RISC-V 外部デバッグサポート バージョン 0.13.2

はじめに

この文章は RISC-V の外部デバッグサポートマニュアルを @shibatchii が RISC-V アーキテクチャ勉強のためメモしながら訳しているものです。
原文は https://riscv.org/specifications/ にある riscv-debug-release.pdf です。

原文のライセンス表示
ですが、
”The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2", Editors Andrew Waterman and Krste Asanovic, RISC-V Foundation, May 2017.
や
”The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10", Editors Andrew Waterman and Krste Asanovic, RISC-V Foundation, May 2017.
のように
Creative Commons Attribution 4.0 International License
表示がありません。

本文中に
1.1.1 C ontext
This document is written to work with:
1. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2 (the ISA Spec)
2. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 (the Privileged Spec)
とあるのでそれを引き継いでいるとも(ちょっと強引かもしれませんが)考えられます。

なのでまずは「RISC-V 外部デバッグサポート バージョン 0.13.2」 日本語訳 @shibatchii
ということで進めますが、まずいよー となったら速攻削除します。
RISC-V のメーリングリストで聞いてみれば良いのかな。

https://github.com/shibatchii/RISC-V
に置いてあります。

英語は得意でないので誤訳等あるかもしれません。ご指摘歓迎です。
Twitter: @shibatchii

Google 翻訳、Bing 翻訳、Mirai 翻訳、Webilo 翻訳、Exclite 翻訳 を併用しながら翻訳し、勉強しています。

まずは意味が分からないところもあるかもしれませんが、ざっくり訳して 2 周位回ればまともになるかなと。
体裁とかは後で整えようと思います。

文章は以下の様に色分けしてます。

黒文字：翻訳した文書。
赤文字：@shibatchii コメント。わからないところとか、こう解釈したとか。
青文字：RISC-V にあまり関係なし。訳した日付とか、集中力が切れた時に書くヨタ話とか。

2019/04/14 @shibatchii

RISC-V 外部デバッグサポート
バージョン 0.13.2
d5029366d59e8563c08b6b9435f82573b603e48e

編集者：
ティム・マンション <tim@sifie.com>、SiFive、Inc.
ミーガン ワッツ <megan@sifie.com>、SiFive、Inc.

3 月 22 日（金）09:06:04 2019 -0700

アルファベット順の仕様全バージョン貢献者。（修正を提案するには編集者に連絡してください）。
ブルース・アビディンガー、クレステ・アサノビッチ、アレン・バウム、マーク・ビール、アレックス・ブラッドベリー、チャンハ―・チャン、ジョンホー・チェン、モンテ・ダーリンプル、ヴャチェスラフ・ディアチェンコ、ピーター・ イゴールド、マルクス・ゴエール、ロバート・ゴッラ、ジョン・ハウザー、リチャード・ ハーベイユ、ヨンチン・ シャオ、ポウエイ・フアング、スコット・ジョンソン、ジャンリュック・ナーゲル、アラム・ナヒディプール、リジール・ニシール、ガジンダー・パンザー、ディーパック・パンワル、アントニー・パヴロフ、クラウス・クルーゼ・ペダーセン、ケン・ペティット、ジョー・ラーメ、ギャヴィン・スターク、ウェズリー・ターレット、ヤン・ウィレム・ヴァン・ド・ヴェールト、ステファン・ウォーレントイッツ、レイ・ヴァン・デ・ウォーカー、アンドリュー・ウォーターマン、アンディ・ライト、そして、ブライアン・ワイアット。

内容

図の一覧

テーブル一覧

RISC-V 外部デバッグサポート バージョン 0.13.2

前書き

設計がシミュレーションからハードウェア実装に進むと、ユーザーの制御とシステムの現在の状態の理解は劇的に低下します。
低レベルのソフトウェアやハードウェアを起動してデバッグするためには、ハードウェアに優れたデバッグサポートを組み込むことが重要です。
堅牢な OS がコア上で実行されている場合、ソフトウェアは多くのデバッグタスクを処理できます。
ただし、多くの場合、ハードウェアサポートは不可欠です。

この資料は RISC-V プラットフォームの外部デバッグサポートのための標準的なアーキテクチャを概説したものです。
このアーキテクチャは、さまざまな RISC-V 実装を補完する、さまざまな実装やトレードオフを可能にします。
同時に、この仕様では、デバッグツールやコンポーネントが RISC-V ISA をベースにしたさまざまなプラットフォームを対象にできるように、共通のインタフェースを定義しています。

システム設計者はハードウェアデバッグサポートを追加することを選択するかもしれませんが、この仕様は一般的な機能のための標準インタフェースを定義します。

-- 2019/04/14

1.1 用語

プラットフォームは、1 つ以上のコンポーネントで構成される単一の集積回路です。
一部のコンポーネントは RISC-V コアですが、その他のコンポーネントは異なる機能を持つ場合があります。
通常、それらはすべて単一のシステムバスに接続されます。
単一の RISC-V コアには、ハートと呼ばれる 1 つ以上のハードウェアスレッドが含まれています。
ハートの DXLEN は、misa の MXL の現在の値を無視して、その最も広くサポートされている XLEN です。

1.1.1 コンテキスト

この文書は以下のものを扱うように書かれています。

1. RISC-V 命令セットマニュアル第 1 巻：ユーザーレベルの ISA、文書バージョン 2.2（ISA 規格）

2. RISC-V 命令セットマニュアル第 2 巻：特権アーキテクチャ、バージョン 1.10（特権仕様）

1.1.2 バージョン

この文書のバージョン 0.13 は RISC-V 財団の理事会によって承認されました。
バージョン 0.13.x はその批准された仕様へのバグ修正リリースです。
バージョン 0.14 はバージョン 0.13 との互換性があります。

1.2 この文書について

1.2.1 構造

この文書は 2 部構成です。
この文書の主要部分は仕様であり、それは番号付きセクションに示されています。
この文書の 2 番目の部分は一連の付録です。
付録の情報は、例を明確にして提供することを目的としていますが、実際の仕様の一部ではありません。

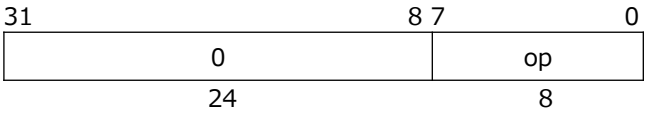
1.2.2 レジスタ定義形式

この文書内のすべてのレジスタ定義は以下に示すフォーマットに従います。
単純なグラフィックは、どのフィールドがレジスターにあるかを示します。
上位および下位のビットインデックスは、各フィールドの左上と右上に表示されます。
フィールドの合計ビット数はその下に表示されます。

グラフィックの後に、各フィールドの名前、説明、許可されたアクセス、およびリセット値をリストした表が続きます。
許可されているアクセスを表 1.2 に示します。
リセット値は定数または「プリセット」です。
後者は、それが実装固有の有効な値であることを意味します。

レジスタ名とそのフィールドはそれらの定義へのハイパーリンクであり、82 ページのインデックスにもリストされています。

1.2.2.1 ロングネーム（ショートネーム、0x123）



フィールド	説明	アクセス	リセット
フィールド	このフィールドが何のために使われているかの説明。	R/W	15

表 1.2：レジスタアクセスの略語

R	読み出しのみ。
R/W	読み出し／書き込み。
R/W1C	読み出し／書き込み。フィールドの各ビットについて、1 を書き込むとそのビットがクリアされます。 0 を書き込んでも効果はありません。
W	書き込み専用。このフィールドを読むと 0 が返されます。
W1	書き込み専用 1 を書くだけで効果があります。
WARL	任意の書き込み、正規読みだし。 デバッガは任意の値を書くことができます。 値がサポートされていない場合、実装はその値をサポートされている値に変換します。

