RISC-V Debug 调试

JMJ （直译和意译）

可能有不准确的地方，可提供建议

目录

1 简介

当设计从模拟发展到硬件实现时，用户对系统当前状态的控制和理解会急剧下降。为了帮助开发和调试低级软件和硬件，在硬件中内置良好的调试支持是至关重要的。当一个健壮的操作系统在内核上运行时，软件可以处理许多调试任务。然而，在许多情况下，硬件支持是必不可少的。

本文概述了 RISC-V 平台上外部调试支持的标准体系结构。这种架构允许多种实现和交易，这是对各种 RISC-V 实现的补充。同时，该规范定义了通用接口，允许调试工具和组件可以基于 RISC-V ISA 的各种平台。

系统设计人员可以选择添加额外的硬件调试支持，但此规范定义了通用功能的标准接口。

1.1 术语

平台是由一个或多个组件组成的单个集成电路。一些组件可能是 RISC-V 核心，而其他组件可能有不同的功能。通常，他们都将连接到单个系统总线。单个 RISC-V 内核包含一个或多个硬件线程，称为 harts。

Hart 的 DXLEN 是它最广泛支持的 XLEN，忽略 misa 寄存器中的 MXL 的当前值。

1.1.1 上下文

本文编写的工作环境：

1、 RISC-V 指令集手册，第一卷《User-Level ISA， Document Version 2.2》（ISA 规范）

2、 RISC-V 指令集手册，第二卷《Privileged Architecture, Version 1.10》（特权规范）

1.1.2 版本

本文 0.13 版本是由 RISC-V 基金会董事会批准。版本 0.13.x 是针对一些 bug 的 fix 版本。

版本 0.14 将兼容版本 0.13。

1.2 本文内容

1.2.1 结构

本文由两部分组成。主要部分是规范，在编号章节部分给出。另一部分是一套附录，附录中的信息旨在澄清和提供示例，但不是实际规格的一部分。

1.2.2 寄存器定义格式

本文中所有寄存器定义均遵从以下格式。有一个简单的图形显示哪些字段在寄存器中。上下位宽索引显示在每个字段的左上角和右上角。字段的总位数也在下面表示。

在图形后面是一个表格，其中列出了每个字段的名称，说明，允许访问的类型和 reset 值。表 1.2 列出了允许的访问类型。Reset 值可以是常量也可以是“预设值”，后者意味着它是一个特定于实现的合法值。

寄存器及其字段的名称是指向其定义的超链接。由于时间有限，本人 JMJ 可能并不是所有超链接都做了，所以看得时候有没有超链接的，就自己前后翻看本文。

Table 1.2: Register Access Abbreviations

R	Read-only.
R/W	Read/Write.
R/W1C	Read/Write. For each bit in the field, writing 1 clears that bit. Writing 0 has no effect.
W	Write-only. When read this field returns 0.
W1	Write-only. Only writing 1 has an effect.
WARL	Write any, read legal. A debugger may write any value. If a value is unsupported, the implementation converts the value to one that is supported.

1.2.3 Long Name（shortname, at 0x123）

名称	描述	访问类型	Reset 值
field	描述这个字段的功能	R/W	15

1.3 背景

专用调试硬件有几种用例，既在 CPU 内核内部，也在外部连接中。本规范解决了下面列出的用例。可以选择不实现所有功能，这意味着可能不支持某些用

例。

- 在没有操作系统或其他软件的情况下调试底层软件。
- 操作系统本身的调试问题。
- 在系统中任意可执行代码执行之前，先引导系统进行测试，配置和编程组件系统中。
- 在没有可用 CPU 的情况下访问系统上的硬件。

此外，即使没有硬件调试接口，RISC-V CPU 中的体系结构可以通过硬件触发和断点来帮助软件调试和性能分析。

1.4 支持的功能

本规范中描述的调试接口支持以下功能：

1.所有 hart 寄存器（包括 CSR）可以读取/写入。

2.可以从 hart 的角度访问内存，或直接通过系统总线访问内存，或两者同时访问内存。

3.都支持 RV32，RV64 和将来的 RV128。

4.平台中的任何故障位均可独立调试。

5.调试器无需用户配置即可发现几乎需要了解的所有内容。

6.可以从执行的第一条指令调试每个 hart。

7.执行软件断点指令时，可以停止 RISC-V hart。

8.硬件单步执行，一次可以执行一条指令。

9.调试功能独立于所使用的调试传输。

10.调试器不需要了解有关正在调试的 harts 的微体系结构的任何信息。

11.任意子集可以同时停止和恢复。（可选的）

12.任意指令可以在挂起的 hart 中执行。这意味着，当内核具有其他或自定义的指令或状态时，只要存在可以将该状态移入 GPR 的程序，就不需要新的调试功能。（可选的）

13.可以在不挂起的情况下，访问寄存器。（可选的）

14.运行中的 hart 可以直接执行一小段指令，而开销很小。（可选的）

15.系统总线主控器允许在不涉及任 hart 的情况下进行内存访问。（可选的）

16.当触发器与 PC，读/写地址/数据或指令操作码匹配时，可以停止 RISC-V 中的

hart。（可选的）

本文档不建议硬件测试，调试或错误检测技术策略的实现。Scan，BIST 等不在本规范的范围之内，但是本规范无意限制它们在 RISC-V 系统中的使用。可可以调试使用软件线程的代码，但是对此没有特殊的调试支持。

2 系统介绍

图 2.1 显示了外部调试支持的主要组件。虚线所示的块是可选的。

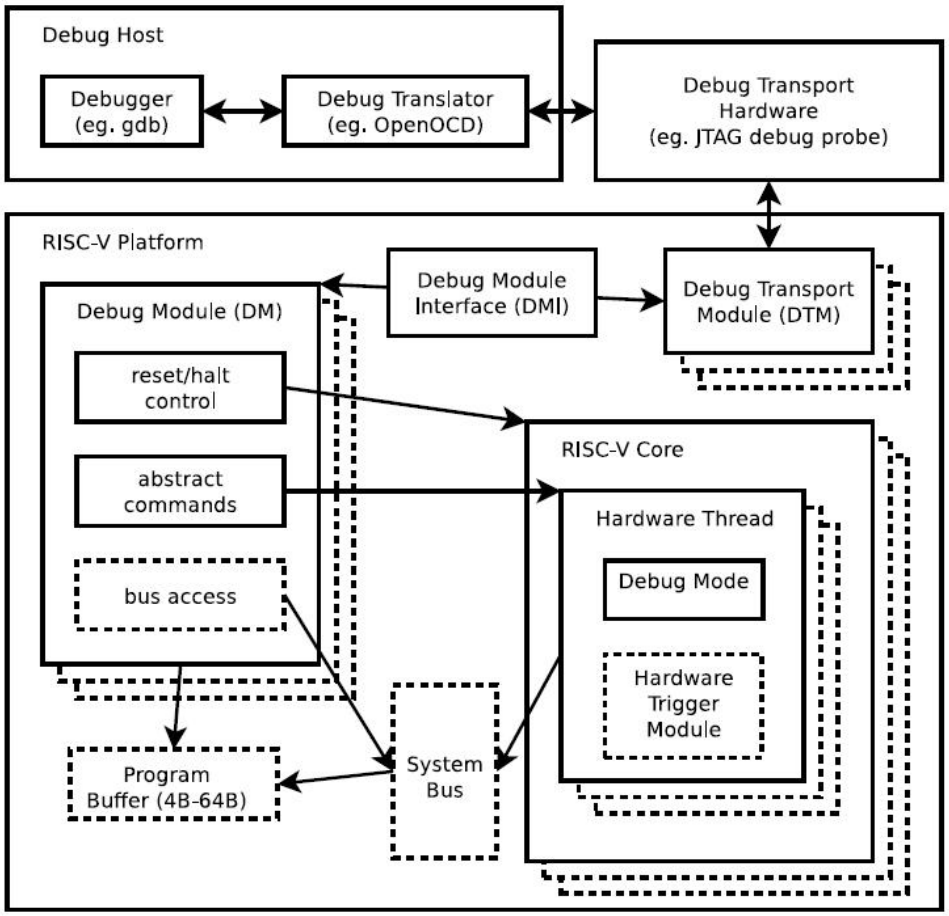


Figure 2.1: RISC-V Debug System Overview

用户与调试主机 Debug Host（例如笔记本电脑）进行交互，调试器 debugger（例如 gdb）运行在调试主机上。

调试器 debugger 通过与调试转换器 Debug Translator（例如 OpenOCD，可能包含硬件驱动程序）进行通信，以实现与调试传输硬件 Debug Transport Hardware（例如 Olimex USB-JTAG 适配器）进行通信。

调试传输硬件 Debug Transport Hardware 将调试主机 Debug Host 连接到平台的调试传输模块（DTM, Debug Transport Module）上。DTM 使用调试模块接口（DMI）提供对一个或多个调试模块（DM）的访问。

平台中的每个 hart 均由一个 DM 控制。hart 可能是各种各样的的。Hart-DM 映射没有更多限制，但是通常单个核心中的所有 hart 都由同一 DM 控制。在大多数平台中，只有一个 DM 可以控制平台中的所有 hart。

DM 可以在平台中中对 hart 提供运行控制。抽象命令提供对 GPR 的访问。可通过抽象命令或通过编写程序来访问其他寄存器，其中编写的程序是写入可选的程序缓冲区 Program Buffer。

程序缓冲区允许调试器在 hart 上执行任意指令。此机制也可以用于访问内存。可选的系统总线访问块允许存储访问，而无需使用 RISC-V Hart 进行访问。

每个 RISC-V 架构都可以实现触发模块。当满足触发条件时，hart 将停止并通知调试模块它们已停止。

3 Debug 模块（DM）

调试模块在抽象调试操作与特定实现之间实现转换接口。它可能支持以下操作：

1.为调试器提供有关实现的必要信息。（需要）

2.允许任何 hart 停止和恢复。（需要）

3.提供挂起的 hart 的状态。（需要）

4.提供对挂起的 hart 的 GPR 的抽象读写访问。（需要）

5.提供对复位信号的访问，允许在 reset 之后的第一个指令进行调试。（需要）

6.提供一种调试机制，允许 hart 立即退出重置(无论重置原因如何)（可选的）

7.提供对非 GPR 的 hart 寄存器的抽象访问。（可选的）

8.提供一个程序缓冲区 Program Buffer 以强制 hart 执行任意指令。（可选的）

9.允许同时停止，恢复和/或重置多个 hart。（可选的）

10.允许从 hart 的角度进行内存访问。（可选的）

11.允许直接访问系统总线。（可选的）

为了符合此规范，必须实现：

1.实现上面列出的所有必需功能。

2.实现程序缓冲区，系统总线访问或抽象访问存储器命令机制中的至少一种。

3.至少执行以下一项操作：

（a）实施程序缓冲区 Program Buffer。

（b）对运行在 hart 上的软件可见的所有寄存器实施抽象访问，包括存在于 hart 上并在表 3.3 中列出的所有寄存器。

（c）对至少所有 GPR，dcsr 和 dpc 实施抽象访问，并宣传该实现符合“Minimal RISC-V Debug Specication 0.13.2”而不是“RISC-V Debug Specification 0.13.2”。

单个 DM 最多可以调试 2^20 个 hart。

3.1 调试模块接口（DMI）

调试模块是调试模块接口（DMI）总线的从属。总线的主设备是调试传输模块 DTM。调试模块接口 DMI 可以是简单的总线，具有一个主机和一个从机，或

者使用功能更全的总线，例如 TileLink 或 AMBA 高级外围总线。细节留给系统设计者。

DMI 使用 7 到 32 个地址位。它支持读取和写入操作。地址空间的底部通常用于第一个 DM，多余的空间可用于定制调试设备，其他内核，其他 DM 等。如果此 DMI 上还有其他 DM，则 DMI 地址空间中下一个 DM 的基地址在 nextdm 中给出。

通过对 DMI 地址空间寄存器的访问来控制调试模块。

3.2 重置控制

调试模块控制全局重置信号 ndmreset（非调试模块重置 non-debug module reset），该信号可以重置或保持重置平台中的每个组件，DM 和 DTM 除外。可以从执行的第一个指令调试程序，此复位的确切含义取决于实现。调试模块 DM 自己的状态和寄存器仅应在上电时重置，dmcontrol 中的 dmactive 为 0。如果 dmactive 为 1，则应该在系统重置期间保持 harts 的挂起状态，触发 CSR 可能会被清除。

由于时钟和电源域交叉问题，可能无法在系统重置期间执行任意 DMI 访问。当 ndmreset 或任何外部重置时，仅支持的 DM 操作是访问 dmcontrol。其他访问的行为是不确定的。

对 ndmreset 的持续时间没有要求。该实现必须确保将 ndmreset 写入 1 之后再将 ndmreset 写入 0 触发系统复位。如 allunavail 和 anyunavail 所报告的，系统可能要花费任意长时间才能复位。

单个 hart（或一次多个）可以通过设置然后清除 hartreset 来重置。在这种情况下，可能会实现重置更多的 hart，而不仅仅是所选的。通过选择调试器并检查 anyhavereset 和 allhavereset，调试器可以发现重置了哪些其他 hart（如果有）。

当 hart 重置后，必须将 havereset 状态位置为 sticky。可以在 dmstatus 的 anyhavereset 和 allhavereset 中为选定的 harts 读取概念性的 havereset 状态位。这些位无论复位原因如何，都必须置位。可以通过在 dmcontrol 中的 ackhavereset 中写入 1 来清除所选 harts 的 havereset 位。当 dmactive 为低电平时，havereset 位可以清除或不清除。

当 hart 退出复位并设置了 haltreq ~~或 resethaltreq~~ 时（C.1.4 更新），hart 将立

即进入调试模式。否则它将正常执行。

3.3 选择 hart

单个 DM 最多可以连接 2^20 个 hart。调试器选择一个 hart，然后该 hart 进行停止，恢复，重置和调试命令。

要枚举所有的 hart，首先调试器必须确定 HARTSELLEN，通过写全 1 到 hartsel（假定最大 size）并读回值以查看实际大小。然后，选择从 0 开始的每个 hart，直到 dmstatus 中的 anynonexistent 为 1，或者达到最高索引（取决于 HARTSELLEN）。

调试器可以使用两种方式来发现 hart 索引与 mhartid 之间的映射：使用接口读取 mhartid；读取系统的配置字符串。

3.3.1 选择单个 hart

所有调试模块都必须支持选择单个 hart。调试器可以通过将索引写入 hartsel 来选择 hart。Hart 索引从 0 开始，并且连续直到最终索引。

3.3.2 选择多个 hart

调试模块可以实现 Hart Array Mask 寄存器，允许一次选择多个 Hart。Hart Array Mask 寄存器中的第 n 位适用于索引为 n 的 hart。如果该位为 1，则选择 hart。通常，DM 将具有足够宽的 Hart Array Mask 寄存器以选择其支持的所有 Hart，但是允许将这些位中的任何一个绑定为 0。

调试器可以使用 hawindowsel 和 hawindow 来设置 hart 的 Hart Array Mask 寄存器中的位，然后通过设置 hasel 将操作应用于所有选定的 harts。如果支持此功能，则可以进行多个 hart 同时停止，恢复和重置。Hart Array Mask 寄存器的状态不受 hasel 的设置或清除影响。

dmcontrol 发起的动作可以一次应用于多个 hart，Abstract Commands 抽象命令仅应用于 hartsel 选择的 hart。

3.4 Hart 状态

每个被选中的 hart 会处于四个状态之一：存在，不可用，正在运行和暂停。如下表所示。

如果永远不将其纳入该系统，则无论用户等待多长时间，hart 都不存在。例

如，在简单的单一 hart 系统中，只有一个 hart 存在，而其他所有 hart 都不存在。调试器可能会认为系统没有索引高于第一个不存在的索引。

Hart 状态	访问类型	
存在	allnonexistent	anynonexistent
不可用	allunavail	anyunavail
正在运行	allrunning	anyrunning
暂停	allhalted	anyhalted

