A mode"consistently. (#609) |
| 08e072a | 2020-12-29 | Ernie Edgar | update version of priv spec to one with hypervisor (#608) |
| bb578a4 | 2020-12-28 | Tim Newsome | Clarify details around fence and progbuf. (#601) |
| 7e47254 | 2020-12-28 | Tim Newsome | Rebuild PDF. |
| 90ba168 | 2020-12-21 | Tim Newsome | Clarify how many bits mcontrol/mcontrol6 compare. (#604) |
| 27d735c | 2020-12-18 | Tim Newsome | Clarify maskmax corner cases. (#607) |
| 2930c1d | 2020-12-18 | Tim Newsome | Clarify that ASID might come from satp or vsatp. (#606) |

| | | | |
|---|---|---|---|
| c98d7e6 | 2020-12-18 | Tim Newsome | Mark 1.0 as STABLE. (#605) |
| edd6482 | 2020-12-17 | Tim Newsome | Breakpoint in trap handler *might* be unrecoverable. (#603) |
| c49e9c3 | 2020-12-16 | Tim Newsome | Rebuild PDF. |
| 4772b19 | 2020-12-03 | Ernie Edgar | Add version 1.0 value to dtmcs.version field (#602) |
| 393d965 | 2020-11-23 | Tim Newsome | Document changes since 0.13. (#600) |
| f2ff7a6 | 2020-11-16 | Tim Newsome | Rebuild PDF. |
| 2d7190c | 2020-11-16 | Tim Newsome | Add clic bit to tmexttrigger. (#599) |
| 0198481 | 2020-11-16 | Tim Newsome | Explain how to simply write any trigger. (#598) |
| b35af12 | 2020-11-16 | Tim Newsome | Clarify trigger CSR behavior when XLEN changes. (#597) |
| f116aea | 2020-11-16 | benscotstaveley | add dmstatus.ndmresetpending to allow a debugger to determine when ndmreset is complete (#594) |
| 3dd952c | 2020-11-13 | Tim Newsome | Chains should all have the same type. (#596) |
| 34f80c6 | 2020-11-13 | Tim Newsome | AMO operations may be ignored by mcontrol triggers. (#595) |
| 9fea4c5 | 2020-11-13 | Tim Newsome | Add keepalive feature. (#592) |
| 53191a4 | 2020-11-12 | Bruce Ableidinger | Added sbytemask field to textra32 and textra64 (#588) |
| 26040fd | 2020-11-11 | Tim Newsome | hit must be set on fire, may be set on match (#593) |
| a2cf8fc | 2020-11-11 | Tim Newsome | Create LaTeX macros for fields without descriptions. (#591) |
| 2bde1e4 | 2020-11-10 | Tim Newsome | Change version number to 1.0. (#590) |
| 75a7607 | 2020-11-10 | Paul Donahue | Fix #587 (#589) |
| cabb06d | 2020-11-09 | Tim Newsome | Rebuild PDF. |
| c89895c | 2020-11-06 | Tim Newsome | Change quick access exceptions to halt the target. (#585) |
| 46804a6 | 2020-11-04 | Paul Donahue | Recommend mprven=1 (#580) |
| 219d105 | 2020-11-03 | Tim Newsome | Add pending state/bit to icount. (#574) |
| b860d53 | 2020-11-03 | Tim Newsome | RISC-V External Debug Support -> RISC-V Debug Support (#581) |
| fd654d4 | 2020-11-02 | Paul Donahue | textra32/64 also affects the new mcontrol6 triggers (#578) |
| 3d24926 | 2020-10-30 | Tim Newsome | List all DM registers in Table 3.8. (#579) |
| 971d0aa | 2020-10-30 | Paul Donahue | Add support for the A extension (#561) |
| d589bc3 | 2020-10-30 | Paul Donahue | Remove all uses of the ambiguous term "may not"(#576) |
| db8e814 | 2020-10-29 | Paul Donahue | Triggers affect harts, not the system. For instance, there may be (#575) |
| c877e9c | 2020-10-29 | Paul Donahue | Added exttrigger capability as type 7 (#543) |
| 0a81ec3 | 2020-10-28 | Tim Newsome | Debuggers should know when harts are unavailable. (#520) |
| 65af35f | 2020-10-26 | Ernie Edgar | Update debug_module.tex (#577) |
| 99dfc98 | 2020-10-23 | Scott Johnson | Don't decrement icount.count when exception is blocked by tcontrol.mte (#557) |
| 66481cf | 2020-10-16 | Paul Donahue | Fix broken reference (#567) |
| 0b42843 | 2020-10-16 | benscotstaveley | require polling of dmactive low as well as dmactive high transitions (#566) |

| | | | |
|---|---|---|---|
| 1e81d58 | 2020-10-16 | Paul Donahue | Use official RISC-V terminology (#564) |