→legal ってどう訳すのがいいのだろう。法的 は違うよね。

1.3 背景

専用デバッグハードウェアには、CPU コアの内部と外部接続の両方の使用例がいくつかあります。
この仕様は、下記のユースケースを扱います。
実装は、すべての機能を実装しないことを選択できます。つまり、一部のユースケースはサポートされていない可能性があります。

- OS や他のソフトウェアがない状態で低レベルのソフトウェアをデバッグする。
- OS 自体の問題をデバッグする。
- システムに実行可能コードパスが存在する前に、システムをブートストラップしてコンポーネントをテスト、構成、およびプログラムします。
- 動作している CPU なしでシステム上のハードウェアにアクセスする。

さらに、ハードウェアデバッグインタフェースがなくても、RISC-V CPU のアーキテクチャサポートは、ハードウェアトリガとブレークポイントを可能にすることによってソフトウェアデバッグとパフォーマンス分析を支援することができます。

1.4 サポートされている機能

この仕様で説明されているデバッグインタフェースは、次の機能をサポートしています。

1.すべてのハートレジスタ（CSR を含む）は読み書き可能です。

2.メモリは、ハートの観点から、システムバスを介して直接、またはその両方からアクセスすることができます。

3. RV32、RV64、および将来の RV128 がすべてサポートされています。

4.プラットフォーム内の任意のハートは独立して(個別に)デバッグできます。

5.デバッガは、ユーザ設定なしで、自分自身を知るために必要なほとんど[1]すべてを発見できます。

---

[1] 注目すべき例外には、メモリマップと周辺機器に関する情報が含まれます。

6.各ハートは、実行された最初の命令からデバッグできます。

7.ソフトウェアブレークポイント命令を実行すると、RISC-V ハートを停止できます。

8.ハードウェアシングルステップは一度に 1 つの命令を実行できます。

9.デバッグ機能は、使用されるデバッグ転送とは無関係です。

10.デバッガは、デバッグしているハートのマイクロアーキテクチャについて何も知る必要はありません。

11.ハートの任意のサブセットを同時に停止して再開することができます。 （オプション）

12.停止したハートに対して任意の命令を実行できます。
つまり、コアに追加の命令やカスタムの命令や状態がある場合、その状態を GPR に移行できるプログラムが存在する限り、新しいデバッグ機能は不要です。 （オプション）

13.停止することなくレジスタにアクセスできます。 （オプション）

14.実行中のハートは、わずかなオーバーヘッドで、短い一連の命令を実行するように指示されることができます。 （オプション）

15.システムバスマスタは、ハードを使わずにメモリアクセスを可能にします。 （オプション）

16.トリガが PC、読み出し/書き込みアドレス/データ、または命令オペコードに一致すると、RISC-V ハートを停止することができます。 （オプション）

17.この資料はハードウェアテスト、デバッグまたはエラー検出技術のための戦略か実装を提案しません。
スキャン、BIST などはこの仕様の範囲外ですが、この仕様は RISC-V システムでの使用を制限する意図はありません。

18.ソフトウェアスレッドを使用するコードをデバッグすることは可能ですが、それに対する特別なデバッグサポートはありません。

第 2 章

システム概要

図 2.1 は、外部デバッグサポートの主要コンポーネントを示しています。
点線で示されているブロックはオプションです。

ユーザはデバッガ（例えば、ｇｄｂ）を実行しているデバッグホスト（例えば、ラップトップ）と対話する。
デバッガはデバッグ転送ハードウェア（例えば Olimex USB-JTAG アダプタ）と通信するためにデバッグトランスレータ（例えば
ハードウェアドライバを含むことができる OpenOCD）と通信します。
デバッグ転送ハードウェアはデバッグホストをプラットフォームのデバッグ転送モジュール（DTM）に接続します。
DTM は、デバッグモジュールインタフェース（DMI）を使用して 1 つ以上のデバッグモジュール（DMs）へのアクセスを提供します。

プラットフォーム内の各ハートは、厳密に 1 つの DM によって制御されます。
ハーツは不均質であるかもしれません。
ハート DM のマッピングにこれ以上の制限はありませんが、通常、単一コア内のすべてのハートは同じ DM によって制御されます。
ほとんどのプラットフォームでは、プラットフォーム内のすべてのハートを制御する DM は 1 つだけです。

DM はプラットフォームで自分のハートの実行制御を提供します。
抽象コマンドは GPR へのアクセスを提供します。
追加のレジスタは、抽象コマンドを介して、またはオプションのプログラムバッファにプログラムを書き込むことによってアクセス
できます。

プログラムバッファはデバッガがハート上で任意の命令を実行することを可能にします。
このメカニズムはメモリへのアクセスにも使用できます。
オプションのシステムバスアクセスブロックを使用すると、RISC-V ハートを使用せずにアクセスを実行できます。