如果以后可能存在/变得可用，或者其他 hart 的索引高于此索引，则 hart 不可用。可能由于多种原因无法使用 Harts，包括重置，暂时关闭电源以及未将其插入系统。具有大量 hart 的系统可能会在制造期间永久禁用，否则会留下漏洞空间。为了使调试器发现所有 hart，即使它们不可能出现，它们也必须显示为不可用。

Hart 在正常执行时会运行，就像没有连接调试器一样。这包括处于低功耗模式或等待中断，只要停止请求发出将导致 hart 被挂起。

当处于调试模式时，Hart 将挂起，仅代表调试器执行任务。

复位的 hart 的状态依赖于实现。复位时以及解除复位后的一段时间，可能无法使用 Harts。他们可能会在解除复位后的一段时间运行。最终，根据 haltreq 和 resethaltreq 的值，它们最终会选择运行还是停止。

3.5 运行控制

对于每个 hart，调试模块都会跟踪 4 个概念状态位：停止请求，恢复确认，复位后停止请求和复位（后两个请求位是可选的），复位值和对应关系如下表。

状态位	对应名字	Reset 值
halt request	停止请求	0
resume ack	恢复确认	0 或 1
halt-on-reset request	复位后停止请求	0
hart reset	复位	0

DM 从每个 hart 接收停止，运行和 havereset 信号。调试器可以在 allresumeack 和 anyresumeack 中观察 resume ack 的状态，在 allhalted 和 anyhalted 中观察 halted 的状态，在 allrunning 和 anyrunning 中观察 running 的状态，在 allhavereset 和 anyhavereset 观察 havereset 的状态。停止的 halt 会忽略其停止请求位。

当调试器将 1 写入 haltreq 时，每个选定的 hart 的暂停请求位都将置 1。当运行中的 hart 或刚退出重置的 hart 看到其暂停请求为高时，它会通过暂停进行响

应，取消其运行信号，并声明其停止信号。停止的 hart 会忽略其停止请求位。

当调试器将 1 写入 resumereq 时，将清除每个选定的暂存器的恢复确认位，并向每个选定的已暂停的 hart 发送恢复请求。Hart 通过恢复，清除其停止信号，并声明它的运行信号来作出反应。~~在此过程结束时，将恢复确认位 resume ack 置位。~~恢复确认位 **resume ack 在 resuming 后置位**（附录 C.1.1 更新）。所有选中的 hart 的状态信号会以 allresumeack，anyresumeack，allrunning 和 anyrunning 表示。正在运行的 hart 会忽略恢复请求。

除非无法使用，否则当请求停止或恢复时，hart 必须一秒内做出响应。（这是如何实现的不作进一步说明，几个时钟周期将是一个更典型的等待时间）。

DM 可以通过将 hasresethaltreq 设置为 1，来指示每个 HART 的可选的复位停止位。这意味着 DM 要实现 setresethaltreq 和 clrresethaltreq 位。将 1 写入 setresethaltreq 会为每个选定的 Hart 设置复位停止请求位。当设置了 hart 的复位暂停请求位时，下次重置后，该 hart 将立即进入调试模式。不论重置原因如何都是如此。Hart 的复位暂停请求位保持置位，直到选定 hart 的调试器将 1 写入 clrresethaltreq 或通过 DM 复位将其清除为止。

3.6 抽象命令

DM 支持一组抽象命令，其中大多数是可选的。根据实现的不同，即使选定的 hart 没有停止，调试器也能够执行一些抽象命令。调试器只能在给定的状态下尝试去 abstractcs 中查看 cmderr 来确定它们是否成功，来确定选择的 hart 支持哪些抽象命令。某些选项集可能支持命令，而其他选项集则不支持。如果命令设置了不支持的选项，则 DM 必须将 cmderr 设置为 2（不支持）。

示例：每个系统都必须支持访问寄存器命令，但可能不支持访问 CSR。如果在这种情况下调试器请求读取 CSR，则该命令将返回“不支持”。

调试器通过将抽象命令写入 command 来执行它们。可以通过读取 abstractcs 中的 busy 来确定抽象命令是否完成，完成后，cmderr 表示命令是否成功。命令可能会失败，因为 hart 未停止，未运行，不可用，或者因为它们在执行期间遇到错误。

如果命令带有参数，则调试器必须在写入 command 之前将参数写入 data 寄存器。如果命令返回结果，则调试模块 DM 必须确保在 busy 被清之前放入 data 寄存器中。表 3.1 中描述了哪些 data 寄存器被用作参数变量。在所有情况下，最

低有效字都放置在编号最低的 data 寄存器中。参数宽度取决于正在执行的命令，在未明确指定的情况下为 DXLEN。

Table 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

抽象命令接口旨在允许调试器尽可能快地编写命令，然后再检查它们是否已正确完成。在通常情况下，调试器的运行速度将比目标运行速度慢得多，并且命令执行成功，可以实现最大的吞吐量。如果发生故障，该接口可确保在发生故障后没有命令执行。为了发现哪个命令失败，调试器必须查看 DM 的状态（例如 data0 的内容）或 hart 的状态（例如 Program Buffer 程序修改的寄存器的内容）以确定哪个命令失败。

在启动抽象命令之前，调试器必须确保 haltreq，resumereq 和 ackhavereset 都为 0。

虽然一个抽象命令在执行时（abstractcs[busy]为高），调试器不能改变 hartsel，并且不能写 1 到 haltreq，resumereq，ackhavereset，setresethaltreq 和 clrresethaltreq 字段。

如果抽象命令未在预期的时间内完成，并且似乎已挂起，则可以尝试以下过程来中止该命令：首先，调试器重置 hart（使用 hartreset 或 ndmreset），然后重置调试模块 DM（使用 dmactive）。

如果选定的 hart 不可用时启动了抽象命令，或者在执行一个抽象命令的过程中 hart 不可用，则调试模块可以终止抽象命令，将 busy 设置为低，并将 cmderr 设置为 4（停止/恢复）。或者，该命令只是挂起了（busy 不会变低）。

3.6.1 抽象命令清单

本节描述了每个不同的抽象命令，以及将它们写入 command 时应如何解释它们的字段。

每个抽象命令是一个 32 位值。前 8 位包含 cmdtype，它确定命令的类型。表 3.2 列出了所有命令。

Table 3.2: Meaning of cmdtype

cmdtype	Command
0	Access Register Command
1	Quick Access
2	Access Memory Command

3.6.1.1 访问寄存器

该命令使调试器可以访问 CPU 寄存器，并允许其执行程序缓冲区。它执行以下操作序列：

1.如果 write 被清除且 transfer 置位，则将数据从 regno 指定的寄存器复制到 data 的 arg0 区域，并执行从 M 模式读取该寄存器时发生的任何副作用。

2.如果置位了 write 和 transfer，则将数据从 data 的 arg0 区域复制到 regno 指定的寄存器中，并执行从 M 模式写入该寄存器时发生的任何副作用。

3.如果设置了 aarpostincrement，则增加 regno。

4.如果设置了 postexec，执行 Program Buffer。

如果这些操作中的任何一个失败，则将设置 cmderr，并且不执行其余任何步骤。一个实现可能会及早发现即将发生的故障，并在导致失败的步骤到达之前让整个命令 fail。如果失败是因为请求的寄存器不存在于 hart 中，则必须将 cmderr 设置为 3（例外）。

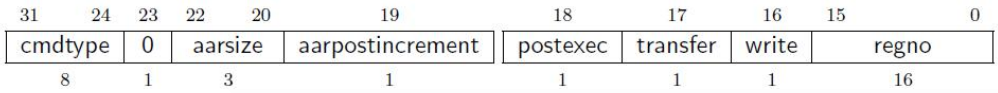
调试模块必须实现此命令，并且在选定的 hart 停止时，支持对所有 GPR 的读写访问。调试模块可以选择支持访问其他寄存器，或在 hart 运行时访问寄存器。在读取，写入和暂停状态下，可以分别支持每个寄存器（除了 GPR）。

选择 aarsize 的编码以匹配 sbcs 中的 sbaccess。

仅当读取寄存器时，此命令才会修改 arg0。其他 data 寄存器不变。

Table 3.3: Abstract Register Numbers

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.



名称	描述
cmdtype	为 0 表示访问寄存器命令。
aarsize	2：访问寄存器的最低 32 位。 3：访问寄存器的最低 64 位。 4：访问寄存器的最低 128 位。 如果 aarsize 指定的大小大于寄存器的实际大小，则访问必须失败。如果可访问寄存器，则必须支持小于或等于寄存器实际大小的读取。 该字段控制引用的参数宽度见表 3.1。
aarpostincrement	0：无效。这种变体必须得到支持。

	1：寄存器访问成功后，regno 递增（缠绕到 0）。支持此变体是可选的。
postexec	0：无效。必须支持此变体，如果 progbufsize 为 0，则是唯一支持的变体。 1：如果有的话，执行传输后，在程序缓冲区中恰好执行一次程序。支持此变体是可选的。
transfer	0：不执行 write 指定的操作。 1：执行 write 指定的操作。 该位可用于仅执行程序缓冲区，而不必担心将有效值放入 aarsize 或 regno 中。
write	当 transfer 被置位后，write 含义如下： 0：将数据从指定的寄存器复制到 data 的 arg0 部分。 1：将 data 中的 arg0 复制到指定寄存器中。
regno	要访问的寄存器号，如表 3.3 中所述。如果未挂起的 hart 支持此命令，则 dpc 可用作 PC 的别名。

3.6.1.2 快速访问

执行以下操作序列：

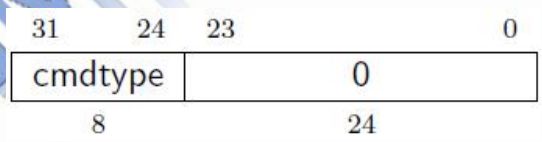
1.如果 hart 挂起，该命令会将 cmderr 设置为 4“halt/resume”，并且不会继续。

2.如果 hart 因其他原因（例如断点）而挂起，该命令会将 cmderr 设置为 4“halt/resume”，并且不会继续。

3.执行 Program Buffer，如果发生异常，则将 cmderr 设置为 3“exception”，并且程序缓冲区执行结束，但是快速访问命令继续。

4.恢复 hart。

此命令是可选的。该命令不触碰 data 寄存器。



名称	描述
cmdtype	为 1 表示快速访问命令。

3.6.1.3 存储器访问

使用此命令调试器可以执行存储器访问，该访问和所选 hart 执行的内存和权限访问完全相同。这包括对 hart 本地内存映射寄存器的访问，等等。该命令执行以下操作序列：

1.如果 write 清除，则将数据从 arg1 中指定的存储位置复制到 data 的 arg0 中。

2.如果 write 置位，则将数据从 data 的 arg0 复制到 arg1 中指定的存储位置。

3.如果设置了 aampostincrement，则增加 arg1。

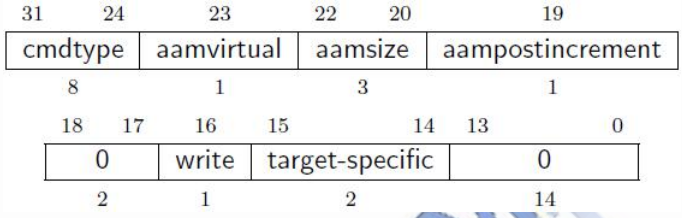
如果这些操作中的任何一个失败，则将设置 cmderr，并且不执行其余任何步

骤。仅当运行在 M 模式下 hart 的代码，尝试相同的访问才会失败。实现可能会及早发现即将发生的故障，并在导致故障的步骤之前使整个命令失败。

调试模块可以选择实现此命令，当选择的 hart 正在运行或停止时，可以支持对存储器位置的读写访问。如果此命令支持在运行 hart 时进行内存访问，它也必须支持在停止 hart 上进行内存访问。

选择 aamsize 的编码以匹配 sbcs 中的 sbaccess。

仅当读取内存时，此命令才会修改 arg0。仅在设置了 aampostincrement 后才修改 arg1。其他 data 寄存器不变。



名称	描述
cmdtype	为 2 表示存储器访问命令。
aamvirtual	不必同时实现虚拟访问和物理访问，但是必须使它不支持的访问失败。 0：地址是物理地址（对地址执行）。 1：地址是虚拟的，并按照设置 MRV 的方式从 M 模式转换为地址。
aamsize	~~0：访问存储器位置的最低 8 位。~~ ~~1：访问存储器位置的最低 16 位。~~ ~~2：访问存储器位置的最低 32 位。~~ ~~3：访问存储器位置的最低 64 位。~~ ~~4：访问存储器位置的最低 128 位。~~ 访问存储器抽象命令的参数宽度由 DXLEN 决定，而不由 aamsize 决定。（附录 C.1.2 更新）
aampostincrement	内存访问完成后，如果该位为 1，则将 arg1（包含使用的地址）增加 aamsize 中编码的字节数。
write	0：将数据从 arg1 中指定的存储位置复制到 data 的 arg0 部分。 1：将数据从 data 的 arg0 部分复制到 arg1 指定的存储位置。
target-specific	这些位保留用于目标特定用途。

3.7 程序缓冲区

为了支持在挂起的 hart 上执行任意指令，调试模块可以包括程序缓冲区，调试器可以将小程序写入其中。仅使用抽象命令支持所有必需功能的系统可以选择省略程序缓冲区。

调试器可以将一个小程序写入程序缓冲区，然后使用访问寄存器抽象命令将其执行一次，并在 command 中设置 postexec 位。调试器可以编写任何喜欢的程

序（包括从程序缓冲区跳出），但该程序必须以 ebreak 或 c.ebreak 结尾。实现可能支持隐式 ebreak，该隐式 ebreak 在 hart 运行到程序缓冲区的末尾时执行。这由 impebreak 表示。利用此功能，仅 2 个 32 位字的程序缓冲器就可以提供有效的调试。

如果 progbufsize 为 1，则 impebreak 必须为 1。程序缓冲区可能仅可容纳一条 32 位或 16 位指令，因此，在这种情况下，无论其大小如何，调试器必须只写一条指令。该指令可以是 32 位指令，也可以是低 16 位的压缩指令（并伴随高 16 位的 nop 压缩）。

与大小为 1 的程序缓冲区的行为略有不一致，是为了适应那些希望在挂起时将指令直接填充到管道中的硬件设计，而不是在某个地方的地址空间中存在程序缓冲区。

在执行这些程序时，hart 不会退出调试模式（请参见第 4.1 节）。如果在执行 Program Buffer 期间遇到异常，则将不再执行任何指令，hart 将保持在调试模式，并且 cmderr 设置为 3（异常错误）。如果调试器执行的程序没有以 ebreak 指令终止，则调试器将保持在调试模式，调试器将失去对 hart 的控制。

执行程序缓冲区 Program Buffer 可能会破坏 dpc。在这种情况下，可以使用未设置 postexec 的抽象命令来读取/写入 dpc。调试器必须在暂停和执行程序缓冲区之间保存 dpc，然后在退出“调试模式”之前还原 dpc。

允许将程序缓冲区执行去破坏 dpc，可以实现没有单独 PC 寄存器的设计，在执行程序缓冲区时确实需要使用 PC。

程序缓冲器可以被实现为可被 hart 访问的 RAM。如果是这种情况，调试器可以执行一个小程序，试图去读写正在执行的 program buffer 上的相对 pc。如果是这样，调试器在使用程序缓冲区时可以具有更大的灵活性。