| 5c65dfe | 2020-10-09 | Tim Newsome | Rebuild PDF. |
| 9c083f1 | 2020-10-05 | Ernie Edgar | Clarify that PMP must allow access to DM for debug to be possible (#554) |
| 7aa5978 | 2020-09-22 | Paul Donahue | Hypervisor support (#549) |
| 597281c | 2020-09-16 | Ernie Edgar | Update rocket-chip link to specific commit for permanence (#552) |
| 072affe | 2020-09-16 | Paul Donahue | Follow suggestion in #544 (#548) |
| 8345674 | 2020-09-15 | Paul Donahue | Add dret to rule 8. (#547) |
| 175090c | 2020-09-11 | Paul Donahue | Add mcontrol6. (#538) |
| de1ec1a | 2020-09-10 | Paul Donahue | Fix links to point to fields in the correct registers. (#546) |
| 4f625ca | 2020-09-03 | Ernie Edgar | Add scontext2 alias for scontext (#535) |
| 7c0a6d5 | 2020-08-28 | Jan Matyas | aamvirtual: Clarification for systems without address translations (#542) |
| 30b1a97 | 2020-08-24 | Tim Newsome | Remove end-of-line whitespace in generated comments. (#540) |
| 6e90a60 | 2020-08-21 | Tim Newsome | Add header to debug_defines.h (#539) |
| 0200b27 | 2020-08-21 | Tim Newsome | Improve formatting of autogenerated C header files (#537) |
| 97d51c2 | 2020-08-11 | Tim Newsome | Rebuild PDF. |
| fcf4002 | 2020-08-11 | Tim Newsome | authdata should only should be implemented if used (#521) |
| 0570f14 | 2020-08-05 | Paul Donahue | Add abstractcs.relaxedpriv (#536) |
| 2210002 | 2020-07-07 | Tim Newsome | Rebuild PDF. |
| b9959e5 | 2020-06-30 | Tim Newsome | Make explicit that aampostincrement is optional. (#532) |
| 67fed8f | 2020-06-19 | Tim Newsome | Explicitly allow uni-directional external triggers. (#526) |
| 9c69bf3 | 2020-06-09 | Tim Newsome | Rebuild PDF. |
| 85bf4df | 2020-05-21 | Tim Newsome | Add Kai Meinhard to contributors list. |
| 2f1c133 | 2020-05-21 | Kai Meinhard | Appendix B suggests signals for a DMI with one DTM connected to one DM (#524) |
| 708b1e0 | 2020-04-10 | Tim Newsome | Add Larry Madar. |
| e02a8b6 | 2020-04-07 | Tim Newsome | Rebuild PDF. |
| 372b27f | 2020-03-23 | Tim Newsome | All tdata functionality is optional... (#444) |
| 50f5c8f | 2020-03-11 | Tim Newsome | Explicitly allow hard-coded halt/resume groups. (#517) |
| f4794bb | 2020-03-10 | Tim Newsome | Rebuild PDF. |
| e3ec24e | 2020-02-13 | bdwyatt | Adding version encoding for 0.14 spec. (#512) |
| cf9a884 | 2020-02-11 | Tim Newsome | Rebuild PDF. |
| fdd5ad6 | 2020-02-11 | Philipp Wagner | dcsr.prv should be WARL, not R/W (#498) |
| 38b2794 | 2020-02-11 | Tim Newsome | sizehi only exists if Xlen>64. (#514) |
| 5a54283 | 2020-01-16 | Tim Newsome | Use exception, trap, and interrupt as in ISA spec (#511) |
| a989a71 | 2020-01-13 | Tim Newsome | Clarify dmireset/dmihardreset. (#508) |
| d10d8d0 | 2020-01-06 | Tim Newsome | Rebuild PDF. |

| | | | |
|---|---|---|---|
| efc0143 | 2020-01-06 | Tim Newsome | Clarify action=1 (enter Debug Mode) with dmode=0 (#501) |
| 439fb93 | 2020-01-06 | Tim Newsome | Fix conflict in sbdata0/sbautoincrement definition. (#507) |
| d35ce10 | 2019-12-10 | Tim Newsome | Add resume groups. (#506) |
| 2726f30 | 2019-12-06 | Tim Newsome | Rebuild PDF. |
| a310a37 | 2019-12-04 | Tim Newsome | Make haltsum0 optional if there is only one hart. (#505) |
| 349c826 | 2019-11-26 | Tim Newsome | Halt state may not be preserved across reset. (#504) |
| 4ab79d7 | 2019-11-26 | Tim Newsome | Clear MPRV when resuming into lower privilege mode. (#503) |
| c9c286b | 2019-11-22 | Tim Newsome | Time may pass before dmactive becomes high. (#500) |
| 9d55a57 | 2019-11-21 | Megan Wachs | Make the emitted registers chisel3 |
| 014505f | 2019-10-08 | Tim Newsome | Rebuild PDF. |