各ＲＩＳＣ－Ｖハートはトリガモジュールを実装してもよいです。
トリガ条件が満たされると、ハートは停止し、デバッグモジュールにそれらが停止したことを通知します。

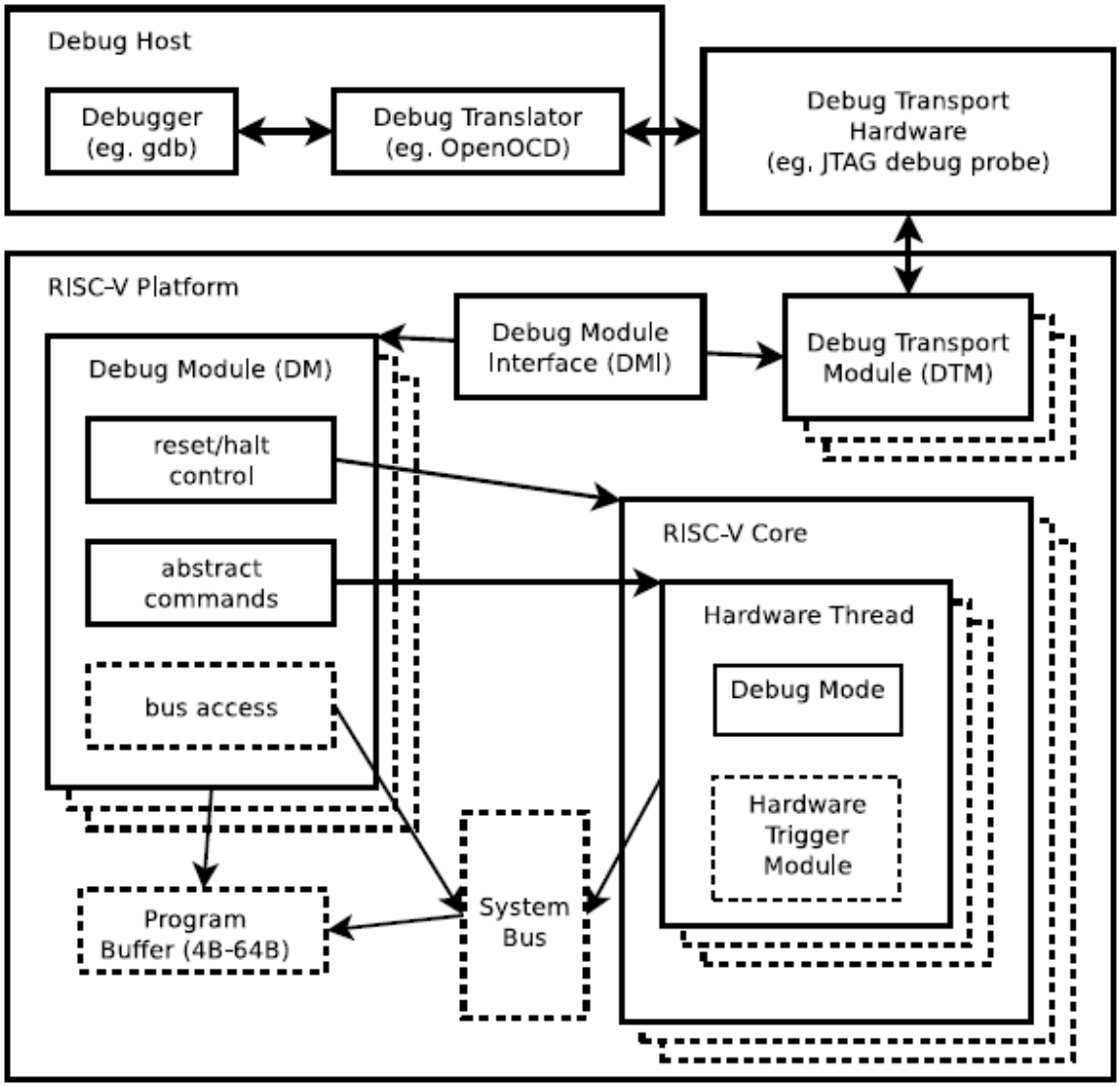


図 2.1：RISC-V デバッグシステムの概要

第 3 章

デバッグモジュール（DM）

デバッグモジュールは、抽象デバッグ操作とそれらの特定の実装との間の変換インタフェースを実装します。
以下の操作をサポートします。

1.実装に関する必要な情報をデバッガに渡します。（必須）
2.個々のハートを停止して再開することを許可します。（必須）
3.どのハートが停止しているかのステータスを入力します。（必須）
4.停止したハートの GPR への抽象的な読み取りおよび書き込みアクセスを提供します。（必須）
5.リセット後の最初の命令からデバッグを可能にするリセット信号へのアクセスを提供します。（必須）
6.（リセットの原因に関係なく）デバッグハートをリセットからすぐに解除できるようにするメカニズムを提供します。（オプション）
7.非 GPR ハートレジスタへの抽象アクセスを提供します。（オプション）
8.ハートに任意の命令を実行させるためのプログラムバッファを用意します。（オプション）
9.複数のハートを同時に停止、再開、またはリセットすることができます。（オプション）
10.ハートの観点からメモリアクセスを許可します。（オプション）
11.システムバスへの直接アクセスを許可します。（オプション）

この仕様に準拠するために、以下の実装を行わなければなりません。：

1.上記の必須機能をすべて実装します。
2.プログラムバッファ、システムバスアクセス、または抽象アクセスメモリのコマンドメカニズムのうち少なくとも 1 つを実装します。
3.少なくとも次のいずれかを行います。
（a）プログラムバッファを実装します。
（b）ハート上に存在し、表 3.3 に記載されているすべてのレジスタを含む、ハート上で実行されているソフトウェアに見えるすべてのレジスタへの抽象アクセスを実装する。
（c）少なくともすべての GPR、dcsr、および dpc への抽象アクセスを実装し、「RISC-V デバッグ仕様 0.13.2」ではなく「最小 RISC-V デバッグ仕様 0.13.2」に準拠していることを宣伝します。

1 つの DM で最大 2^{20} ハートをデバッグできます。

3.1 デバッグモジュールインタフェース（DMI）

デバッグモジュールは、デバッグモジュールインタフェース（DMI）と呼ばれるバスへのスレーブです。
バスのマスタはデバッグトランスポートモジュールです。
デバッグ モジュール インターフェースは、1 つのマスターと 1 つのスレーブを持つ簡単なバスでも、TileLink や AMBA アドバンスド ペリフェラル バスのようなよりフル機能のバスを使用することもできます。
詳細はシステム設計者に任されています。

DMI は 7～32 のアドレスビットを使用します。
読み書き操作をサポートします。
アドレス空間の下部は、最初の（そして通常は唯一の）DM に使用されます。
カスタムデバッグデバイス、他のコア、追加の DM などに追加(余分の)のスペースを使用できます。
この DMI に追加の DMs がある場合は、DMI アドレス空間内の次の DM のベースアドレスが nextdm に示されます。

デバッグモジュールは、その DMI アドレス空間へのレジスタアクセスを介して制御されます。

-- 2019/05/01

3.2 リセット制御

デバッグモジュールはグローバルリセット信号 ndmreset（非デバッグモジュールリセット）を制御します。これは、デバッグモジュールとデバッグトランスポートモジュールを除く、プラットフォーム内のすべてのコンポーネントをリセットするか、リセットを保持します。
実行された最初の命令からプログラムをデバッグすることが可能である限り、正確にこのリセットによって影響を受けるものは実装依存です。
デバッグモジュール自身の状態とレジスタは、電源投入時および dmcontrol の dmactive が 0 の間にのみリセットする必要があります。
トリガー CSRs はクリアされますが、dmactive が 1 であれば、ハートの停止状態はシステムリセットの間中維持されるべきです。

クロックドメインと電源ドメインの交差問題により、システムリセットの間に任意の DMI アクセスを実行することは不可能かもしれません。
ndmreset または(任意の)外部リセットがアサートされている間、サポートされている唯一の DM 操作は dmcontrol へのアクセスです。
他のアクセスの動作は定義されていません。

ndmreset のアサーションの継続期間に関する要件はありません。
実装は、1 への ndmreset の書き込みとそれに続く 0 への ndmreset の書き込みがシステムリセットを引き起こすことを保証しなければなりません。
allunavail、anyunavail が報告しているように、システムがリセットから抜け出すまでには、かなり長い時間がかかることがあります。

個々のハート（または一度に複数のハート）は、それらを選択し、設定してからハートリセットをクリアすることでリセットできます。
この場合、実装は選択されたものより多くのハートをリセットするかもしれません。
デバッガは、他のどのハートがリセットされているか(存在する場合)、それらを選択して anyhavereset と allhavereset をチェックすることによって、発見することができます。

ハートがリセットされると、スティッキーな hasreset 状態ビットを設定する必要があります。
概念的な havereset 状態ビットは、anyhavereset 内の選択されたハートおよび dmstatus 内の allhavereset について読み取ることができます。
これらのビットはリセットの原因に関係なく設定する必要があります。
dmcontrol の ackhavereset に 1 を書き込むことにより、選択したハートのリセットビットをクリアすることができます。
dmactive がローのとき、hasreset ビットはクリアされる場合とされない場合があります。

ハートがリセットから出て、haltreq または resethaltreq が設定されると、ハートは直ちにデバッグモードに入ります。
それ以外の場合は正常に実行されます。

3.3 ハートの選択

1 つの DM に最大 2^{20} ハートを接続できます。
デバッガはハートを選択し、その後の停止、再開、リセット、およびデバッグコマンドはそのハートに固有のものになります。