3.8 总的状态机

图 3.1 显示了运行/暂停调试过程中 hart 传递的状态的概念图，该状态受 dmcontrol，abstractcs，abstractauto 和 command 的不同字段影响。

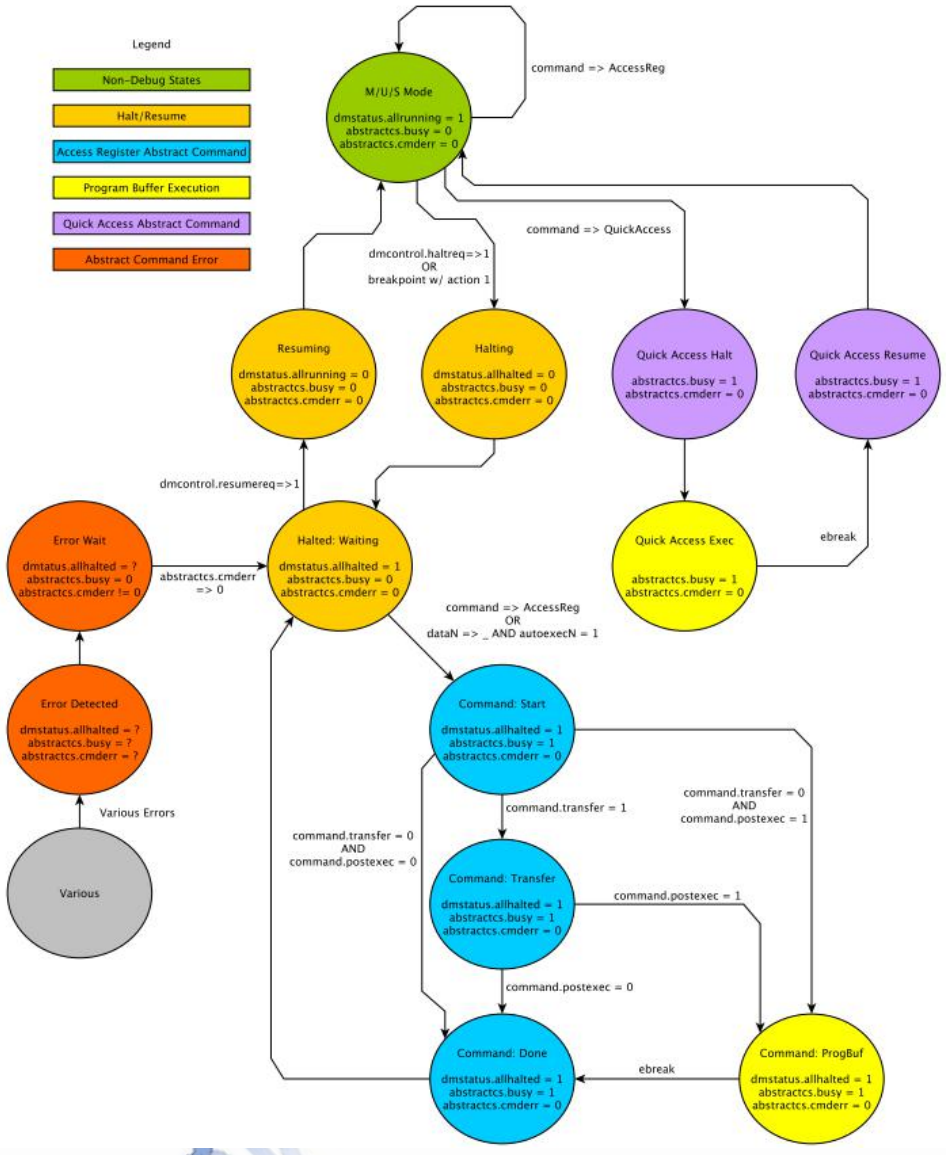


图 3.1：单 hart 系统的运行/暂停调试状态机。由于调试器仅能看到少量状态，因此状态和转换是概念性的。

3.9 系统总线访问

从 hart 的角度，调试器可以使用程序缓冲区或抽象访问内存命令访问内存（这两个功能都是可选的）。调试模块可能还包括一个系统总线访问块可提供内存访问，而不会涉及任何 hart，无论是否实现了程序缓冲区。系统总线访问块使用物理地址。

系统总线访问模块可以支持 8 位，16 位，32 位，64 位和 128 位访问。表 3.7 显示了 sbdata 中的哪些位用于每种访问大小。

Table 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

不同的微架构，通过系统总线存取访问的数据可能与每个 hart 观察到的数据并不总是一致的。如果实现不一致，则调试器将强制执行一致性。本规范未定义执行此操作的标准方法。可能包括写入特殊的内存映射位置，或通过程序缓冲区执行特殊的指令。

即使调试模块实现了程序缓冲区，实现系统总线访问块也有很多好处。首先，可以以最小的影响访问正在运行的系统中的内存。其次，它可以提高访问内存时的性能。此外，它可以对 hart 无法访问的设备进行访问。

3.10 最小程度的干扰调试

取决于执行的任务，某些 hart 只能非常短暂地挂起。有几种机制可以访问一个正在运行的系统中的资源，而对运行时的 hart 影响最小。

第一，可以允许某些抽象命令执行而不需要 hart 挂起。

第二，快速访问抽象命令可用于挂起 hart，快速执行程序缓冲区的内容让 hart 再次运行。如 3.12.3 所述，参考那些 Program Buffer 代码可以访问 data 寄存器的指令，这些指令可用于快速访问存储器或寄存器。对于某些系统，这些很麻烦，但是对于许多不能停下来系统来说，它是能承受这种偶尔的几百个甚至更少的 cycle 损耗。

第三，如果实现了系统总线访问块，则可以在运行的 hart 中使用它来访问系统内存。

3.11 安全

为了保护知识产权，可能需要锁定对调试模块的访问。只允许在制造过程中访问，之后不可以访问，合理的解决方案是在调试模块添加一个熔丝位，可用于永久禁用它。由于这是特定于技术的，因此在本规格书中将不作进一步处理。

另一种选择是，仅允许具有访问密钥的用户将 DM 解锁。可以通过 authenticated，authbusy 和 authdata，支持任意复杂的身份验证机制。当 authenticated 被清，DM 既不得与平台的其余部分进行交互，也不得公开连接到 DM 的 harts 详细信息。所有 DM 寄存器应读为 0，而写操作应被忽略，但以下强制性例外：

1.dmstatus 中的 authenticated 是可读的。

2.dmstatus 中的 authbusy 是可读的。

3.dmstatus 中的 version 是可读的。

4.dmcontrol 中的 dmactive 是可读可写的。

5.authdata 是可读可写的。

3.12 调试模块寄存器

通过 DMI 总线访问本节中描述的寄存器。每个 DM 都有一个基地址（第一个 DM 为 0）。下面的寄存器地址是基于该基地址的偏移量。

Table 3.8: Debug Module Debug Bus Registers

Address	Name
0x04	Abstract Data 0 (`data0`)
0x0f	Abstract Data 11 (`data11`)
0x10	Debug Module Control (`dmcontrol`)
0x11	Debug Module Status (`dmstatus`)
0x12	Hart Info (`hartinfo`)
0x13	Halt Summary 1 (`haltsum1`)
0x14	Hart Array Window Select (`hawindowsel`)
0x15	Hart Array Window (`hawindow`)
0x16	Abstract Control and Status (`abstractcs`)
0x17	Abstract Command (`command`)
0x18	Abstract Command Autoexec (`abstractauto`)
0x19	Configuration String Pointer 0 (`confstrptr0`)
0x1a	Configuration String Pointer 1 (`confstrptr1`)
0x1b	Configuration String Pointer 2 (`confstrptr2`)
0x1c	Configuration String Pointer 3 (`confstrptr3`)
0x1d	Next Debug Module (`nextdm`)
0x20	Program Buffer 0 (`progbuf0`)
0x2f	Program Buffer 15 (`progbuf15`)
0x30	Authentication Data (`authdata`)
0x34	Halt Summary 2 (`haltsum2`)
0x35	Halt Summary 3 (`haltsum3`)
0x37	System Bus Address 127:96 (`sbaddress3`)
0x38	System Bus Access Control and Status (`sbcs`)
0x39	System Bus Address 31:0 (`sbaddress0`)
0x3a	System Bus Address 63:32 (`sbaddress1`)
0x3b	System Bus Address 95:64 (`sbaddress2`)
0x3c	System Bus Data 31:0 (`sbdata0`)
0x3d	System Bus Data 63:32 (`sbdata1`)
0x3e	System Bus Data 95:64 (`sbdata2`)
0x3f	System Bus Data 127:96 (`sbdata3`)
0x40	Halt Summary 0 (`haltsum0`)

读取时，未实现的调试模块 DMI 寄存器返回 0。写入它们无效。

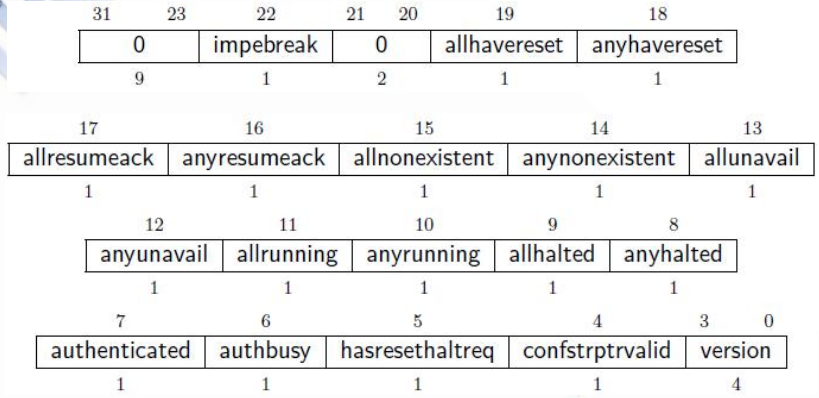
对于每个寄存器来说，可以通过两种方式确定其是否实现：

（1）通过读取它并获取一个非零值（例如 sbcs）；

（2）通过检查另一个寄存器中的位（例如 progbufsize）。

3.12.1 Debug Module Status (dmstatus, at 0x11)

该寄存器声明了整个调试模块的状态，通过 hasel 定义了当前被选中的 hart，因为它包含版本，所以它的地址将来不会更改。整个寄存器是只读的。



名称	描述	访问类型	Reset 值
impebreak	如果为 1，则在程序缓冲区之后，在不存在的单词中有一个隐式的 ebreak 指令。这使调试器不必编写 ebreak 本身，并使程序缓冲区小了一个字。 当 progbufsize 为 1 时，该位必须为 1。	R	preset
allhavereset	当所有当前选定的 hart 都已重置，且其中任何一个 hart 未被确认重置时，此字段为 1。（重置、确认组合）	R	-
anyhavereset	当至少有一个当前选择的 hart 已被重置，且该 hart 的重置未被确认时，此字段为 1。	R	-
allresumeack	当所有当前选定的 hart 都已确认其最后的恢复请求时，此字段为 1。	R	-
anyresumeack	当任何当前选定的 hart 确认了其最后的恢复请求时，此字段为 1。	R	-
allnonexistent	当前所有选择的 hart 不在此平台中时，该字段为 1。	R	-
anynonexistent	当任何一个被选定的 hart 不存在此平台中，此字段为 1。	R	-
allunavail	当前所有被选定的 hart 都不可用时，此字段为 1。	R	-
anyunavail	当前任何选定的 hart 为不可用，此字段为 1。	R	-
allrunning	当前所有选中的 harts 正在运行，此字段为 1。	R	-
anyrunning	当前任何选定的 hart 正在运行，此字段为 1。	R	-
allhalted	当前所有选中的 hart 停止时，此字段为 1。	R	-
anyhalted	当前任何选定的 hart 停止时，此字段为 1。	R	-
authenticated	0：使用 DM 之前需要认证。 1：认证通过。 在未实现身份验证的组件上，此位必须预设为 1。	R	preset
authbusy	0：认证模块已准备就绪，可以处理对 authdata 的下一次读/写。 1：认证模块忙，访问 authdata 会导致未指定的行为。 在对 authdata 的访问立即响应时才设置 authbusy。	R	0
hasresethaltreq	如果此调试模块可由 setresethaltreq 和 clrresethaltreq 位控制的复位暂停功能 halt-on-reset，则为 1。否则为 0。	R	preset
confstrptrvalid	0：confstrptr0-confstrptr3 保留与配置字符串无关的信息。	R	preset

	1：confstrptr0-confstrptr3 保存配置字符串的地址。		
version	0：不存在调试模块。 1：有一个调试模块，它符合此规范的 0.11 版。 2：有一个调试模块，它符合此规范的 0.13 版。 15：有一个调试模块，但它不符合该规范的任何可用版本。	R	2

3.12.2 Debug Module Control (dmcontrol, at 0x10)

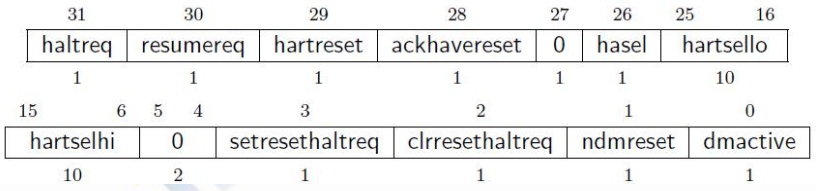
该寄存器控制整个调试模块，且在 hasel 中定义了当前选择的 harts。

在整个文档中，我们所说的 hartsel，是 hartselhi 与 hartsello 的组合。而 spec 允许使用 20 个 hartsel 位，实际实现可以少于该数量。hartsel 的实际宽度称为 HARTSELLEN，该值至少为 0 且至多为 20。调试程序通过写全 1 到 hartsel（假定最大 size），并读回来发现 HARTSELLEN 返回值以查看实际设置的位。在执行抽象命令时，调试器不得更改 hartsel。

有单独的 setresethaltreq 和 clrresethaltreq 位时，并非所有选定的 hart 都具有相同的配置时，每个选定的 hart 可以写入 dmcontrol 而不更改 hart-on-reset 复位停止请求位。

在任何给定的写入中，调试器最多只能向以下一位写入 1：resumereq，hartreset，ackhavereset，setresethaltreq 和 clrresethaltreq。其他必须写为 0。

resethaltreq 是每个 HART 状态的可选内部位，不能被读取，但可以使用 setresethaltreq 和 clrresethaltreq 写入。



名称	描述	访问类型	Reset 值
haltreq	写入 0 将清除所有当前选中的 hart 的停止请求位。这可能会取消那些未完成的暂停请求的 hart。 写入 1 将设置所有当前选中的 hart 的停止请求位。正在运行的 hart 会停止，不管他们的暂停请求位是否被置位。 写入应用于 hartsel 和 hasel 的新值。	W	-
resumereq	如果在写入发生时将其暂停，则写入 1 会使当前选定的 hart 恢复一次，它还清除了该 hart 的 resume ack。 如果设置了 haltreq，则会忽略 resumereq。 写入应用于 hartsel 和 hasel 的新值。	W1	-
hartreset	该可选字段为所有当前选中的 hart 写入 reset 位。要执行复位，调试器将写入 1，然后写 0 解除复位信号。 当该位为 1 时，调试器不得更改所选的 harts。 如果未实现此功能，则该位始终保持为 0，因此在写入 1 之后，调试器可以读回寄存器以查看是否支持该功能。	R/W	0