| 62c63b8 | 2019-10-04 | Tim Newsome | Document forward progress guarantees in Debug Mode. (#496) |
| d933bec | 2019-10-02 | Tim Newsome | Rewrite/clarify DM Reset Control (#494) |
| 039bd5a | 2019-09-23 | Philipp Wagner | Fix wrong table reference (#484) |
| 106b4f2 | 2019-09-16 | Tim Newsome | DM reset must also reset all the DM's harts. (#493) |
| 8bfcd17 | 2019-09-13 | Tim Newsome | Explicitly list cmderr=6 (reserved). (#491) |
| 448de85 | 2019-09-12 | Philipp Wagner | dmcontrol.hartreset is WARL, not R/W (#490) |
| 8637b3c | 2019-09-10 | Tim Newsome | Rebuild PDF. |
| f00f436 | 2019-09-10 | Philipp Wagner | Tiny style fix for email "link"on title page (#486) |
| 3646788 | 2019-09-10 | Philipp Wagner | Fix page references in cmdtype table (#487) |
| 99ae160 | 2019-09-09 | Megan Wachs | Update implementations.tex (#482) |
| f9c9ed4 | 2019-09-04 | Philipp Wagner | Update registers.py to use Python 3 (#483) |
| 37d8ee1 | 2019-09-03 | Philipp Wagner | Git ignore intermediate and output files (#485) |
| 1e99ce7 | 2019-08-13 | Tim Newsome | Tighten up trigger specification. (#478) |
| a121ee1 | 2019-08-13 | Tim Newsome | Rebuild PDF. |
| 7d126a9 | 2019-07-16 | Tim Newsome | Mention the scontext reg number isn't conventional (#474) |
| b5df5bd | 2019-07-16 | Tim Newsome | Explicitly document confstrptr[1-3]. (#475) |
| e6311af | 2019-07-12 | Tim Newsome | Change R/W1C to reduce requirements on hardware. (#472) |
| 178e749 | 2019-07-11 | Tim Newsome | Define what we mean by virtual address. (#473) |
| 340c302 | 2019-07-09 | Tim Newsome | Rebuild PDF. |
| 77d58e6 | 2019-07-08 | Tim Newsome | Numerous tweaks, responding to Marc Gauthier (#463) |
| ab89a86 | 2019-07-04 | Tim Newsome | Addressing more feedback from Marc Gauthier. (#465) |
| 624a6b8 | 2019-06-26 | Tim Newsome | Without S-mode, textra.svalue and .sselect should be 0 (#469) |
| 1977166 | 2019-06-11 | Tim Newsome | Rebuild PDF. |
| b06eb70 | 2019-06-06 | Tim Newsome | Clarify mcontrol.size. (#460) |
| 165f120 | 2019-05-29 | Tim Newsome | Fully qualify register/field macro names. (#457) |
| c47f0a0 | 2019-05-29 | Paul Donahue | Fix #452 (#459) |
| 633ee13 | 2019-05-28 | Paul Donahue | Fixed #453 (#458) |
| 96ef519 | 2019-05-20 | Tim Newsome | The *external* debugger must restore tselect. (#456) |

| | | | |
|---|---|---|---|
| e11f777 | 2019-05-08 | Tim Newsome | Rebuild PDF. |
| 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 sbbusyerror 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. |
| aeee8f3 | 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. -> 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 |
| fffe3c2 | 2018-04-10 | Tim Newsome | Clarify SBA unsupport 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 inaccuarices 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 | hawindowsel 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->haltsum |
| ee51758 | 2018-02-14 | Megan Wachs | Modification of > 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 IRMAK | 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->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 | sbaddress: 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 \Fsberror |
| 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 -> tval (Priv Spec 1.9 -> 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 \Fimpebreak does. |
| 78082b5 | 2017-10-11 | Tim Newsome | Mention \Fimpebreak 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 hmode -> dmode replacement |
| 16e11f3 | 2017-10-11 | Tim Newsome | Remove hmode reference, to fix build. |
| 84b9a6a | 2017-10-11 | Tim Newsome | Add \Fimpebreak, to support of implicit ebreak. |