すべてのハートを列挙するには、デバッガは最初にすべてのものを hartsel に書き込み（最大サイズを想定）、その値を読み戻して実際にどのビットが設定されているかを確認することによって HARTSELLEN を決定する必要があります。
次に、 dmstatus の anynonexistent が 1 になるまで、または（HARTSELLEN に応じて）最も高いインデックスに達するまで、0 から始まる各ハートを選択します。

デバッガは、インタフェースを使用して mhartid を読み取るか、またはシステムの構成文字列を読み取ることによって、ハートインデックスと mhartid の間のマッピングを検出できます。

3.3.1 単一ハートの選択

すべてのデバッグモジュールは単一ハートの選択をサポートしなければなりません。
デバッガは "hartsel"にインデックスを書くことでハートを選択することができます。
ハートインデックスは 0 から始まり、最後のインデックスまで連続しています。

3.3.2 複数のハートを選択する

デバッグモジュールは、一度に複数のハートを選択できるようにハートアレイマスクレジスタを実装することができます。
ハートアレイマスクレジスタの n 番目のビットは、インデックス n のハートに適用されます。
ビットが 1 の場合、ハートが選択されます。
通常、DM には、サポートするすべてのハートを選択するのに十分な幅の Hart Array Mask レジスタがありますが、これらのビットのいずれかを 0 に固定することもできます。

デバッガは、 hawindowsel および hawindow を使用してハート配列マスクレジスタのビットを設定し、次に hasel を設定することによって選択したすべてのハートにアクションを適用できます。
この機能がサポートされている場合は、複数のハートを同時に停止、再開、およびリセットできます。
ハートアレイマスクレジスタの状態は、 hasel の設定またはクリアによる影響を受けません。

dmcontrol によって開始されたアクションのみが一度に複数のハートに適用できます。抽象コマンドは、hartsel によって選択されたハートにのみ適用されます。

3.4 ハート状態

選択できるハートはすべて 4 つの状態のうちの 1 つに属します。
選択されたハートがどの状態にあるかは、
allnonexistent、anynonexistent、allunavail、anyunavail、allrunning、anyrunning、allhalted、および anyhalted によって反映されます。

ユーザーがどれだけ長く待っても、ハートがこのシステムの一部にならない場合、ハートは存在しません。
例えば。 単純なシングルハートシステムでは、ハートは 1 つだけ存在し、それ以外は存在しません。
デバッガは、システムに、最初に存在しないインデックスよりも高いインデックスを持つハートがないと仮定することができます。

ハートが存在する、または後で利用可能になる可能性がある場合、またはこれよりも高いインデックスを持つ他のハートがある場合は、ハートは使用できません。
リセット、一時的な電源切断、システムに接続されていないなど、さまざまな理由でハートが使用できない場合があります。

非常に多数のハートを有するシステムは、製造中に永久的にいくつかを無効にし、そうでなければ連続的なハートインデックススペースに穴を残す可能性がある。
デバッガにすべてのハートを検出させるには、それらが利用可能になる可能性がなくても、それらを利用不可として表示する必要があります。

デバッガが接続されていない場合と同様に、ハートは通常の実行時に実行されています。
これには、停止要求によってハートが停止される限り、低電力モードであること、または割り込みを待つことが含まれます。

デバッグモードにあるとき、ハートは停止され、デバッガに代わってタスクを実行するだけです。

リセットされるハートがどの状態を通過するかは実装に依存します。
リセットがアサートされている間、およびリセット後しばらくしてからハートが使用できなくなる可能性があります。
リセットが解除されてからしばらくの間、実行に移行する可能性があります。
最後に、それらは haltreq と resethaltreq に応じて実行中または停止します。

3.5 実行制御

デバッグモジュールは、ハートごとに 4 つの概念的な状態ビットを追跡します。要求の停止、確認応答の再開、リセットの停止要求、およびハートリセットです。
（ハートトリセットおよびホールトオンリセット要求ビットはオプションです。）
これらの 4 ビットは、0 または 1 にリセットされる可能性がある resume ACK を除いて 0 にリセットされます。
ＤＭは各ハートから停止信号、走行信号、リセット信号を受信します。
デバッガは、 "allresumeack"および "anyresumeack"で resume ack の状態を確認し、 "allhalted"、 "anyhalted"、 "allrunning"、 "anyrunning"、 "allhavereset"、そして「anyhavereset」で停止、実行中、およびリセット信号を確認できます。
他のビットの状態は直接観察することはできません。

デバッガが "haltreq"に 1 を書き込むと、選択された各ハートの停止要求ビットがセットされます。
実行中のハート、またはリセットから出たばかりのハートが停止要求ビットをハイレベルにすると、停止、実行中信号のアサート解除、および停止信号のアサートによって応答します。
停止ハートは停止要求ビットを無視します。

デバッガが "resumereq"に 1 を書き込むと、選択された各ハートの再開 ACK ビットがクリアされ、選択された各停止ハートに再開要求が送信されます。
ハートは、再開し、停止したシグナルをクリアし、実行中のシグナルをアサートすることによって応答します。
このプロセスの終わりに再開確認ビットが設定されます。
選択されたすべてのハートのこれらのステータス信号は、「allresumeack」、「anyresumeack」、「allrunning」、および「anyrunning」に反映されます。
再開要求は、実行注の harts によって無視されます。

停止または再開が要求された場合、ハートは、利用できない場合を除き、1 秒以内に応答しなければなりません。
（これがどのように実装されているかは、さらに詳しく規定されていない。
数クロックサイクルがより典型的な待ち時間になるでしょう）。

DM はハートごとにオプションの halt-on-reset ビットを実装できます。これは、hasresethaltreq を 1 に設定することによって示されます。
これは DM が setresethaltreq と clrresethaltreq ビットを実装することを意味します。
setresethaltreq に 1 を書き込むと、選択したハートごとにリセット停止要求ビットがセットされます。
ハートのリセット停止要求ビットがセットされると、ハートは次のリセット解除時に直ちにデバッグモードに入ります。
これはリセットの原因に関係なく当てはまります。
ハートが選択されている間に clrresethaltreq に 1 を書き込むデバッガによってクリアされるか、または DM リセットによってクリアされるまで、ハートのリセット停止要求ビットはセットされたままになります。

-- 2019/05/06

3.6 抽象コマンド

DM は一連の抽象コマンドをサポートしていますが、そのほとんどはオプション(省略可能)です。
実装によっては、選択したハートが停止していなくても、デバッガは抽象コマンドを実行できる場合があります。
デバッガは、それらを試行してから abstractcs 内の cmderr を調べて、それらが成功したかどうかを確認することによってのみ、特定の状態で特定のハートによってサポートされている抽象コマンドを特定できます。
--とりあえずコマンド投げてみてうまくいったかを cmderr を見て判断 といったところか。
コマンドはいくつかのオプションセットではサポートされていますが、他のオプションセットではサポートされていません。
コマンドにサポートされていないオプションが設定されている場合、DM は cmderr を 2（サポートされていない(サポート対象外)）に設定する必要があります。

例：
すべてのシステムがアクセス登録コマンドをサポートする必要がありますが、CSR へのアクセスをサポートしていない場合があります。
その場合にデバッガが CSR の読み取りを要求した場合、コマンドは "not supported"「サポートされていません」を返します。