	写入应用于 hartsel 和 hasel 的新值。		
ackhavereset	0：不生效 1：对任意选定的 hart，清除 havereset 位。 写入应用于 hartsel 和 hasel 的新值。	W1	-
hasel	选择当前选定的 hart 的定义。 0：只有一个当前选定的 hart，由 hartsel 选择。 1：可能有多个当前选择的 hart，由 hartsel 选择的 hart，以及由 hart mask 寄存器选择的 hart。 其中没有实现 hart mask 寄存器，该字段为 0。 希望使用 hart mask 寄存器功能的调试器应将该位置 1，然后回读以查看是否支持该功能。	R/W	0
hartsello	hartsel 的低 10 位：要选择的 hart 的 DM 特定索引。该 hart 始终是当前选定的 hart 的一部分。	R/W	0
hartselhi	hartsel 的高 10 位：要选择的 hart 的 DM 特定索引。该 hart 始终是当前选定的 hart 的一部分。	R/W	0
setresethaltreq	除非将 clrresethaltreq 同时设置为 1，否则此可选字段，为当前所有选定的 harts 写入“halt-on-reset”请求位。设置为 1 时，每个选定的 hart 将在其下一次解除 reset 时挂起。复位暂停请求位不会自动清除，调试器必须写入 clrresethaltreq 以清除它。 写入应用于 hartsel 和 hasel 的新值。 如果 hasresethaltreq 为 0，则不会实现此字段。	W1	-
clrresethaltreq	该可选字段清除当前所有选定的 hart 的复位时停止请求位。 写入应用于 hartsel 和 hasel 的新值。	W1	-
ndmreset	该位控制从 DM 到系统其余部分的复位信号。信号应重置系统的每个部分，包括每个 hart，除了 DM 和任何需要访问 DM 逻辑的。要执行系统重置，调试器将写入 1，然后写入 0 以使重置无效。	R/W	0
dmactive	该位用作调试模块本身的复位信号。 0：模块的状态（包括身份验证机制）采用其重置值（dmactive 位是唯一可以写入其重置值以外的内容的位）。 1：模块正常运行。 不存在其他可能导致上电后重置调试模块的机制，可能（但不建议）的是全局重置信号（用于重置整个平台）的例外。 调试器可以将此位脉冲为低电平，以使调试模块进入已知状态。 实施中可能会注意此位，以进一步帮助调试，例如，当调试处于活动状态时防止对调试模块进行功率门控。	R/W	0

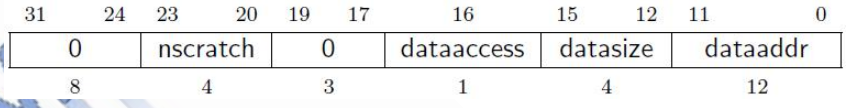
3.12.3 Hart Info (hartinfo, at 0x12)

该寄存器提供有关 hartsel 当前选择的 hart 的信息。

该寄存器是可选的。如果不存在，则应读取全零。

如果包含此寄存器，则调试器可以编写显式访问 data 或 dscratch 寄存器的程序，来对 Program Buffer 进行更多的处理。

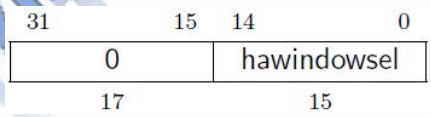
整个寄存器是只读的。



名称	描述	访问类型	Reset 值
nscratch	从 dscratch0 开始，可供调试器在程序缓冲区执行期间使用的 dscratch 寄存器数。调试器不能假设这些寄存器在命令之间的内容。	R	Preset
dataaccess	0：data 寄存器通过 CSR 被隐藏在 hart 中。根据表 3.1，每个 CSR 的大小为 DXLEN 位，并且对应于一个参数。 1：data 寄存器在 hart 的存储器映射中被隐藏。每个寄存器在内存映射中占用 4 个字节。	R	Preset
datasize	如果 dataaccess 为 0：专用于 data 寄存器的 CSR 数量。 如果 dataaccess 为 1：内存映射中专用于 data 寄存器的 32 位字的数量。 由于最多有 12 个 data 寄存器，因此该寄存器中的值必须为 12 或更小。	R	Preset
dataaddr	如果 dataaccess 为 0：专用于 data 寄存器的第一个 CSR 的编号。 如果 dataaccess 为 1：data 寄存器的 RAM 的有符号地址，用于相对于 zero 的访问。	R	Preset

3.12.4 Hart Array Window Select (hawindowsel, at 0x14)

该寄存器选择 hart 阵列掩码寄存器的 32 位部分中的一部分（见 3.3.2 节），该寄存器可在 hawindow 中访问。



名称	描述	访问类型	Reset 值
hawindowsel	此字段的高位可能绑定到 0，具体取决于阵列掩码寄存器的大小。例如，在一个有 48 个 hart 的系统上，仅此位 0 字段实际上可能是可写的。	R/W	0

3.12.5 Hart Array Window (hawindow, at 0x15)

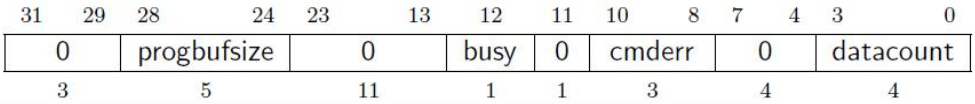
该寄存器提供对 Hart 阵列屏蔽寄存器的 32 位部分的 R/W 访问（请参见第 3.3.2 节）。窗口的位置由 hawindowsel 确定。即，位 0 指 hart hawindowsel*32，而位 31 指 hart hawindowsel*32+31。

由于 hart 数组掩码寄存器中的某些位可能为常数 0，因此此寄存器中的某些位可能为常数 0，具体取决于 hawindowsel 的当前值。

3.12.6 Abstract Control and Status (abstractcs, at 0x16)

执行抽象命令时，如果 cmderr 为0，写入该寄存器会使 cmderr 设置为1（busy）。

datacount 必须至少为 1 以支持 RV32 harts，2 为支持 RV64 harts 或 4 为支持 RV128 harts。



名称	描述	访问类型	Reset 值
progbufsize	程序缓冲区的大小，以 32 位字为单位。有效大小为 0-16。	R	Preset
busy	1：当前正在执行抽象命令。 写入 command 后立即设置该位，直到该命令完成后才清除。	R	0
cmderr	如果抽象命令失败，则获取设置。该字段中的位将保持置位状态，直到写入 1 将其清除为止。在该位重置为 0 之前，不会启动任何抽象命令。 仅 busy 为 0 时，此字段包含有效值。 0（none）：无错误。 1（busy）：在写入 command，abstractcs 或 abstractauto 时，或者在读取或写入 data 或 progbuf 寄存器之一时，正在执行抽象命令。仅当 cmderr 为 0 时，才写入此状态。 2（not supported）：无论 hart 是否在运行，都不支持所请求的命令。 3（exception）：执行命令时（例如，执行程序缓冲区时）发生异常。 4（halt/resume）：由于 hart 不在所需的状态（running/halted）或不可用，因此无法执行抽象命令。 5（bus）：由于总线错误（例如，对齐，访问大小或超时），抽象命令失败。 7（other）：命令由于其他原因失败。	R/W1C	0
datacount	作为抽象命令界面的一部分，实现的 data 寄存器的数量。有效大小为 1-12。	R	Preset

3.12.7 Abstract Command (command, at 0x17)

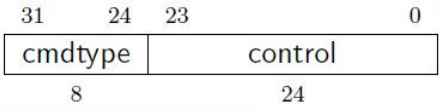
写入该寄存器将执行相应的抽象命令。

执行抽象命令时，如果 cmderr 为0，写入该寄存器会使 cmderr 设置为1（busy）。

如果 cmderr 不为零，则忽略对该寄存器的写操作。

由于性能原因，cmderr 禁止启动新命令来容纳调试器，这些调试器出于性能原因而发送多个要连续执行的命令，而无需检查它们之间的 cmderr。他们可以放心地这样做，并在最后检查 cmderr，而不必担心一个命令失败了，但随后又通过

了一个命令（可能取决于前一个命令）。

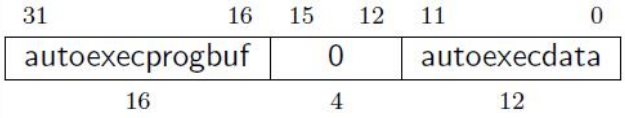


名称	描述	访问类型	Reset 值
cmdtype	确定此抽象命令的整体功能。	W	0
control	以特定于命令的方式解释此字段，为每个抽象命令进行描述。	W	0

3.12.8 Abstract Command Autoexec (abstractauto, at 0x18)

该寄存器是可选的。实现它将允许更有效的 burst 突发访问。调试器可以通过写该位并将其读回来检测是否支持该功能。

执行抽象命令时，如果 cmderr 为 0，写入该寄存器会使 cmderr 设置为 1（busy）。



名称	描述	访问类型	Reset 值
autoexecprogbuf	当该字段的某个位为 1 时，对相应的 progbuf 字的读或写访问将导致再次执行 command 中的命令。	R/W	0
autoexecdata	当该字段的某个位为 1 时，对相应 data 字的读或写访问将导致再次执行 command 中的命令。	R/W	0

3.12.9 Conguration String Pointer 0 (confstrptr0, at 0x19)

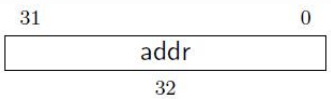
当 confstrptrvalid 有效时，读取该寄存器将返回配置字符串指针的 31:0 位。读取其他 confstrptr 寄存器将返回地址的高位。

实施系统总线主控后，该地址是可与系统总线访问模块一起使用的地址。否则，在 ID 为 0 的 hart 上，该地址可用于访问配置字符串。

如果 confstrptrvalid 为 0，则 confstrptr 寄存器保存的标识信息在本文档中未作进一步规定。

配置字符串本身在**特权规范**中进行了描述。

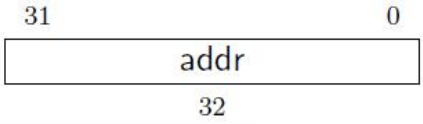
整个寄存器是只读的。



3.12.10 Next Debug Module (nextdm, at 0x1d)

如果 DMI 上可访问的 DM 超过一个，则该寄存器包含链中下一个 DM 的基地址；如果该地址是链中的最后一个，则该寄存器为 0。

整个寄存器是只读的。



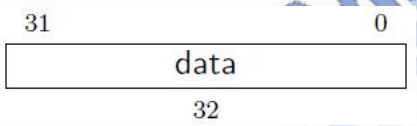
3. 12. 11 Abstract Data 0 (data0, at 0x04)

data0 到 data11 是读/写寄存器，可以由抽象命令读取或更改。datacount 表示已实现的数量，从 data0 开始，递增计数。表 3.1 显示了抽象命令如何使用这些寄存器。

执行抽象命令时，如果 cmderr 为 0，写入该寄存器会使 cmderr 设置为 1（busy）。

当 busy 时，写它们不会更改它们的值。

执行抽象命令后，可能不会保留这些寄存器中的值。对其内容的唯一保证是有关命令所提供的保证。如果命令失败，则不能对这些寄存器的内容做任何假设。

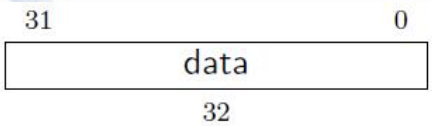


3. 12. 12 Program Buffer 0 (progbuf0, at 0x20)

progbuf0 到 progbuf15 时可选的，提供对程序缓冲区的读/写访问。progbufsize 指示从 progbuf0 开始实现的数量（递增计数）。

执行抽象命令时，如果 cmderr 为 0，写入该寄存器会使 cmderr 设置为 1（busy）。

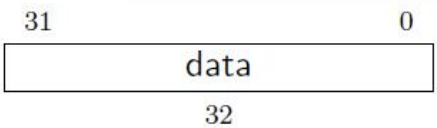
当 busy 时，写它们不会更改它们的值。



3. 12. 13 Authentication Data (authdata, at 0x30)

该寄存器用作往返于身份验证模块的 32 位串行端口。

当 authbusy 被清后，调试器可以通过读取或写入该寄存器来与身份验证模块进行通信。没有单独的机制来指示上溢/下溢。

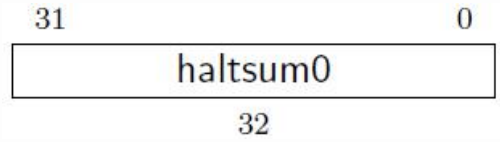


3.12.14 Halt Summary 0 (haltsum0, at 0x40)

该只读寄存器中的每一位都指示是否挂起了某个特定 hart。不可用/不存在的 hart 不被视为已挂起。

LSB 反映了 hart{hartsel[19:5]， 5'h0}的挂起状态，而 MSB 反映了 hart{hartsel[19:5]，5'h1f}的挂起状态。

整个寄存器是只读的。



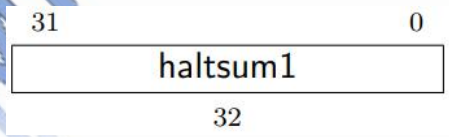
3.12.15 Halt Summary 1 (haltsum1, at 0x13)

该只读寄存器中的每一位指示是否挂起了一组 hart。不可用/不存在的 hart 不被视为已挂起。

该寄存器在少于 33 个 harts 的系统中可能不存在。

LSB 反映了 hart{hartsel[19:10]，10'h0}到{hartsel[19:10]，10'h1f}的挂起状态。MSB 反映了 hart{hartsel[19:10]，10'h3e0}到{hartsel[19:10]，10'h3ff}的挂起状态。

整个寄存器是只读的。



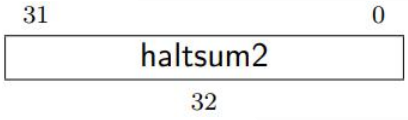
3.12.16 Halt Summary 2 (haltsum2, at 0x34)

该只读寄存器中的每一位指示是否挂起了一组 hart。不可用/不存在的 hart 不被视为已挂起。

在少于 1025 个 Harts 的系统中，可能不存在此寄存器。

LSB 反映了 hart{hartsel[19:15]，15'h0}到{hartsel[19:15]，15'h3ff}的挂起状态。MSB 反映了{hartsel[19:15]，15'h7c00}到{hartsel[19:15]，15'h7fff}的挂起状态。

整个寄存器是只读的。



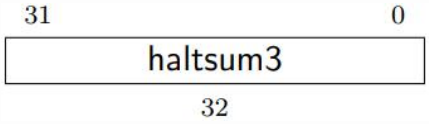
3. 12. 17 Halt Summary 3 (haltsum3, at 0x35)

该只读寄存器中的每一位指示是否挂起了一组 hart。不可用/不存在的 hart 不被视为已挂起。

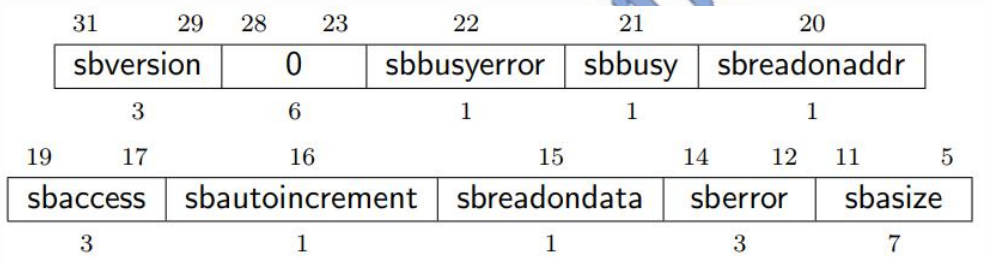
在少于 32769 个 Harts 的系统中，可能不存在此寄存器。

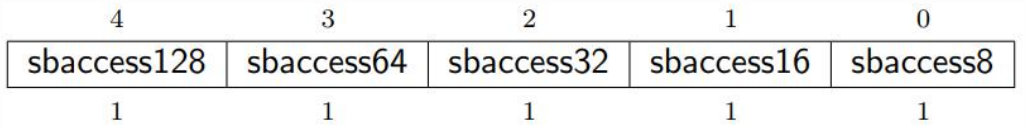
LSB 反映了 hart 20'h0 到 20'7fff 的挂起状态。MSB 反映了 hart 20'hf8000 到 20'hfffff 的挂起状态。