| cc90b77 | 2017-10-11 | mwachs5 | Remove reference to 'H' mode from the figure |
| cc6a9de | 2017-10-11 | Megan Wachs | Change old reference to 'hmode' to 'dmode' |
| 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 | haltreq, resumereq, hartreset are per-hart bits |
| d4118ab | 2017-09-30 | Tim Newsome | ndmreset can't reset logic required to access DM. |
| c6bd8d1 | 2017-09-29 | Tim Newsome | and -> or |
| 58c2441 | 2017-09-29 | Tim Newsome | Mention \Fstepie in Single Step |
| 94c5f78 | 2017-09-29 | Tim Newsome | Clarify ndmreset. |
| 12810b4 | 2017-09-29 | Tim Newsome | Clarify that sbaddress is physical. |
| 5862fdf | 2017-09-29 | Tim Newsome | Unify M mode and mprv comment. |
| aea1bd5 | 2017-09-29 | Tim Newsome | Define behavior when haltreq and resumereq are set |
| 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 \Fdmerr. |
| 25685eb | 2017-09-28 | Tim Newsome | Explain that bus master or progbuf is required. |
| f75ee7d | 2017-09-28 | Tim Newsome | Clarify debugger can discover "almost"everything |
| 71e6788 | 2017-09-27 | Tim Newsome | Remove description of manual stepping. |
| 9aea347 | 2017-09-27 | Tim Newsome | Move Running/Single Step near Halting. |
| 2090d9b | 2017-09-27 | Tim Newsome | data0 should be sbdata0 in this table. |
| 5858cfe | 2017-09-27 | Tim Newsome | Clarify why \Rpriv exists. |
| bc3c2aa | 2017-09-27 | Tim Newsome | Mention where priv encoding comes from. |
| ef77cc4 | 2017-09-27 | Tim Newsome | One more attempt to clarify DPC after single step. |
| 80a288e | 2017-09-27 | Tim Newsome | Clarify instret not incrementing on ebreak. |
| c163d22 | 2017-09-20 | Tim Newsome | Remove ebreakh. |
| 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 sbdata0. |
| d4eda18 | 2017-09-20 | Tim Newsome | Explain that only sbdata0 has side effects. |
| ae781c6 | 2017-09-20 | Tim Newsome | Don't refer to internal system bus registers. |
| 875922e | 2017-09-20 | Tim Newsome | Explain sbdata0 being stale a bit more. |
| cd44fd5 | 2017-09-20 | Tim Newsome | Clarify autoread |
| 194484b | 2017-09-20 | Tim Newsome | Clarify hawindow. |
| 02f1aac | 2017-09-20 | Tim Newsome | Clarify that \Fdataaddr is relative to \Rzero. |
| 0e9b6ae | 2017-09-20 | Tim Newsome | Clarify nonexistent vs unavailable. |
| b55ff41 | 2017-09-20 | Tim Newsome | Fix devtreevalid. |
| 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 \Ftransfer might be used. |
| 6b2ee61 | 2017-09-20 | Tim Newsome | Explain where \Fsize encoding came from. |
| c9f3b73 | 2017-09-14 | Tim Newsome | Fix typo. |
| 4b25400 | 2017-09-13 | Tim Newsome | Mention dpc in CSRs abstract register numbers. |
| c3ee426 | 2017-09-13 | Tim Newsome | Move abstract regno table closer to its reference. |
| 111b9a3 | 2017-09-13 | Tim Newsome | cycle -> operation |
| 994afdc | 2017-09-13 | Tim Newsome | Account for multiple selected harts. |
| aa4a297 | 2017-09-13 | Tim Newsome | Halt Control -> Run Control |
| e97c821 | 2017-09-13 | Tim Newsome | continuous -> contiguous |
| 97f73ff | 2017-09-13 | Tim Newsome | Clarify ndmreset behavior. |
| 6078220 | 2017-09-13 | Tim Newsome | Explain ndmreset |
| a3d4f30 | 2017-09-13 | Tim Newsome | Describe ‘halt region‘ |
| 272b3d9 | 2017-09-13 | Tim Newsome | Clarify accessing unimplemented DM DMI regs |
| 3e91f1b | 2017-09-13 | Tim Newsome | Clarify either Prog Buf or Sys Bus Acc is required |
| e8a6145 | 2017-09-13 | Tim Newsome | Clarify CSR access; remove serial port |
| ce20766 | 2017-09-13 | Tim Newsome | Remove section referencing itself. |
| 1195a61 | 2017-09-18 | Tim Newsome | Generate constants to be unsigned for clang. |
| 8967b0a | 2017-08-16 | Megan Wachs | Compressed instructions are c.foo, not foo.c |