デバッガは抽象コマンドを command(コマンド)に書き込むことによって実行します。
abstractcs の busy を読むことで、抽象コマンドが完了したかどうかを判断できます。
完了後、cmderr はコマンドが成功したかどうかを示します。
ハートが停止していない、実行されていない、使用できない、または実行中にエラーが発生したためにコマンドが失敗する可能性があります。

コマンドが引数を取る場合、デバッガは command(コマンド)に書き込む前にそれらをデータレジスタに書き込まなければなりません。
コマンドが結果を返す場合、デバッグモジュールはビジーがクリアされる前にそれらがデータレジスタに配置されていることを確認する必要があります。
引数に使用されるデータレジスタは表 3.1 で説明されています。
すべての場合において、最下位ワードは最も小さい番号のデータレジスタに配置されます。
引数の width(幅)は、実行されているコマンドによって異なり、明示的に指定されていない場合は DXLEN です。

表 3.1：データレジスタの使用

引数の幅	arg0/riturn 値	arg1	arg2
32	data0	data1	data2
64	data0,data1	data2,data3	data4,data5
128	data0-data3	data4-data7	data8-data11

Abstract Command(抽象コマンド)インタフェースは、デバッガができるだけ速くコマンドを記述し、後でそれらがエラーなしで完了したかどうかを確認できるように設計されています。
一般的な場合では、デバッガはターゲットやコマンドが成功するよりもはるかに遅くなり、最大のスループットが可能になります。
--デバッガが遅くなるとスループットが上がるってなんだろ
失敗した場合、インターフェイスは失敗したコマンドの後にコマンドが実行されないようにします(保証します)。
どのコマンドが失敗したかを発見するためには、デバッガは、ＤＭの状態（例えば、データ０の内容）またはハート（例えば、プログラムバッファプログラムによって修正された(変更された)レジスタの内容）を調べてどのコマンドが失敗したかを決定しなければなりません。

抽象コマンドを開始する前に、デバッガは haltreq、resumereq、および ackhavereset がすべて 0 であることを確認する必要があります。

抽象コマンドの実行中（abstractcs でビジー状態がハイ(高い)）は、デバッガが hartsel を変更してはいけません。また、haltreq、resumereq、ackhavereset、setresethaltreq、または clrresethaltreq に 1 を書き込んではいけません。

抽象コマンドが予想される時間内に完了せず、ハングアップしているように見える場合は、次の手順を実行してコマンドを中止することができます。
最初にデバッガはハートをリセットし（hartreset または ndmreset を使用）、次にデバッグモジュールをリセットします（dmactive を使用）。

選択されたハートが使用不可の間に抽象コマンドが開始された場合、または抽象コマンドの実行中にハートが使用不可になった場合、デバッグ・モジュールは抽象コマンドを終了し、ビジーを低く設定し、cmderr を 4（停止/再開）にします。
あるいは、コマンドが単にハングしたように見えることもあります（ビジーが決して低くならない）。

3.6.1 抽象コマンド一覧

-- 2019/06/04

この節では、それぞれ異なる抽象コマンドと、それらが command に書き込まれるときにそれらのフィールドがどのように解釈されるべきかについて説明します。

各抽象コマンドは 32 ビット値です。
上位 8 ビットには、コマンドの種類を決定する cmdtype が含まれています。
表 3.2 に全コマンド一覧を示します。

表 3.2：cmdtype の意味

cmdtype	コマンド	ページ
0	アクセス レジスタ コマンド	12
1	クイック アクセス	14
2	アクセス メモリー コマンド	14

3.6.1.1 アクセスレジスタ

このコマンドはデバッガに CPU レジスタへのアクセスを許可し、プログラムバッファを実行できるようにします。
次の一連の操作を実行します。

1. 書き込みがクリアで転送が設定されている場合は、regno で指定されたレジスタからデータの arg0 領域にデータをコピーし、このレジスタを M モードから読み取るときに発生する副作用を実行します。
2. 書き込みが設定され転送が設定されている場合、データの arg0 領域から regno で指定されたレジスタにデータをコピーし、このレジスタが M モードから書き込まれたときに発生する副作用を実行します。
3. 後部増分(aarpostincrement)が設定されている場合は、regno を増分します。
4. postexec が設定されている場合は、プログラムバッファを実行します。

– side effect って副作用って訳されるんだけど、 副作用を実行ってなんか変だよね。なんかいい訳はないかな。

これらの操作のいずれかが失敗すると、cmderr が設定され、残りの手順は実行されません。
実装は、来るべき失敗を早く検出し、失敗を引き起こすであろうステップに達する前に全体のコマンドを失敗させるかもしれません(失敗させることがあります)。
失敗が要求されたレジスタがハート内に存在しないことである場合、cmderr は 3（例外）に設定されなければなりません。

デバッグモジュールはこのコマンドを実装し、選択されたハートが停止したときにすべての GPR への読み書きアクセスをサポートする必要があります。
デバッグモジュールは、他のレジスタへのアクセス、またはハートの実行中のレジスタへのアクセスをオプションでサポートします(できます)。
(GPRs を除く)各個々のレジスタは、読み取り、書き込み、および停止状態にわたって異なってサポートされる場合があります。

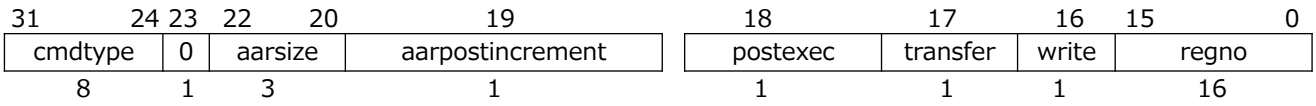
---

aarsize のエンコーディングは sbcs の sbaccess と一致するように選択されました。

このコマンドは、レジスタが読み込まれたときにのみ arg0 を変更します。
他のデータレジスタは変更されません。

表 3.3：抽象レジスタ番号

0x0000 - 0x0fff	GSRs。"PC"は dpc からここにアクセスできます。
0x1000 - 0x101f	GPRs
0x1020 - 0x103f	浮動小数点レジスタ
0xc000 - 0xffff	標準外の拡張機能と内部使用のために予約されています。



フィールド	説明
cmdtype	これは、アクセス登録コマンドを示すために 0 です。
aarsize	2：レジスタの下位 32 ビットにアクセスします。 3：レジスタの最下位 64 ビットにアクセスします。 4：レジスタの下位 128 ビットにアクセスします。 aarsize がレジスタの実際のサイズより大きいサイズを指定している場合、アクセスは失敗します。 レジスタがアクセス可能な場合、レジスタの実際のサイズ以下の aarsize の読み込みがサポートされていなければなりません。 このフィールドは表 3.1 で参照されるように引数の幅を制御します。
aarpostincrement	0：影響なし この変種(変数)はサポートされなければなりません。 1：レジスタアクセスが成功した後、regno がインクリメントされます（0 にラップアラウンド）。 この変種(変数)のサポートはオプションです。 -- variant の訳は変数でよいのかな
postexec	0：影響なし この変種(変数)はサポートされている必要があり、progbufsize が 0 の場合にサポートされる唯一のものです。 1：転送を実行した後、プログラムバッファ内でプログラムを 1 回だけ実行します。 この変種(変数)のサポートはオプションです。
transfer	0：ライトで指定した動作をしません。 1：ライトで指定した動作をします。 このビットは、有効値を aarsize または regno に設定することを心配せずにプログラムバッファを実行するためだけに使用できます。
write	転送設定時： 0：指定レジスタのデータをデータの arg0 部分にコピーします。 1：データの arg0 部分から指定したレジスタにデータをコピーします。
regno	表 3.3 に示すように、アクセスするレジスタの番号。 このコマンドが非停止ハートでサポートされている場合は、dpc を PC のエイリアスとして使用できます(使用されていることがあります)。