整个寄存器是只读的。



3. 12. 18 System Bus Access Control and Status (sbcs, at 0x38)





名称	描述	访问类型	Reset 值
sbversion	0：系统总线接口符合 2018.1.1 之前主线草稿的规范。 1：系统总线接口符合此版本的规范。 其他值保留用于将来的版本。	R	1
sbbusyerror	当调试器在进行读时尝试读取数据，或者调试器在进行新访问（已设置 sbbusy 时）时发起一个新访问时，设置该 bit。它会一直保持设置状态，直到调试器将其清除为止。 设置此字段后，调试模块将无法启动更多系统总线访问。	R/W1C	0
sbbusy	当为 1 时，表明系统总线主设备正忙。（与系统总线本身是否正忙相关，但不相同。）当发生任何读或写的请求时，该位立即变高，直到访问完全完成才会变低。 当 sbbusy 为高时，写入 sbcs 会导致未定义的行为。调试器在 sbbusy 为 0 之前，不得写入 sbcs。	R	0
sbreadonaddr	为 1 时，每次写入 sbaddress0 都会自动触发在系统总线新地址处的读操作。	R/W	0
sbaccess	选择用于系统总线访问的访问大小。 0：8 位	R/W	2

	1：16 位 2：32 位 3：64 位 4：128 位 如果在 DM 启动总线访问时，sbaccess 的值不被支持，则不执行访问，并且 sberror 设置为 4。		
sbautoincrement	设置为 1 时，每次系统总线访问后，sbaddress 都会增加 sbaccess 中选择的访问大小（以字节为单位）。	R/W	0
sbreadondata	为 1 时，对 sbdata0 的每次读取都会自动触发在（可能是自动递增的）系统总线地址上的读操作。	R/W	0
sberror	调试模块的系统总线主设备遇到错误时，将设置此字段。该字段中的位将保持置位状态，直到写入 1 将其清除为止。当此字段为非零值时，调试模块将无法启动更多系统总线访问。 一个实现可以针对任何错误情况报告“其他”（7）。 0：没有总线错误。 1：有一个超时。 2：访问了错误的地址。 3：出现对齐错误。 4：请求访问的大小不支持。 7：其他	R/W1C	0
sbasize	系统总线地址的宽度（以位为单位）。（0 表示不支持总线访问。）	R	Preset
sbaccess128	当支持 128 位系统总线访问时为 1。	R	Preset
sbaccess64	当支持 64 位系统总线访问时为 1。	R	Preset
sbaccess32	当支持 32 位系统总线访问时为 1。	R	Preset
sbaccess16	当支持 16 位系统总线访问时为 1。	R	Preset
sbaccess8	当支持 8 位系统总线访问时为 1。	R	Preset

3. 12. 19 System Bus Address 31:0 (sbaddress0, at 0x39)

如果 sbasize 为 0，则该寄存器不存在。

当系统总线主机忙时，对该寄存器进行写操作将设置 sbbusyerror，并且不执行其他任何操作。

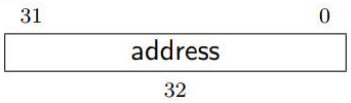
如果 sberror 为 0，sbbusyerror 为 0，并且设置了 sbreadonaddr，然后对该寄存器进行写操作，将开始以下操作：

1.设置 sbbusy。

2.通过 sbaddress 的值执行总线读取。

3.如果读取成功并且设置了 sbautoincrement，则增加 sbaddress。

4.清除 sbbusy 状态。



名称	描述	访问类型	Reset 值
address	访问 sbaddress 中的物理地址的 31:0 位。	R/W	0

3. 12. 20 System Bus Address 63:32 (sbaddress1, at 0x3a)

如果 sbasize 小于 33，则该寄存器不存在。

当系统总线主机忙时，对该寄存器进行写操作将设置 sbbusyerror，并且不执行其他任何操作。

名称	描述	访问类型	Reset 值
address	访问 sbaddress 中的物理地址的 63:32 位（如果系统地址总线有那么宽）。	R/W	0

3. 12. 21 System Bus Address 95:64 (sbaddress2, at 0x3b)

如果 sbasize 小于 65，则该寄存器不存在。

当系统总线主机忙时，对该寄存器进行写操作将设置 sbbusyerror，并且不执行其他任何操作。

名称	描述	访问类型	Reset 值
address	访问 sbaddress 中的物理地址的 95:64 位（如果系统地址总线有那么宽）。	R/W	0

3. 12. 22 System Bus Address 127:96 (sbaddress3, at 0x37)

如果 sbasize 小于 97，则该寄存器不存在。

当系统总线主机忙时，对该寄存器进行写操作将设置 sbbusyerror，并且不执行其他任何操作。

名称	描述	访问类型	Reset 值
address	访问 sbaddress 中的物理地址的 127:96 位（如果系统地址总线有那么宽）。	R/W	0

3. 12. 23 System Bus Data 31:0 (sbdata0, at 0x3c)

如果 sbcs 中的所有 sbaccess 位均为 0，则该寄存器不存在。

任何的系统总线读取成功后，都会更新 sbdata。如果读取访问的宽度小于 sbdata 的宽度，则其余高位的内容可以取任何值。

如果 sberror 或 sbbusyerror 都不都是 0，则访问不执行任何操作；如果总线忙，置位 sbbusyerror，并且不执行其他任何操作。**即**：只要 sbbusyerror 为 1，则不做任何操作，和 sberror 没有太大关系。

写入该寄存器将开始以下操作：

1.设置 sbbusy。

2.执行总线写操作，将 sbdata 的新值写入 sbaddress。

3.如果写入成功并且设置了 sbautoincrement，则增加 sbaddress。

4.清除 sbbusy 状态。

从该寄存器中读取执行以下内容：

1.“返回”数据。

2.设置 sbbusy。

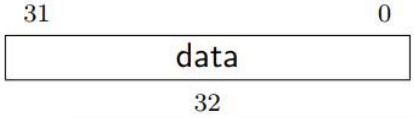
~~3.如果设置了 sbreadondata，请执行系统总线从 sbaddress 中包含的地址读取，将结果放在 sbdata 中。（附录 C.1.3 更新）~~

3.如果设置了 sbreadondata，再次执行系统总线读取。

4.如果设置了 sbautoincrement，则增加 sbaddress。

5.清除 sbbusy 状态。

仅 sbdata0 具有此行为，其他 sbdata 寄存器没有这些副作用。在总线宽度超过 32 位的系统上，调试器应在访问其他 sbdata 之后访问 sbdata0 寄存器。

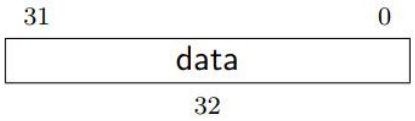


名称	描述	访问类型	Reset 值
data	访问 sbdata 中的 31:0 位	R/W	0

3.12.24 System Bus Data 63:32 (sbdata1, at 0x3d)

如果 sbaccess64 和 sbaccess128 为 0，则该寄存器不存在。

如果总线忙，则访问置位 sbbusyerror，并且不执行其他任何操作。

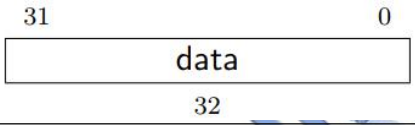


名称	描述	访问类型	Reset 值
data	访问 sbdata 中的 63:32 位（如果系统地址总线有那么宽）。	R/W	0

3.12.25 System Bus Data 95:64 (sbdata2, at 0x3e)

仅当 sbaccess128 为 1 时，此寄存器才存在。

如果总线忙，则访问置位 sbbusyerror，并且不执行其他任何操作。

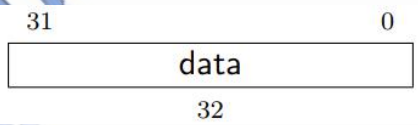


名称	描述	访问类型	Reset 值
data	访问 sbdata 中的 95:64 位（如果系统地址总线有那么宽）。	R/W	0

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

仅当 sbaccess128 为 1 时，此寄存器才存在。

如果总线忙，则访问置位 sbbusyerror，并且不执行其他任何操作。



名称	描述	访问类型	Reset 值
data	访问 sbdata 中的 127:96 位（如果系统地址总线有那么宽）。	R/W	0

4 RISC-V 调试

对 RISC-V 内核进行最小幅度修改，去支持调试。有一个特殊的执行模式（调试模式）和一些额外的 CSR。DM 负责其余的工作。

为了符合本规范，必须实现本节中未明确列出为可选内容的所有内容

4.1 调试模式

调试模式是一种特殊的处理器模式，仅在 hart 挂起进行外部调试时使用。此处未指定如何实现调试模式。

当从可选的程序缓冲区执行代码时，hart 保持在调试模式，并且遵循以下条件：

1.除了根据 mprven 可以忽略 mstatus 中的 MPRV 之外，所有操作都以机器模式特权级别执行。

2.所有中断（包括 NMI）均被屏蔽。

3.例外不会更新任何寄存器。其中包括 cause，epc，tval，dpc 和 mstatus。但它们会结束程序缓冲区的执行。

4.如果触发器匹配，则不执行任何操作。

5.计数器可能会停止，具体取决于 dcsr 中的 stopcount。

6.计时器可能会停止，具体取决于 dcsr 中的 stoptime。

7.wfi 指令充当 nop。

8.几乎所有更改特权级别的指令都具有未定义的行为。这包括 ecall，mret，sret 和 uret。（要更改特权级别，调试器可以在 dcsr 中写入 prv）。唯一的例外是 ebreak。在调试模式下执行该操作时，它会再次挂起 hart，但不会更新 dpc 或 dcsr。

9.完成程序缓冲区执行被视为输出，以用于 fence 指令。

10.如果所有控制传递指令的目的地在程序缓冲区中，则它们可能会充当非法指令。如果其中一条指令为非法指令，则所有此类指令必须为非法指令。

11.如果所有控制传递指令的目的地在程序缓冲区之外，则它们可能会充当非法指令。如果其中一条指令为非法指令，则所有这些指令必须作为非法指令。

12.更改 PC 值的指令（例如 auipc）可能会构成非法指令。

13.有效的 XLEN 是 DXLEN。

通常，调试器有望能够模拟 MPRV 的所有效果。Sv32 系统例外，它需要 MPRV 功能才

能访问 34 位物理地址。其他系统可能会将 mprven 设置为 0。

4.2 Load-Reserved/Store-Conditional 指令

当进入调试模式或在调试模式下时，lr 指令在内存地址上注册的保留可能会丢失。这意味着，如果在 lr 和 sc 对之间进入“调试模式”，则可能没有前进的进度。

这是调试用户必须了解的行为。如果它们在 lr 和 sc 对之间设置了断点，或者正在逐步执行此类代码，则 sc 可能永远不会成功。幸运的是，在一般情况下，按这样的顺序执行的指令很少，任何调试它的人都会很快注意到没有发生 reservation。在这种情况下，解决方案是在 sc 之后的第一条指令上设置一个断点并运行它。更高级别的调试器可以选择自动执行此操作。

4.3 WFI 指令

WFI 是 Wait for Interrupt 的缩写。

如果在执行 wfi 的过程中要求挂起，则 hart 必须离开 stalled 状态，完成该指令的执行，然后进入调试模式。

4.4 Single Step

调试器可以使挂起的 hart 执行一条指令，然后在设置 resumereq 之前通过设置 step 重新进入调试模式。

如果执行指令或取指令导致异常，则将 PC，tval 和 cause 更新之后（PC 更改为异常入口的 pc），立即重新进入调试模式。

如果执行指令或取指令导致 trigger 触发，则在触发条件触发后立即重新进入调试模式。在这种情况下，cause 设置为 2（trigger），而不是 4（single step）。指令是否执行取决于触发器的具体配置。

如果执行的指令导致异常，且该 PC 更改为取指令的地址，则该异常直到下一次 hart 恢复时才会发生。同样，在新的地址尝试执行该指令之前，触发器不会在新地址触发。

如果要跳过的指令是 wfi，通常会使 hart 挂起，然后将该指令视为 nop。

4.5 复位

如果 halt 信号（由调试模块中的 hart 的停止请求位驱动）或 resethaltreq 在复位后完成被发现，则该 hart 必须在完成硬件的初始化后、执行第一条指令前进入 debug 模式。

4.6 dret 指令

要从调试模式返回，定义了一条新指令：dret，它的编码为 0x7b200073。在支持此指令的 harts 上，在调试模式下执行 dret 指令，会将 pc 值更改为存储在 dpc 中的值。当前特权级别更改为 dcsr 中 prv 指定的特权级别。Hart 退出调试模式。

在调试模式之外执行 dret 会导致非法指令异常。

调试器不必知道实现是否支持 dret，因为调试模块将确保在必要时执行该指令。在本规范中仅定义为保留操作码，并允许可重用的调试模块实现。

4.7 XLEN

在调试模式下，XLEN 是 DXLEN。由调试器决定在正常程序执行过程中的 XLEN（通过查看 misa），并将其清楚地传达给用户。

4.8 核心调试寄存器

必须为每个可调试的 HART 实现核心调试寄存器。它们是 CSR，可使用 RISC-V csr 操作码以及可选的抽象调试命令进行访问。

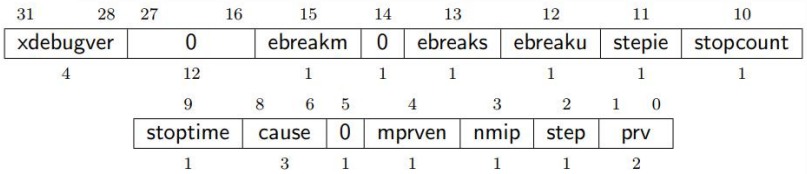
这些寄存器只能在调试模式下访问。

Table 4.1: Core Debug Registers

Address	Name
0x7b0	Debug Control and Status (dcsr)
0x7b1	Debug PC (dpc)
0x7b2	Debug Scratch Register 0 (dscratch0)
0x7b3	Debug Scratch Register 1 (dscratch1)