| | | | |
|---|---|---|---|
| b5698a9 | 2017-08-16 | Megan Wachs | clarify progbufsize description |
| d221bab | 2017-08-16 | Megan Wachs | Remove progbufsize enums from register description |
| 0498102 | 2017-08-16 | Megan Wachs | appendix: Use standard assembly format for sw |
| 4456d99 | 2017-08-09 | Tim Newsome | Rename progsize to progbufsize. |
| 55d5b66 | 2017-08-09 | Tim Newsome | Clarify that trigger comparisons are unsigned. |
| 21e35ef | 2017-08-09 | Tim Newsome | Configuration String -> 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 dpc 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 abstract_commands.xml |
| 6e8cdf1 | 2017-07-10 | Megan Wachs | Update abstract_commands.xml |
| cf6e3f2 | 2017-07-10 | Megan Wachs | clarify DCSR.cause |
| 79ffbb9 | 2017-07-10 | Megan Wachs | Clarify implications of CSR read, write, halt |
| 013e191 | 2017-07-10 | Megan Wachs | Clarify when you would get error halt/resume |
| 231e457 | 2017-07-10 | Megan Wachs | Quick Access error clarification |
| 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 (Temporary Acct.) | Remove virtual register from core_registers.xml |
| bfc64fb | 2017-05-10 | Megan Wachs (Temporary Acct.) | Add missing sw_registers.tex file |
| 0512f5d | 2017-05-06 | mwachs5 | Move virtual 'prv' register to a seperate 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 instruciton 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 seperate files. |
| 287d5c6 | 2017-04-03 | mwachs5 | moving more files to seperate tex files. |
| 9e873f4 | 2017-04-03 | mwachs5 | move trigger info into seperate file. |
| 2c89a86 | 2017-04-03 | mwachs5 | move risc-v core debug info into seperate file. |
| e676491 | 2017-04-03 | mwachs5 | Move System Overview to seperate file |
| 03df6ee | 2017-04-03 | mwachs5 | Move Debug Module description to a seperate 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 seperate 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 seperate 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 seperation 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->progbuf |
| 9666e51 | 2017-02-22 | mwachs5 | IBUF->PROGBUF |
| 5308ecd | 2017-02-22 | mwachs5 | Remove last references to "Instruction Supply" |
| f6ebde9 | 2017-02-22 | Tim Newsome | Move authentication to a serial protocol. |
| 0f079c8 | 2017-02-22 | Tim Newsome | Reserve bit for per-hart reset. |
| f2c93ac | 2017-02-22 | Tim Newsome | Clarify that dmactive resets authentication. |
| f5e7b1c | 2017-02-22 | Alex Bradbury | Clarify that the halt state of all harts is maintained through reset |
| 3dfe8fd | 2017-02-22 | Tim Newsome | More Debug Mode -> Halt Mode. |
| d29fc1f | 2017-02-22 | Tim Newsome | Debug Mode -> Halt Mode |
| 55d6030 | 2017-02-21 | Tim Newsome | Generate debug_defines.h as part of normal make |
| b0e6a7f | 2017-02-21 | Tim Newsome | Minor clarifications. |
| 0f9885c | 2017-02-20 | Tim Newsome | Various clarifications. |
| 0802d5a | 2017-02-15 | mwachs5 | Use consistent 'Control and Status' naming for CS registers. |
| 5accc7d | 2017-02-15 | Tim Newsome | Change all the "other"JTAG IRs to just reserved. |
| bcbd7da | 2017-02-15 | mwachs5 | sm_diagram: Show using resumereq bit to resume. |
| 18f6e55 | 2017-02-14 | Tim Newsome | Introduce resumereq command, similar to haltreq. |
| 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->DMI |
| 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 >32-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 -> preexec, execa -> 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. |
| 9ccef3d | 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 |