--2019/06/08

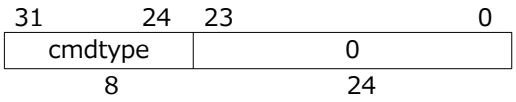
3.6.1.2 クイックアクセス

次の一連の操作を実行してください。

1. ハートが停止した場合、コマンドは cmderr を "halt / resume"に設定し、続行しません。
2. ハートを停止します。ハートが他の理由（ブレークポイントなど）で停止した場合、コマンドは cmderr を "halt / resume" に設定し、続行しません。
3. プログラムバッファを実行します。例外が発生すると、cmderr は "exception"に設定され、プログラムバッファの実行は終了しますが、クイックアクセスコマンドは続行されます。
4. ハートを再開します。

このコマンドの実装はオプションです。

このコマンドはデータレジスタには影響しません。



フィールド	説明
cmdtype	これは、クイックアクセスコマンドを示すための 1 です。

3.6.1.3 メモリアクセス

このコマンドにより、デバッガは選択されたハートとまったく同じメモリビューとパーミッションでメモリアクセスを実行できます。これには、ハートローカルメモリマップレジスタなどへのアクセスが含まれます。
このコマンドは次の一連の操作を実行します。

1. 書き込みがクリアされている場合は、arg1 で指定されたメモリ位置からデータの arg0 部分にデータをコピーします。
2. 書き込みが設定されている場合、データの arg0 部分から arg1 で指定されたメモリ位置にデータをコピーします。
3. aampostincrement が設定されている場合は、arg1 をインクリメントします。

これらの操作のいずれかが失敗すると、cmderr が設定され、残りの手順(ステップ)は実行されません。
M モードコードを実行しているハートが同じアクセスを試みるときに同じ失敗が発生する可能性がある場合にのみ、アクセスが失敗する可能性があります。
実装は、来るべき失敗を早く検出し、失敗を引き起こすであろうステップに達する前に全体のコマンドを失敗させるかもしれません。

デバッグモジュールは任意選択(オプション)でこのコマンドを実装してもよく、選択されたハートが実行中または停止しているときにメモリ位置への読み書きアクセスをサポートされる場合があります。
このコマンドがハートの実行中にメモリアクセスをサポートする場合、ハートが停止している間もメモリアクセスをサポートする必要があります。

---

aamsize のエンコーディングは、sbcs の sbaccess と一致させるために選ばれました。

このコマンドは、メモリが読み込まれるときにのみ arg0 を変更します。
aampostincrement が設定されている場合のみ、arg1 を変更します。
他のデータレジスタは変更されません。

フィールド	説明
cmdtype	これは、Access Memory コマンドを示す 2 です。
aamvirtual	実装は、仮想アクセスと物理アクセスの両方を実装する必要はありませんが、サポートしていないアクセスに失敗する必要があります。 0：アドレスは物理的です（それらが実行されているハードに）。 1：アドレスは仮想的であり、MPRV が設定された状態で M モードからの方法で変換されます。
aamsize	0：メモリ位置の最下位 8 ビットにアクセスします。 1：メモリ位置の最下位 16 ビットにアクセスします。 2：メモリ位置の最下位 32 ビットにアクセスします。 3：メモリ位置の最下位 64 ビットにアクセスします。 4：メモリ位置の最下位 128 ビットにアクセスします。
aampostincrement	メモリアクセスが完了した後、このビットが 1 の場合は、arams1 でエンコードされたバイト数だけ arg1（使用されるアドレスを含む）をインクリメントします。
write	0：arg1 で指定されたメモリ位置からデータの arg0 部分にデータをコピーします。 1：データの arg0 部分から arg1 で指定されたメモリ位置にデータをコピーします。
target-specific	これらのビットは、ターゲット固有の用途に予約されています。

3.7 プログラムバッファ

停止したハート上での任意の命令の実行をサポートするために、デバッグモジュールはデバッガが小さなプログラムを書くことができるプログラムバッファを含むことができます。
抽象コマンドのみを使用して必要なすべての機能をサポートするシステムは、プログラムバッファを省略することを選択できます。
デバッガは小さなプログラムをプログラムバッファに書き込み、次に Access Register Abstract Command(アクセスレジスタ 抽象コマンド)を使用して正確に 1 回実行し、command(コマンド)の postexec ビットを設定できます。
デバッガは自分が好きなプログラム（プログラムバッファからのジャンプを含む）を書くことができますが、プログラムは ebreak または c.ebreak で終わらなければなりません。
実装は、ハートがプログラムバッファの最後から実行されたときに実行される暗黙の突破をサポートする場合があります。
これは impebreak(妨害)によって示されます。
この機能により、たった 2 32 ビットワードのプログラムバッファで効率的なデバッグが可能になります。

progbufsize が 1 の場合、impebreak は 1 でなければなりません。
プログラム バッファが 1 つの 32 ビットまたは 16 ビット命令だけを保持できることが可能で、故にこの場合、デバッガはその寸法に拘らず単一の命令を書くだけでよいです。
この命令は、32 ビット命令、または上位 16 ビットの圧縮 nop を伴う下位 16 ビットの圧縮命令です。

サイズ 1 のプログラムバッファでの少し矛盾する振る舞いは、プログラムバッファがどこかのアドレス空間に存在するのではなく、停止時に直接命令をパイプラインに詰め込むことを好むハードウェア設計に対応することです。

これらのプログラムが実行されている間、ハートはデバッグモードを終了しません（セクション 4.1 を参照）。
プログラムバッファの実行中に例外が発生した場合、それ以上命令は実行されず、ハートはデバッグモードのままになり、cmderr は 3（例外エラー）に設定されます。
デバッガが ebreak 命令で終了しないプログラムを実行すると、ハートはデバッグモードのままになり、デバッガはハートの制御を失います。

プログラムバッファを実行すると、dpc が上書き(痛めつけ)される可能性があります。
-- プログラムバッファを実行って、いまいちピンとこない。バッファって実行するもんなん？
その場合は、postexec が設定されていない抽象コマンドを使用して、dpc を読み書きできるようにする必要があります。
デバッガはプログラムバッファの停止と実行の間に dpc を保存してから、デバッグモードを終了する前に dpc を復元する必要があります。

プログラムバッファの実行を壊しに許可すると、dpc は、別の PC レジスタを持たない直接実装を可能にし、プログラムバッファの実行時に PC を使用する必要があります。
-- clobber って 殴り倒す、痛めつける、みたいな意味みたいだけどどう訳すか。あとここは cpc の前で一回区切るべきか。

プログラムバッファはハートにアクセス可能なＲＡＭとして実現されてもよい(実装することができます)。
デバッガは、プログラムバッファから実行している間に、pc に対して書き込みと読み戻しを試みる小さなプログラムを実行することによって、そうであるかどうかを判断できます。
そうであれば、デバッガはプログラムバッファを使ってできることにもっと柔軟性があります。(プログラムバッファーで実行できる操作がより柔軟になります。)