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

cause 分配优先级，以使最难以预测的事件具有最高优先级。



名称	描述	访问类型	Reset 值
xdebugver	0：不支持外部调试。 4：存在外部调试，如本文档中所述。 15：有外部调试支持，但不符合该规范的任何可用版本。	R	Preset
ebreakm	0：M 模式下的 ebreak 指令的行为如 Privileged Spec 中所述。 1：M 模式下的 ebreak 指令进入调试模式。	R/W	0

ebreaks	0：S 模式下的 ebreak 指令的行为如 Privileged Spec 中所述。 1：S 模式下的 ebreak 指令进入调试模式。	R/W	0
ebreaku	0：U 模式下的 ebreak 指令的行为如 Privileged Spec 中所述。 1：U 模式下的 ebreak 指令进入调试模式。	R/W	0
stepie	0：单步执行中禁止中断。 1：单步执行中允许中断。 可以将该位硬连接为 0。在这种情况下，调试器可以模拟中断行为。 当 hart 运行时，调试器不得更改该位的值。	WARL	0
stopcount	0：与往常一样递增计数器。 1：在调试模式下或执行进入调试模式的 ebreak 指令时，请勿增加任何计数器。这些计数器包括 cycle CSR 和 instret CSR。对于大多数调试方案而言，这是首选的。 一个实现可以将该位硬连接为 0 或 1。	WARL	Preset
stoptime	0：与往常一样递增计时器。 1：在调试模式下，请勿增加任何本地 hart 计时器。 一个实现可以将该位硬连接为 0 或 1。	WARL	Preset
cause	说明进入调试模式的原因。 如果在一个周期内，进入调试模式有多个原因，则硬件应将 cause 设置为具有最高优先级的原因。 1：执行了 ebreak 指令。（优先级 3） 2：触发模块导致断点异常。（优先级 4，最高） 3：调试器使用 haltreq 请求进入调试模式。（优先级 1） 4：因为设置了 step，所以 hart 单步调试。（优先级 0，最低） 5：由于 resethaltreq，hart 直接停止退出复位。当发生这种情况时，也可以报告 3。（优先级 2） 其他值保留供将来使用。	R	0
mprven	0：在调试模式下，mstatus 中的 MPRV 被忽略。 1：mstatus 中的 MPRV 在调试模式下生效。 实现该位是可选的。它可以连接到 0 或 1。	WARL	Preset
nmip	设置后，hart 将有一个不可屏蔽中断（NMI）挂起。 由于 NMI 可以指示硬件错误情况，因此一旦该位置 1，调试就可能不再可靠。这是依赖于实现的。	R	0
step	设置后且不在调试模式下，则 hart 只执行一条指令，然后进入调试模式。如果指令由于异常而未能完成，则设置适当的异常寄存器，在准备进入陷阱处理程序之前，hart 将立即进入调试模式。 当 hart 运行时，调试器不得更改该位的值。	R/W	0
prv	进入调试模式前 hart 正在操作的特权级别。编码在表 4.5 中描述。 退出调试模式时，调试器可以更改此值以更改 hart 的特权级别。 并非所有的 harts 都支持所有特权级别。如果不支持所编写的编码，或者不允许调试器对其进行更改，则 hart 可能会更改为任何受支持的特权级别。	R/W	3

4.8.2 Debug PC (dpc, at 0x7b1)

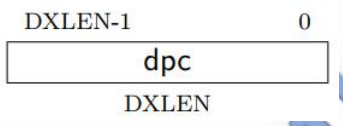
进入调试模式后，将使用下一条要执行的指令的虚拟地址更新 dpc。该行为

在表 4.3 中有更详细的描述。

表 4.3：进入调试模式时 DPC 中的虚拟地址

Cause	DPC 中的虚拟地址
ebreak	ebreak 指令的地址
single step	如果没有调试发生，保存将被执行的下一个指令的地址。对于不更改程序流程的 32 位指令就是 pc + 4，跳转/分支发生则保存分支目标地址的 pc。
trigger module	如果 timing 为 0，则保存导致触发器触发的指令的地址。 如果 timing 为 1，则在进入调试模式时，要保存执行的下一条指令的地址。
halt request	在进入调试模式时，要保存执行的下一条指令的地址

Resuming 后，hart 的 PC 将更新为 dpc 中存储的虚拟地址。调试器可以编写 dpc 来更改 hart 的恢复位置。



4.8.3 Debug Scratch Register 0 (dscratch0, at 0x7b2)

如果有需要，则可以实现该可选的 scratch 寄存器。除非 hartinfo 明确提及它，否则调试器不得写入该寄存器（调试模块可以在内部使用该寄存器）。

4.8.4 Debug Scratch Register 1 (dscratch1, at 0x7b3)

如果有需要，则可以实现该可选的 scratch 寄存器。除非 hartinfo 明确提及它，否则调试器不得写入该寄存器（调试模块可以在内部使用该寄存器）。

4.9 虚拟调试寄存器

虚拟寄存器在硬件中不直接存在，但调试器使其公开。调试软件应该实现它们，但是硬件可以跳过本节。虚拟寄存器存在，使用户可以访问标准调试器所不具备的功能；而调试器在访问这些相同寄存器的情况下，无需他们修改调试寄存器。

表 4.4：虚拟核心调试寄存器

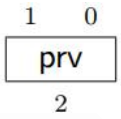
地址	名称
virtual	Privilege Level (priv)

4.9.1 Privilege Level (priv, at virtual)

用户可以读取该寄存器，以检查挂起的 hart 的特权级别。用户可以编写此寄存器，更改 hart 恢复时将在其中运行的特权级别。

该寄存器包含来自 dcsr 的 prv，但此位置用户可以访问。用户不应直接访问

dcsr，因为这样做可能会干扰调试器。



名称	描述	访问类型	Reset 值
prv	进入调试模式前，hart 正在操作的特权级别。编码在表 4.5 中进行了描述，并与 Privileged Spec 中的特权级别编码匹配。退出调试模式时，用户可以写入此字段以更改 hart 的特权级别。	R/W	0

Table 4.5: Privilege Level Encoding

Encoding	Privilege Level
0	User/Application
1	Supervisor
3	Machine

5 Trigger 模块

触发器可以导致断点异常，进入调试模式或执行跟踪操作，而无需执行特殊指令。当从 ROM 调试代码时，这使它们非常有价值。它们可以在给定的存储器地址或加载/存储中的地址/数据上执行指令时触发。这些都是在没有调试模块的情况下有用的功能，因此触发模块被分解为一个单独的部分，可以单独实现。

hart 可以完全符合此规范，而无需实现任何触发功能，但是如果实现，则必须符合本节的规定。

在 debug 模式下，触发器不会重新触发。

每个触发器可以支持多种功能。调试器可以构建所有触发器及其功能的列表，如下所示：

1.给 tselect 写 0。

2.读回 tselect 并检查它是否包含写的值。如果不是，请退出循环。

3.读 tinfo。

4.如果这引起异常，则调试器必须读取 tdata1 以发现 type。（如果 tdata1[type]为 0，此触发器不存在，退出循环。）

5.如果 tinfo[info]为 1，则此触发器不存在，退出循环。

6.否则，所选触发器支持在 tinfo[info]中发现的类型。

7.重复，在 tselect 中增加值。

上面的算法回读了 tselect，因此具有 2^n 个触发器的实现只需要实现 n 位 tselect。该算法检查 tinfo 和 tdata1[type]，以防实现中有 m 位 tselect 但少于 2^m 个触发器。

具有“进入调试模式”操作（1）的触发器和具有“引发断点异常”操作（0）的另一个触发器有可能同时重新执行。首选行为是同时执行两个操作。两者中的哪一个首先发生取决于实现。这样既可以确保外部调试器的存在不影响执行，也可以确保用户代码设置的触发器不影响外部调试器。如果未实现，则 hart 必须进入 debug 模式并忽略断点异常。在后一种情况下，action 为 0(产生断点异常)的触发器的 hit 必须被置位，从而使调试器有机会处理这种情况。当具有不同动作的触发器同时响应时，跟踪动作会发生什么，也要留给跟踪规范。

5.1 Native M-Mode 触发器

触发器可用于本机调试。在功能齐全的系统上，将使用 u 或 s 设置触发器，并且在触发时，它们可能会导致断点异常进入更高特权的模式。也可以将触发器本身设置为重新进入 M 模式，在这种情况下，没有更高的特权模式可捕获。如果此类触发器已在陷阱处理程序中而导致断点异常，则将使系统无法恢复正常执行。

在功能齐全的系统上，这是一个生僻的案例，可能会被忽略。但是，在仅实现 M 模式的系统上，建议实现此问题的两种解决方案之一。这种方式的触发器对于 M 模式代码的本地调试可能有用。

一种简单的解决方案是让硬件在 M 模式下以及 mstatus[MIE]为 0 时阻止 action=0 的触发器触发。其局限性在于，当用户希望触发触发器时，中断可能会禁止。

一个更复杂的解决方案是在 tcontrol 中实现 mte 和 mpte。该解决方案的好处是它仅在陷阱处理程序期间禁用触发器。

用户设置的 M 模式触发器会导致断点异常，因此必须知道他们正在使用的特定系统可能出现的任何问题。

5.2 触发器寄存器

trigger 寄存器是 CSR 寄存器，可以使用 RISC-V CSR 操作码访问，也可以选择使用抽象调试命令。

大多数触发器功能是可选的，所有 tdata 寄存器遵循 WARL 语义。如果调试器写入不支持配置，则寄存器将读回支持的值（可能只是禁用的触发器）。这意味着调试器必须始终读取它写入的一个 tdata 寄存器的值，除非它已经知道支持什么。写入一个 tdata 寄存器不能修改其他 tdata 寄存器的内容，也不能修改除当前选定的触发器之外的任何触发器的配置。

触发器寄存器只能在 M 模式和 debug 模式下访问，以防止不受信任的用户代码在未经操作系统许可的情况下导致进入调试模式。

在本节中，XLEN 在 M 模式下是 MXLEN，在 debug 模式下是 DXLEN。注意：这使得 tdata1 中的几个字段根据当前执行模式和 mxlen 的值移动。

Table 5.1: action encoding

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

Table 5.2: Trigger Registers

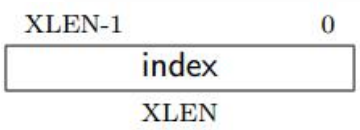
Address	Name
0x7a0	Trigger Select (`tselect`)
0x7a1	Trigger Data 1 (`tdata1`)
0x7a1	Match Control (`mcontrol`)
0x7a1	Instruction Count (`icount`)
0x7a1	Interrupt Trigger (`itrigger`)
0x7a1	Exception Trigger (`etrigger`)
0x7a2	Trigger Data 2 (`tdata2`)
0x7a3	Trigger Data 3 (`tdata3`)
0x7a3	Trigger Extra (RV32) (`textra32`)
0x7a3	Trigger Extra (RV64) (`textra64`)
0x7a4	Trigger Info (`tinfo`)
0x7a5	Trigger Control (`tcontrol`)
0x7a8	Machine Context (`mcontext`)
0x7aa	Supervisor Context (`scontext`)

5.2.1 Trigger Select (tselect, at 0x7a0)

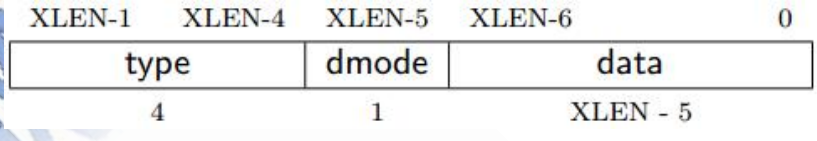
此寄存器决定访问哪个触发器，可访问触发器集从 0 开始，并且是连续的。

写入大于或等于支持的触发器的值可能会导致此寄存器的值与写入的值不同。为了验证他们所写的是有效的索引，调试器可以读回该值并检查 tselect 是否保存了他们所写的内容。

由于 debug 模式和 M 模式都可以使用触发器，因此如果调试器修改了该寄存器，则必须还原它。



5.2.2 Trigger Data 1 (tdata1, at 0x7a1)

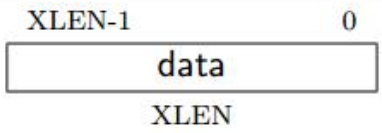


名称	描述	访问类型	Reset 值
type	0：在 tselect 时没有触发器； 1：触发器是传统的 sifive 地址匹配触发器。这些不应该被实现，这里也没有进一步的记录； 2：触发器是地址/数据匹配的触发器。此寄存器中的其余位按 mcontrol 中所述工作； 3：触发器是指令计数触发器。此寄存器中的其余位按 icount 中所述工作； 4：触发器是中断触发器。此寄存器中的其余位按 itrigger 中所述工作； 5：触发器是异常触发器。此寄存器中的其余位按 etrigger 中所述工作； 15：此触发器存在，但当前不可用。 其他值保留供将来使用。	R/W	preset
dmode	0：debug 和 M 模式都可以在 tselect 选择的 tdata 寄存器写值； 1：只有 debug 模式才能在 tselect 选择的 tdata 寄存器写值，忽略其他模式的写入。 此位只能在 debug 模式下写入。	R/W	0
data	触发器特定数据	R/W	preset

5.2.3 Trigger Data 2 (tdata2, at 0x7a2)

触发器特定数据。

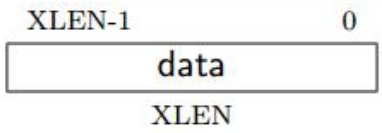
如果 XLEN 小于 DXLEN，写入寄存器的值需要符号位拓展。



5.2.4 Trigger Data 3 (tdata3, at 0x7a3)

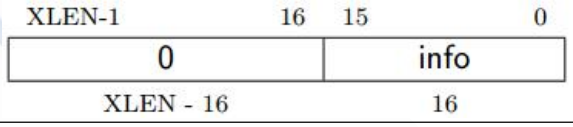
触发器特定数据。

如果 XLEN 小于 DXLEN，写入寄存器的值需要符号位拓展。



5.2.5 Trigger Info (tinfo, at 0x7a4)

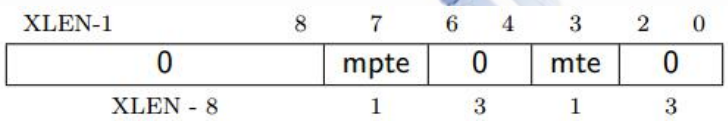
整个寄存器是只读的。



名称	描述	访问类型	Reset 值
info	列举了 tdata1 中每个可能的类型。每个类型占用 1bit，bit N 对应着 type N。 如果设置了某 bit，则当前选定的触发器支持该类型寄存器。 如果当前选择的触发器不存在，这个字段为 1。 如果 type 不可写，则此寄存器可能未实现，这种情况下，读取它可能导致非法指令异常。在这种情况下调试器可以从 tdata1 中读取唯一支持的类型。	R	preset

5.2.6 Trigger Control (tcontrol, at 0x7a5)

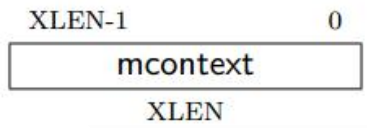
这是个可选的寄存器，该寄存器可以解决 M 模式 trap 处理程序中 action=0 的触发器问题。详细见 5.1 节。



名称	描述	访问类型	Reset 值
mpte	保存以前的 M 模式下的触发器使能字段； 当一个 trap 进入 M 模式时，mpte 被设置为 mte 的值。	R/W	0
mte	mte 仅影响 action 为 0 的触发器（C.1.5 更新） M 模式下触发器使能字段。 0：当 hart 运行在 M 模式下，触发器不匹配/不触发 1：当 hart 运行在 M 模式下，触发器匹配/触发 当 trap 进入 M 模式时，mte 设置为 0。 执行 mret 时，mte 被设置为 mpte 的值	R/W	0

5.2.7 Machine Context (mcontext, at 0x7a8)

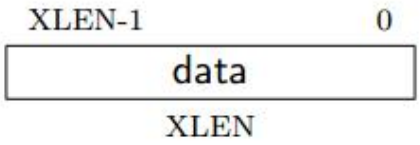
这个寄存器只在 M 模式和 debug 模式下是可写的。



名称	描述	访问类型	Reset 值
mcontext	M 模式下软件可以向该寄存器写入 context 的 number，该 number 用于设置位于该特定 context 的触发器。 可以将这个字段中的任何数量的高位绑定到 0。建议：在 RV32 的实现上不超过 6 位，在 RV64 的实现上不超过 13 位。	R/W	0

5.2.8 Supervisor Context (scontext, at 0x7aa)

这个寄存器只在 S 模式，M 模式和 debug 模式下是可写的。



名称	描述	访问类型	Reset 值
data	S 模式下软件可以向该寄存器写入 context 的 number，该 number 用于设置位于该特定 context 的触发器。 可以将这个字段中的任何数量的高位绑定到 0。建议：在 RV32 的实现上不超过 16 位，在 RV64 的实现上不超过 34 位。	R/W	0

5.2.9 Match Control (mcontrol, at 0x7a1)

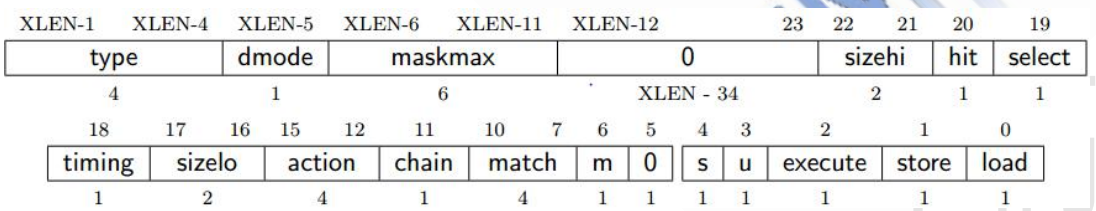
当 tdata1[type]为 2 时，此寄存器可作为 tdata1 访问。

地址和数据触发器的实现，在很大程度上取决于处理器核心的实现方式。为了适应不同的实现，执行、加载和存储地址/数据触发器可以实现在最方便的任何时间点。调试器可以请求 timing(本寄存器的字段)中描述的特定 timings。表 5.8 给出了最佳用户体验的时间安排。

Table 5.8: Suggested Breakpoint Timings

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

当 mcontrol[select]，bit19 始终为 0 时，此触发器类型仅限于地址比较。如果是这种情况，那么 tdata2 必须能够保存所有有效的虚拟地址，但不需要保存其他值。



名称	描述	访问类型	Reset 值
maskmax	当 mcontrol[match]为 1 时，该字段有效，硬件支持的最大范围是 2^n，NAPOT（naturally aligned powers-of-two）。 0 ：表示支持范围是 1 字节； 63：表示支持范围是 2^63 字节。	R	preset
sizehi	如果 XLEN 大于 32，该字段才存在，否则是 0。 它是 size（sizelo+sizehi）的扩展位。	R/W	0
hit	如果实现了该可选位，那么硬件会在这个触发器匹配时置位。触发器的用户可以随时设置或清除它。 如果该位未实现，则始终为 0，写入也没有任何效果。	R/W	0
select	0：对虚拟地址执行匹配； 1：对 load、store 或执行指令进行匹配。	R/W	0
timing	0：此触发器的操作，将在触发它的执行指令之前执行，但在前序提交的指令之后执行； 1：此触发器的操作，将在触发它的执行指令之后执行。在执行下一条指令之前执行，建议最好实现触发器。 大多数硬件只实现一个 timing 或者其他，可能依赖于 mcontrol 的 select、execute、load 和 store。该位主要用于硬件与调试器通信发生什么。硬件可以实现该位为完全写位，这样调试器会有更多的控制权。 当调试器允许 hart 运行时，mcontrol[timing]为 0 的 load 数据加载触发器将会导致相同的 load 加载再次发生。对于 load 数据加载触发器，调试器必须首先尝试设置 mcontrol[timing]为 1 的断点。 一个触发器链如果不是所有的触发器都有相同的 mcontrol[timing]值，就永远不会出现(除非连续的指令与相应的触发器匹配)。 如果 timing 为 0 的触发器匹配，则是否组织 timing 为 1 的触发器也匹配取决于实现。	R/W	0
sizelo	这个字段是 size(sizelo+sizehi)的低两位。Size 的含义如下： 0：只有当 mcontrol[select]为 0 或者访问大小为 XLEN 时，触发器将尝试与任何大小的访问进行匹配； 1：触发器只与 8 位内存访问执行匹配； 2：触发器只与 16 位内存访问或者 16 位指令执行匹配； 3：触发器只与 32 位内存访问或者 32 位指令执行匹配； 4：触发器只与 48 位指令的执行匹配； 5：触发器只与 64 位内存访问或者 64 位指令执行匹配； 6：触发器只与 80 位指令的执行匹配； 7：触发器只与 96 位指令的执行匹配； 8：触发器只与 112 位指令的执行匹配； 9：触发器只与 128 位内存访问或者 128 位指令执行匹配；	R/W	0
action	表 5.1 说明了触发值时要采取的措施。	R/W	0
chain	0：当此触发器匹配时，将执行配置的操作； 1：当这个触发器不匹配时，它会阻止下一个索引的触发器匹配。	R/W	0

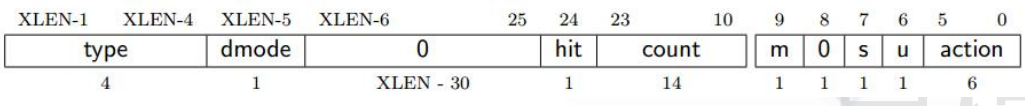
	在 chain 为 0 之后，遇到第一个 chain 为 1 的触发链上开始，在第一个触发器的 chain 为 0 上结束。这个最终的触发器是链上的一部分。除了最终触发器之外，其他所有触发器上的操作都被忽略。当且仅当链中的所有触发器同时匹配时，才会对最终触发器执行操作。 由于 chain 影响下一个触发器，硬件必须在写入 mcontrol 时将其归零，如果下一个触发器的 dmode 为 1，则 mcontrol 将 dmode 设置为 0。此外，如果上一个触发器的 dmode 为 0 且 chain 为 1，则硬件忽略对 mcontrol 的写操作，该写操作将 dmode 设置为 1。 如果调试器正在编写 mcontrol，则必须通过检查前一个触发器上的 chain 来避免后一种情况。希望限制触发链的最大长度的实现（例：满足时序要求）可以通过写入 mcontrol[chain]实现，这将使链太长。		
match	0：当值等于 tdata2 时匹配； 1：当值的前 M 位和 tdata2 的前 M 位匹配时匹配。M 是 XLEN-1 减去 tdata2 中包含 0 的 least-significant 位的索引； 2：当值大于（无符号）或等于 tdata2 时匹配； 3：当值小于（无符号）tdata2 时匹配； 4：当值的下半部分与 tdata2 的上半部分相与(AND)后，新值的下半部分与 tdata2 的下半部分相等时匹配； 5：当值的上半部分与 tdata2 的上半部分相与(AND)后，新值的上半部分与 tdata2 的下半部分相等时匹配； 其余的值保留供将来使用。	R/W	0
m	1：在 M 模式下使能此触发器	R/W	0
s	1：在 S 模式下使能此触发器	R/W	0
u	1：在 U 模式下使能此触发器	R/W	0
execute	1：触发器将显示执行的指令的虚拟地址或操作码	R/W	0
store	1：触发器将显示 store 的虚拟地址或数据	R/W	0
load	1：触发器将显示 load 的虚拟地址或数据	R/W	0

5.2.10 Instruction Count (icount, at 0x7a1)

当 tdata1[type]为 3 时，此寄存器可作为 tdata1 访问。

此触发器类型旨在用作对外部调试器和软件监视器程序都有用的 single step。在这种情况下，不需要支持大于 1 的 count。在这些场景中只有两种模式位组合有用：u 本身，或者 m、s 和 u 都设置。

如果硬件将 count 设置为 1，并且通过改变模式位而不是递减 count，则该寄存器可以仅实现 2 位，一位用于 u，一位用于 m 和 s。如果只需要支持外部调试器或者软件监视器，那么实现 1 位即可。



名称	描述	访问类型	Reset 值
hit	如果实现了该可选位，那么硬件会在这个触发器匹配时置位。触发器的用户可以随时设置或清除它。 如果该位未实现，则始终为 0，写入也没有任何效果。	R/W	0
count	当 count 减至 0 时，触发器触发。代替改变 count 从高减至 0，硬件也可以接受通过改变 m、s 和 u 清除。 如果该寄存器仅为 single step 实现，则可以硬连线为 0。	R/W	1
m	1：M 模式下完成的每条指令或执行的异常都将 count 递减 1	R/W	0
s	1：S 模式下完成的每条指令或执行的异常都将 count 递减 1	R/W	0
u	1：U 模式下完成的每条指令或执行的异常都将 count 递减 1	R/W	0
action	表 5.1 说明了触发值时要采取的措施。	R/W	0

5.2.11 Interrupt Trigger (itrigger, at 0x7a1)

当 tdata1[type]为 4 时，此寄存器可作为 tdata1 访问。

此触发器可以触发 mie 中可配置的任何中断（在特权规范手册中 mie 寄存器）。通过在 tdata2 中设置与 mie 中相同位以启用中断来配置要打开的中断。

硬件可能只支持此触发器的中断子集，调试器可以通过写入 tdata2 后读回值，以确认实际支持哪个中断触发功能。

只有当 hart 因为 interrupt 中断而进入 trap 时，才会触发。（例：当中断使能在 mie 中未打开时，计时器中断发生不起作用）

当触发器触发时，所有 CSR 寄存器都会按照特权规范手册的规定进行更新，并且请求的操作在中断/异常处理程序的第一条指令执行之前执行。

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

名称	描述	访问类型	Reset 值
hit	如果实现了该可选位，那么硬件会在这个触发器匹配时置位。触发器的用户可以随时设置或清除它。 如果该位未实现，则始终为 0，写入也没有任何效果。	R/W	0
m	1：从 M 模式获取的中断启用此触发器	R/W	0
s	1：从 S 模式获取的中断启用此触发器	R/W	0
u	1：从 U 模式获取的中断启用此触发器	R/W	0
action	表 5.1 说明了触发值时要采取的措施。	R/W	0

5.2.12 Exception Trigger (etrigger, at 0x7a1)

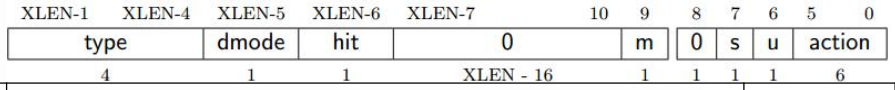
当 tdata1[type]为 5 时，此寄存器可作为 tdata1 访问。

此触发器最多可以触发 mcause 中定义的异常（在特权规范手册中描述，中断=0）。这些 cause 可以通过在 tdata2 中设置相应位来确定（为了获取非法指令，

调试器设置 tdata2 中的 bit2）。

硬件可能只支持此触发器的异常子集，调试器可以通过写入 tdata2 后读回值，以确认实际支持哪个异常触发功能。

当触发器触发时，所有 CSR 寄存器都会按照特权规范手册的规定进行更新，并且请求的操作在中断/异常处理程序的第一条指令执行之前执行。

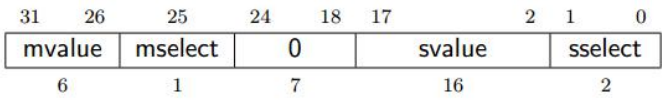


名称	描述	访问类型	Reset 值
hit	如果实现了该可选位，那么硬件会在这个触发器匹配时置位。触发器的用户可以随时设置或清除它。 如果该位未实现，则始终为 0，写入也没有任何效果。	R/W	0
m	1：从 M 模式获取的异常启用此触发器	R/W	0
s	1：从 S 模式获取的异常启用此触发器	R/W	0
u	1：从 U 模式获取的异常启用此触发器	R/W	0
action	表 5.1 说明了触发值时要采取的措施。	R/W	0

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

当 tdata1[type]为 2,3,4 或 5 时，此寄存器可作为 tdata3 访问。

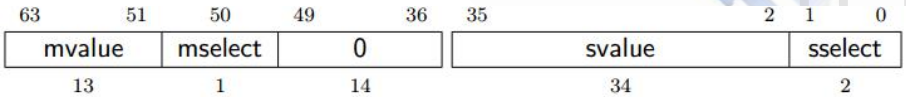
此寄存器中的所有功能都是可选的，value 位可以将任何数量的高位绑定到 0，select 位只能支持 0 位。



名称	描述	访问类型	Reset 值
mvalue	mselect 有效时使用的数据	R/W	0
mselect	0：忽略 mvalue 1：只有当 mcontext 的低位和 mvalue 相等时，此触发器才会匹配	WARL	0
svalue	sselect 有效时使用的数据	R/W	0
sselect	0：~~忽略 svalue；~~使用 svalue；（C.1.6 更新） 1：只有当 scontext 的低位和 svalue 相等时，此触发器才会匹配； 2：只有当 satp 中的 asic 等于 svalue 的较低位 ASIDMAX（特权规范定义）位时，此触发器才会匹配。（Sv32 是 9，Sv39 和 Sv48 是 16）	WARL	0

5.2.14 Trigger Extra (RV64) (textra64, at 0x7a3)

如果 XLEN 是 64，下图是 textra 的布局。以上面的 textra32 对这些字段进行了定义。



6 调试传输模块 DTM

Debug Transport Module 调试传输模块通过一个或多个 transports（如 JTAG 或 USB）提供对 debug module(DM)的访问。

一个平台中可能有多个 DTM，理想情况下，与外部交互的每个组件都包含一个 DTM。例如，USB 组件可以包含 DTM，这将使得任何平台都可以通过 USB 进行调试。已经在使用的 USB 模块还可以访问调试模块接口。

不支持同时使用多个 DTM，用户需要保证不会发生这种情况。

本规范在 6.1 节中定义了 JTAG DTM，在未来的版本中，可能会添加其他的 DTM。

实现了 DTM，但是不符合本节的描述，则必须说明它“符合 RISC-V 调试规范 0.13.2，带有自定义的 DTM”。如果实现了 DTM 且满足本节描述规范，则必须说明它“符合 RISC-V 调试规范 0.13.2，带有 JTAG DTM”。

6.1 JTAG 调试传输模块

这个调试传输模块是基于一个普通的 JTAG Test Access Port(TAP)。JTAG TAP 允许访问任意 JTAG 寄存器，首先使用 JTAG 指令寄存器 IR 选择一个，然后通过 JTAG 数据寄存器 DR 访问。

6.2 JTAG 背景

JTAG 参考 IEEE Std 1149.1-2013。该标准规定了集成电路中可包含的测试逻辑，以测试集成电路之间的互连，测试集成电路本身，并在组件正常运行期间观察或修改电路活动。本规范使用后一种功能。JTAG 标准定义了一个测试访问端口 TAP，可用于读取和写入一些自定义寄存器，这些寄存器可用于与组件中的调试硬件通信。

6.3 JTAG DTM 寄存器

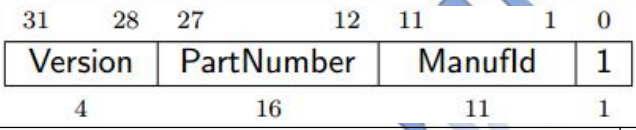
用作 DTM 的 JTAG TAP 必须具有至少 5 位的 IR。当 TAP 复位时，IR 设为 00001，选择 idcode 指令。JTAG 寄存器及其编码的完整列表见表 6.1。如果 IR 实际上超过 5 位，那么表 6.1 中的编码应该在最高有效位处扩展。调试器使用常规的 jtag 寄存器是 bypass 和 idcode，但此规范为许多其他标准 jtag 指令留有空间，未实现的指令必须选择下面的 reserved bypass 寄存器。

Table 6.1: JTAG DTM TAP Registers

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

6.3.1 IDCODE (at 0x01)

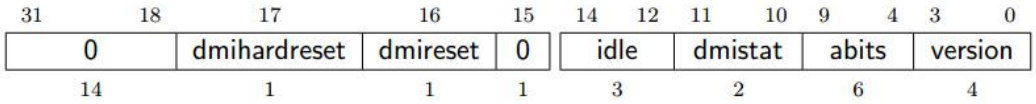
IEEE Std 1149.1-2013 中明确定义，当 TAP 选 IR 寄存器且状态机是 0x01，则选择 IDCODE。该寄存器是只读的。