3.8 状態の概要

図 3.1 は、dmcontrol、abstractcs、abstractauto、および command のさまざまなフィールドの影響を受けた、実行/停止デバッグ中にハートによって渡される状態の概念図を示しています。

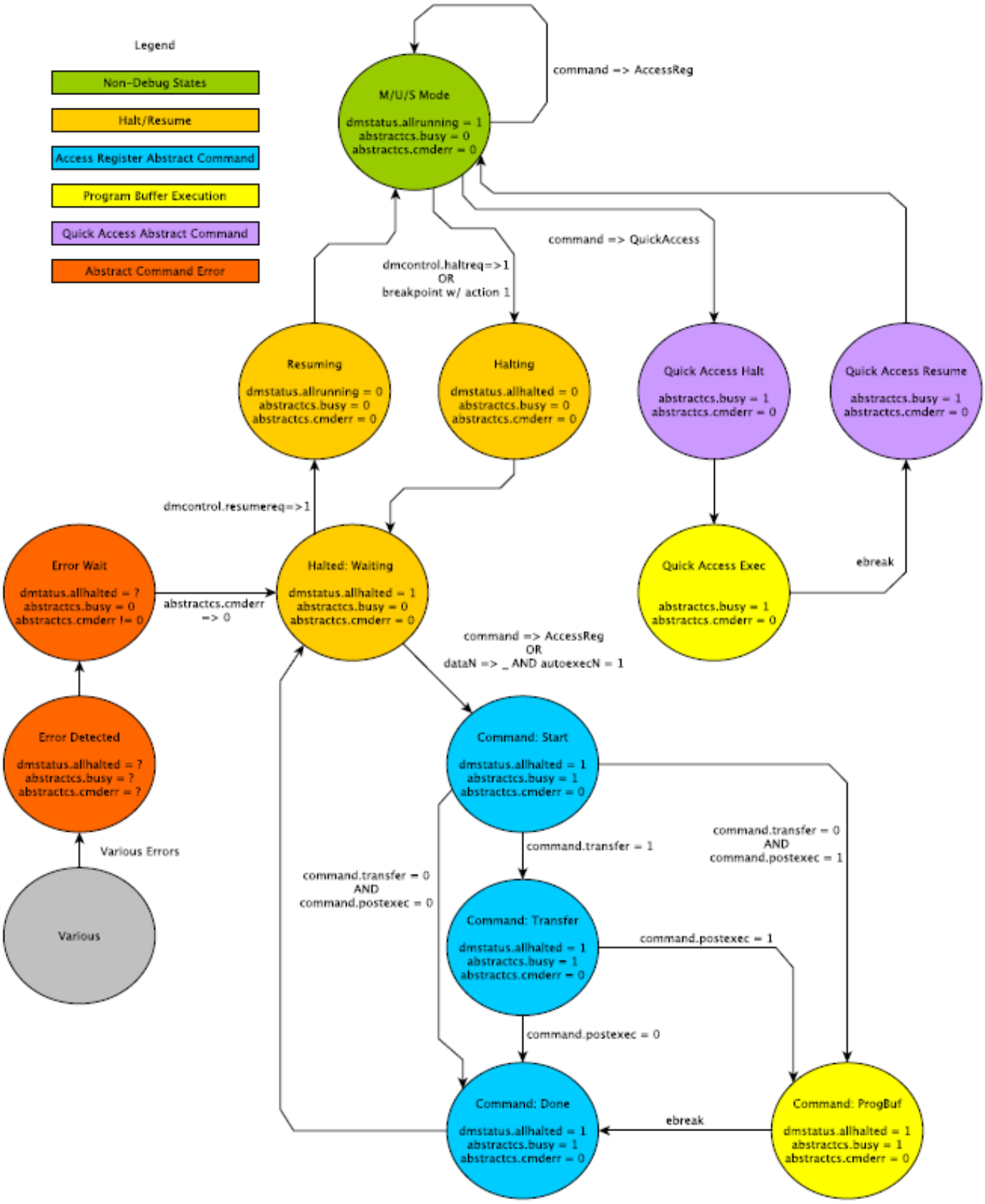


図 3.1 シングルハートトシステム用の実行/停止デバッグステートマシン
ごくわずかな状態しかデバッガに表示されないため、状態と遷移は概念的です。

3.9 システムバスアクセス

デバッガは、プログラムバッファまたは抽象アクセスメモリコマンドを使用して、ハートの観点からメモリにアクセスできます。
（これらの機能はどちらもオプションです。）
デバッグモジュールは、プログラムバッファが実装されているかどうかにかかわらず、ハートを含まずにメモリアクセスを提供するためにシステムバスアクセスブロックを含むこともできます。
システムバスアクセスブロックは物理アドレスを使用します。

System Bus Access ブロックは、8、16、32、64、および 128 ビットのアクセスをサポートします。
表 3.7 に、各アクセスサイズに使用される sbdata のビットを示します。

表 3.7：システムバスデータビット

アクセスサイズ	データビット
8	sbdata0 bits 7:0
16	sbdata0 bits 15:0
32	sbdata0
64	sbdata1, sbdata0
128	sbdata3, sbdata2,sbdata1, sbdata0

マイクロアーキテクチャによっては、システムバスアクセスを介してアクセスされるデータが、各ハートによって観察されるデータと常に一貫しているとは限りません。
実装がそうでない場合、一貫性を強制するのはデバッガ次第です。
この仕様はこれを行うための標準的な方法を定義していません。(定義されていません)
可能性としては、特別なメモリマップロケーションへの書き込み、またはプログラムバッファを介した特別な命令の実行があります。

デバッグモジュールがプログラムバッファも実装する場合でも、System Bus Access ブロックを実装することにはいくつかの利点があります。
まず、実行中のシステム内のメモリに最小限の影響でアクセスできます。
次に、メモリにアクセスするときのパフォーマンスが向上する可能性があります。
第三に、ハートがアクセスできないデバイスへのアクセスを提供する可能性があります。

3.10 最小限の侵入型デバッグ

実行しているタスクに応じて、一部のハートはごく短時間しか停止できません。
稼働中のハートへの影響を最小限に抑えて、そのような稼働中のシステム内のリソースにアクセスできるようにするメカニズムがいくつかあります。

第一に、実装によってはハートを止めることなくいくつかの抽象コマンドを実行することを可能にするかもしれません。

第二に、クイックアクセス抽象コマンドを使用してハートを停止し、プログラムバッファの内容を素早く実行して、ハートを再度実行することができます。
3.12.3 で説明したように、プログラムバッファコードがデータレジスタにアクセスできるようにする命令と組み合わせると、これを使用してメモリまたはレジスタアクセスを迅速に実行できます。
いくつかのシステムではこれはあまりにも邪魔になりますが、停止できない多くのシステムでは時折 100 サイクル以下の一時的な問題が発生する可能性があります。(100 またはそれ以下のサイクルの時折のしゃっくりに耐えることができます。)
-- ここは Google 翻訳と Bing 翻訳でちょっと違う訳をするね。100 サイクル以下の一時的な問題(しゃっくり)に耐える　かな

第三に、システムバスアクセスブロックが実装されている場合は、ハートが実行されている間にシステムメモリにアクセスするために使用できます。

3.11 セキュリティ

知的財産を保護するためには、デバッグモジュールへのアクセスをロックすることが望ましいかもしれません。（場合があります）
その後ではなく製造プロセス中にアクセスできるようにするには、デバッグモジュールにヒューズビットを追加して恒久的に無効にすることができます。
--デバッグが終わったら、ヒューズビットを飛ばしてデバッグモジュールにアクセスできなくする方法もあるよ。ってこと
これはテクノロジ固有のものであるため、この仕様ではこれ以上説明しません。