名称	描述	访问类型	Reset 值
Version	版本号	R	preset
PartNumber	设计者的 part number	R	preset
Manufld	确定本部分的设计师/制造商。Bit6:0 必须是由 JEDEC 标准 JEP106 指定的设计师/制造商识别的编码。Bit10:7 包含同一识别码中连续字符数(0x7f)的模 16 计数。	R	preset

6.3.2 DTM Control and Status (dtmcs, at 0x10)

此寄存器的大小在将来的版本中保持不变，以便调试器始终可以确定 DTM 的版本。



名称	描述	访问类型	Reset 值
dmihardreset	将 1 写入该位会导致 DTM 硬置位，从而导致 DTM 忘记任何未完成的 DMI 事物。一般来说，只有当调试器知道未完成的 DMI 事物永远不会完成时（例如，重置条件导致取消了 DMT 事物），才应使用此选项。	W1	-
dmireset	将 1 写入该位会清除粘连错误状态，并允许 DTM 重试或完成上一个事物。	W1	-
idle	这是对调试器的一个提示。提示调试器在每次 DMI 扫描后，在 run-test/idle 中应花费的最小周期数，以避免返回‘busy’代码（dmistat 为 3） 0：不需要输入 run-test/idle；	R	preset

	1：输入 run-test/idle 并立即离开； 2：进入 run-test/idle 并在离开前停留 1 个周期		
dmistat	0：没有错误 1：保留，解释为 2 2：操作失败（导致 dmi[op]为 2） 3：在 DMI 访问仍在进行时尝试一个操作（dmi[op]为 3）	R	0
abits	dim[address]的大小	R	preset
version	0：spec 版本 0.11 1：spec 版本 0.13 15：本规范的任何可用版本中均为描述	R	1

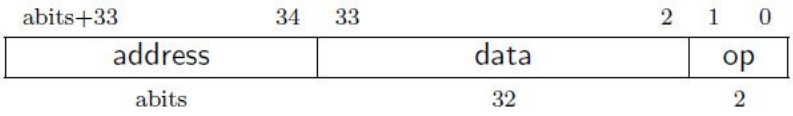
6.3.3 Debug Module Interface Access (dmi, at 0x11)

此寄存器允许访问调试模块接口 DMI。

在 update-dr 时，DTM 启动 op 中的指定操作，除非当前的 op 是 sticky。

在 capture-dr 时，如果当前的 op 不是 sticky，DTM 使用该操作的结果更新 data 字段。

Still-in-progress 的状态是粘性的，以容纳多个扫描，一旦有问题就必须全部执行或停止。例如，一系列扫描可以编写调试程序并执行它。如果其中一个写失败但执行仍在继续，则调试程序可能挂起或出现其他不期待的副作用。

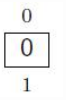


名称	描述	访问类型	Reset 值
address	用于 DMI 访问的地址。在 update-dr 时，DMI 用该值访问 DM。	R/W	0
data	在 update-dr 时，DMI 把该值送给 DM，该 data 也是先前操作的结果从 DM 返回的数据。	R/W	0
op	当调试器写此字段时，它具有以下含义： 0：忽略数据和地址。（nop） 在 update-dr 期间，请勿通过 DMI 发送任何内容。此操作永远不会导致 busy 或 error 响应。地址和数据在后续的 Capture-DR 中定义。 1：从 address 读取。（read） 2：将 data 写入 address。（write） 3：保留。 当调试器读取此字段时，它具有以下含义： 0：上一次操作成功完成。 1：保留。 2：先前的操作失败。扫描数据进入 dm 的访问将被忽略。这种状态是 sticky，可以通过写 dtmcs[dmireset]来清除。 这表明 DM 本身以一个错误响应。没有特定情况下 DM 会回应错	R/W	0

	误，而 DMI 不需要支持返回错误。 3：在 DMI 请求仍在进行中时试图进行操作。扫描到的 dmi 访问数据将被忽略。此状态为 sticky，可以通过写 dtmcs[dmireset]来清除。如果调试器看到此状态，则需要在 Update-dr 和 Capture-DR 之间给目标提供更多的 TCK edges。最简单的方法是在 run-test/idle 中添加额外的过渡。		

6.3.4 BYPASS (at 0x1f)

该寄存器是 1 位的。当调试器不想与此 TAP 通信时使用它。整个寄存器是只读的。



6.3.5 Recommended JTAG Connector（推荐的 jtag 连接器）

为了方便获取调试硬件，此规范建议使用与 MIPI-10 .05 英寸规格的兼容的连接器，如 2011 年 3 月 16 日的 MIPI 联盟 1.10.00 版调试和跟踪连接器中所述。

该连接器的间距为.05 英寸，镀金公头带有.016 英寸厚的硬化铜或铍青铜方柱（SAMTEC FTSH 或同等产品）。母连接器是兼容的 20um 金连接器。

从上方查看公头连接器（引脚指向您的眼睛），目标连接器的外观如表 6.5 所示。表 6.7 中描述了每个引脚的功能。

Table 6.5: MIPI-10 Connector Diagram

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

如果平台需要 nTRST，则可以将 nRESET 引脚重用为 nTRST 信号。如果平台需要系统重置和 TAP 重置，则应使用 MIPI-20 连接器。它的物理连接器实际上与 MIPI-10 相同，只是它的长度是 MIPI-10 的两倍，支持两倍的引脚。其连接器如表 6.6 所示。

Table 6.6: MIPI-20 Connector Diagram

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

相同的连接器可用于 2-wire cJTAG。在这种情况下，TMS 用于 TMSC，而 TCK 用于 TCKC。

Table 6.7: JTAG Connector Pinout

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

附录 A 硬件实现

以下是两种可能的实现。设计师可以选择一个，混合搭配，或者提出自己的设计。

A.1 基于抽象命令

Halting 是将正在执行的 hart 挂起。

寄存器文件上的多路复用器允许使用访问寄存器抽象命令访问 GPR 和 CSR。

使用“抽象访问内存”命令或“系统总线访问”来访问内存。

这种实现方式可以使用调试器，即使在无法执行指令的情况下也可以收集信息。

A.2 基于执行

此实现仅在挂起的 hart 上为 GPR 实现访问寄存器抽象命令，并且依赖于 Program Buffer 进行所有其他操作。它利用了 hart 的现有 pipeline 以及从任意内存位置执行的能力，从而避免对 hart 的数据通路进行修改。

当暂停请求位置 1 时，DM 调试模块将对选定的 hart 产生特殊中断。该中断使每个 hart 进入 Debug 模式并跳转到由 DM 定义的存储区域处理。发生此异常时，会将 pc 保存到 dpc，并将 dcsr 中的 cause 字段更新。

调试模块中的代码使 hart 执行“停放循环 park loop”。在停放循环中，hart 将其 mhartid 写入调试模块内的内存位置，以指示其已停止。为了使 DM 单独控制几个挂起的 hart 中的一个，每个 hart 都会轮询 DM 控制的标志，该标志位于存储器中，以确定调试器是希望其执行**程序缓冲区 Program Buffer** 还是执行**恢复 resume**。

为了执行抽象命令，首先，DM 根据 command 填充程序缓冲区的一些内部字。设置 transfer 后，DM 会使用 lw <gpr>，0x400（zero）或 sw 0x400（zero），<gpr> 填充这些字。64 位和 128 位访问分别使用 ld/sd 和 lq/sq。如果未设置 transfer，则 DM 将这些指令填充为 nop。如果设置了 execute，则执行继续到调试器控制的**程序缓冲区 Program Buffer**，否则 DM 立即执行 ebreak。

当执行 ebreak 时（指示程序缓冲区代码的末尾），hart 返回其停放循环 park loop。如果遇到异常，则 hart 会跳到“调试模块”中的调试异常地址。该地址处

的代码使 hart 写入 Debug 中的地址指示异常的模块。该地址被视为 fence 指令的 I/O（请参阅 4.1 节上的＃9）。然后，hart 跳回到 park loop。DM 从写入中推断出是一个例外，并适当设置了 cmderr。

为了恢复执行，调试模块设置了一个标志，该标志使 hart 执行一个 dret。当执行 dret 时，将从 dpc 恢复 pc，并以 prv 设置的特权恢复正常执行。

data0 等寄存器，仅以 12 位 imm 映射到相对于 zero 的地址的常规存储器中。确切的地址调试器得依赖于实现细节。例如，data 寄存器可能映射到 0x400。

为了提高灵活性，将 progbuf0 等寄存器映射到紧靠 data0 的常规存储器中，以形成可用于程序执行或数据传输的连续存储器区域。

附录 B 调试器实现

本节详细介绍了外部调试器如何使用上述调试接口，通过 6.1 节中描述的 JTAG DTM 在 RISC-V 内核上执行一些常见操作。所有这些示例假定使用 32 位内核，但是该示例适应 64 位或 128 位内核应该很容易。

为了使示例易于理解，它们都假定一切都成功，并且完成速度比调试器执行下一次访问的速度快。它将是一个典型的 JTAG 设置。但是，调试器在执行一系列操作后必须检查粘性错误状态位。如果看到任何内容被设置，则应再次尝试相同的操作（可能会增加一些延迟），或者检查状态位。

B.1 调试模块接口访问

要**读取**任意调试模块寄存器，需选择 dmi，然后将 op 设置为 1，并将 address 设置为所需寄存器地址的值。在 Update-DR 中，操作将开始，在 Capture-DR 中其结果将被捕获到 data 中。如果操作未及时完成，则 op 将设为 3，并且必须忽略 data 中的值。必须通过写 dtmcs[dmireset]来清除*繁忙状态*，然后必须再次执行第二次扫描。这个过程必须重复此操作，直到 op 返回 0。在以后的操作中，调试器应在 Capture-DR 和 Update-DR 之间留出更多时间。

要**写入**任意调试总线寄存器，需选择 dmi，然后将 op 设置为 2 的值进行扫描，并将 address 和 data 分别设置为所需的寄存器地址和数据。从那时起，一切都与读取完全一样，只是执行写入而不是读取。

几乎绝对不需要扫描 IR，从而避免了 JTAG 使用中的大部分不便。

B.2 检查停机

用户可能希望尽可能快地知道 hart 何时挂起了（例如：由于断点）。为了有效地确定在多 hart 时停止了哪些 hart，调试器将使用 haltsum 寄存器。假设存在最大数量的 hart，则首先检查 haltsum3。对于在那里设置的每个位，它都会写入 hartsel，并检查 haltsum2。在 haltsum1 和 haltsum0 重复此过程。根据存在的 hart 数量，该过程应从较低的一个 haltsum 寄存器开始。

B.3 暂停

要挂起一个或多个 hart，调试器选择它们，设置 haltreq，然后等待 allhalted 以指示 hart 停止。然后，它可以将 haltreq 清除为 0，或者将其保持为高电平以捕

获重置停止的 hart。

B.4 运行

首先，调试器应恢复已覆盖的所有寄存器。然后，可以通过设置 resumereq 让选定的 hart 恢复运行。设置好 allresumeack 之后，调试器便知道 hart 已恢复，它可以清除 resumereq。重启后，hart 可能会非常迅速地挂起（例如，通过命中软件断点），因此调试器无法使用 allhalted/anyhalted 来检查 hart 是否已恢复。

B.5 单步调试

硬件单步功能几乎与常规运行相同。调试器在让 hart 运行之前设置 step 和 dcsr。除了可以禁用中断（取决于 stepie），并且在重新进入调试模式之前，它仅获取并执行一条指令，该 Hart 的行为与正在运行的情况完全相同。

B.6 访问寄存器

B.6.1 使用抽象命令

使用抽象命令读取 s0：

Op	Address	Value	Comment
Write	command	aarsize = 2, transfer, regno = 0x1008	Read s0
Read	data0	-	Returns value that was in s0

使用抽象命令写 mstatus：

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

B.6.2 使用程序缓冲区

抽象命令用于与 GPR 交换数据。使用这种机制，其他寄存器可以通过将其值移入/移出 GPR 来访问它们。

使用程序缓冲区写入 mstatus：

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

使用程序缓冲区读取 f1：

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 读取 memory

B.7.1 使用系统总线访问

通过系统总线访问，地址是物理系统总线地址。

使用系统总线访问从内存中读取一个 word：

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

使用系统总线访问读取内存块：

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 使用程序缓冲区

通过 program buffer，hart 执行内存访问。地址是物理的还是虚拟的（取决于 mprven 和其他系统配置）。

使用程序缓冲区从内存中读取一个字：

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

使用程序缓冲区读取内存块：

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 使用抽象内存访问

抽象内存访问的行为就像它们是由 hart 执行的，实现可能会有所不同。

使用抽象内存访问从内存中读取一个单词：

Op	Address	Value	Comment
Write	data1	address	
Write	command	cmdtype=2, aamsize =2	
Read	data0	-	Value read from memory

使用抽象内存访问读取内存块：

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 写 memory

B.8.1 使用系统总线访问

通过系统总线访问，地址是物理系统总线地址。

使用系统总线访问权限将一个 word 写到内存中：

Op	Address	Value	Comment
Write	sbaddress0	address	
Write	sbdata0	value	

使用系统总线访问权限写一个内存块：

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 使用程序缓冲区

通过程序缓冲区，hart 执行存储器访问。地址是物理的还是虚拟的（取决于 mprven 和其他系统配置）。

使用程序缓冲区将一个 word 写入内存：

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

使用程序缓冲区写入内存块：

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 使用抽象内存访问

尽管实际实现可能有所不同，但抽象内存访问的行为就像是由 hart 执行的。

使用抽象内存访问将 word 写到内存中：

Op	Address	Value	Comment
Write	data1	address	
Write	data0	value	
Write	command	cmdtype=2, aamsize =2, write=1	

使用抽象内存访问写一个内存块：

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 触发器

当特定事件发生时，调试器可以使用硬件触发器来挂起 hart。以下是一些示例，但是对由 hart 实现的触发器的功能数量没有要求，因此这些示例可能不适用于所有实现。当调试器想要设置触发器，它将写入所需的配置，然后回读以查看是否支持该配置。

在执行 0x80001234 的指令之前，进入 debug 模式，以用作 ROM 中的指令断点：

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

读取 0x80007f80 的值后立即进入 debug 模式：

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

在写入 0x80007c80 和 0x80007cef 之间的地址（包括）之前，立即进入 debug 模式：

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)

在写入 0x81230000 和 0x8123ffff（包括）之间的地址之前，立即进入 dedug 模式：

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.

从 0x86753090 和 0x8675309f 之间或 0x96753090 和 0x9675309f（含）之间的地址读取后立即进入 debug 模式：

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 异常处理

通常，调试器通过谨慎对待编写的程序来避免异常。然而，有时它们是不可避免的，例如如果用户要求访问未实现的内存或 CSR。典型的调试器对平台的了解不足，无法知道将要发生的情况，因此必须尝试访问以确定结果。

当执行程序缓冲区时发生异常时，cmderr 将置位。调试器可以检查该字段以查看程序是否遇到异常。如果有异常，则将其留给调试器以了解是什么引起的。

B.11 快速访问

在 GPR 和 data 寄存器之间传输数据有多种指令。它们是 loads/stores 或 CSR reads/writes。具体地址也有所不同。这些都在 hartinfo 中指定。此处的示例使用伪运算 **transfer dest, src** 表示所有这些选项。

在最短的时间内挂起 hart 以执行一次内存写入：

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

该示例显示了在 mcontrol 中设置 m 位，在 M 模式下启用硬件断点的示例。之前可能已经使用过类似的快速访问说明来配置在此处启用的触发器：

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