別の選択肢は、アクセスキーを持っているユーザーだけが DM のロックを解除できるようにすることです。
認証済み、authbusy、および authdata の間で、任意に複雑な認証メカニズムをサポートできます。
認証が明確である場合、DM はプラットフォームの他の部分と対話したり、DM に接続されたハートに関する詳細を公開したりしてはなりません。
以下の必須例外を除いて、すべての DM レジスタは 0 を読み取る必要がありますが、書き込みは無視する必要があります。

1. dmstatus で認証されたことは読み取り可能です。
2. dmstatus の authbusy は読み取り可能です。
3. dmstatus のバージョンは読み取り可能です。
4. dmcontrol の dmactive は読み書き可能です。
5. authdata は読み書き可能です。

3.12 デバッグモジュールレジスタ

このセクションで説明されているレジスタは DMI バスを介してアクセスされます。
各 DM は基本アドレス（最初の DM の場合は 0）を持ちます。
以下のレジスタアドレスは、このベースアドレスからのオフセットです。

読み込まれると、未実装のデバッグモジュール DMI レジスタは 0 を返します。
書いても効果はありません。

各レジスタについて、それを読み、ゼロでない値（例えば、sbcs）を得ることによって、または他のレジスタ内のビットをチェックすること（例えば、progbufsize）によってそれが実施されていることを決定することが可能です。

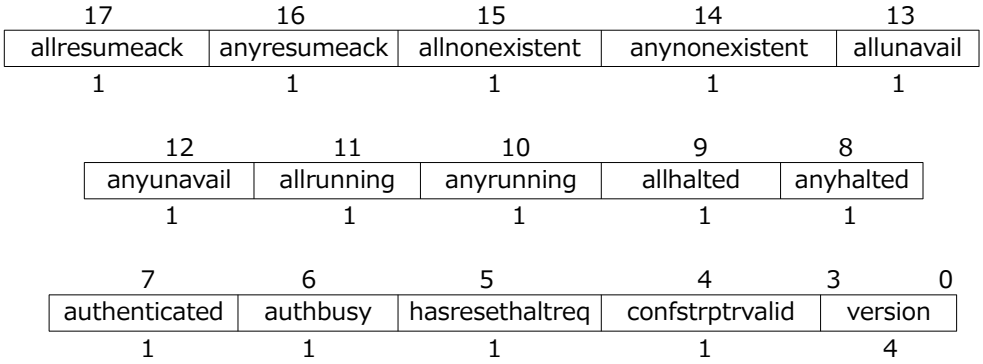
表 3.8 : デバッグモジュールデバッグバスレジスタ

アドレス	名前	ページ
0x04	抽象データ 0（data0）	30
0x0f	要約データ 11（data11）	
0x10	デバッグモジュール制御（dmcontrol）	22
0x11	デバッグモジュールステータス（dmstatus）	20
0x12	ハート情報（hartinfo）	25
0x13	停止要約 1（haltsum1）	31
0x14	ハートアレイウィンドウ選択（hawindowsel）	26
0x15	ハートアレイウィンドウ（hawindow）	26
0x16	抽象制御とステータス（abstractcs）	27
0x17	抽象コマンド（command）	28
0x18	抽象コマンド Autoexec（abstractauto）	29
0x19	設定文字列ポインタ 0（confstrptr0）	29
0x1a	設定文字列ポインタ 1（confstrptr1）	
0x1b	設定文字列ポインタ 2（confstrptr2）	
0x1c	設定文字列ポインタ 3（confstrptr3）	
0x1d	次のデバッグモジュール（nextdm）	30
0x20	プログラムバッファ 0（progbuf0）	30
0x2f	プログラムバッファ 15（progbuf15）	
0x30	認証データ（authdata）	31
0x34	停止要約 2（haltsum2）	32
0x35	停止要約 3（haltsum3）	32
0x37	システムバスアドレス 127 : 96（sbaddress 3）	36
0x38	システムバスアクセス制御とステータス（sbcs）	32
0x39	システムバスアドレス 31 : 0（sbaddress0）	34
0x3a	システムバスアドレス 63:32（sbaddress1）	35
0x3b	システムバスアドレス 95:64（sbaddress 2）	35
0x3c	システムバスデータ 31 : 0（sbdata0）	36
0x3d	システムバスデータ 63:32（sbdata1）	37
0x3e	システムバスデータ 95:64（sbdata2）	37
0x3f	システムバスデータ 127 : 96（sbdata3）	38
0x40	停止要約 0（haltsum0）	31

3.12.1 デバッグモジュールステータス（dmstatus、0x11）

このレジスタは、haselで定義されているように、デバッグモジュール全体と現在選択されているハートのステータスを報告します。
それはバージョンを含んでいるので、このアドレスは将来変更されません。
このレジスタ全体は読み取り専用です。

31 23	22	21 20	19	18
0	impebreak	0	allhavereset	anyhavereset
9	1	2	1	1



領域	説明	アクセス	リセット
impebreak	1 の場合、プログラムバッファの直後の存在しないワードに暗黙的な ebreak 命令があります。 これにより、デバッガが ebreak 自体を記述する必要がなくなり、プログラムバッファを 1 ワード小さくすることができます。 progbufsize が 1 のとき、これは 1 でなければなりません。	R	プリセット
allhavereset	このフィールドは、現在選択されているすべてのハートがリセットされ、リセットが確認されていない場合は 1 です。	R	-
anyhavereset	このフィールドは、少なくとも 1 つの現在選択されているハートがリセットされており、そのハートトについてリセットが確認されていない場合は 1 です。	R	-
allresumeack	このフィールドは、現在選択されているすべてのハートが最後の再開要求を承認したときに 1 になります。	R	-
anyresumeack	このフィールドは、現在選択されているハートが最後の再開要求を承認したときに 1 になります。	R	-
allnonexistent	このフィールドは、現在選択されているすべてのハートがこのプラットフォームに存在しない場合、1 です。	R	-
anynonexistent	このフィールドは、現在選択されているハートがこのプラットフォームに存在しない場合、1 です。	R	-
allunavail	このフィールドは、現在選択されているすべてのハートが使用不可の場合、1 です。	R	-
anyunavail	このフィールドは、現在選択されているハートが使用できない場合、1 です。	R	-
allrunning	このフィールドは、現在選択されているすべてのハートが実行されている場合、1 です。	R	-
anyrunning	このフィールドは、現在選択されているハートが実行中の場合、1 です。	R	-
allhalted	このフィールドは、現在選択されているすべてのハートが停止している場合、1 です。	R	-
anyhalted	このフィールドは、現在選択されているハートが停止している場合、1 です。	R	-

次のページに続く

領域	説明	アクセス	リセット
authenticated	0：DM を使用する前に認証が必要です。 1：認証チェックに合格しました。 認証を実装していないコンポーネントでは、このビットを 1 にプリセットする必要があります。	R	プリセット
authbusy	0：認証モジュールは、authdata への次の読み取り/書き込みを処理する準備ができています。 1：認証モジュールはビジーです。authdata にアクセスすると不特定の動作になります。 authbusy は、authdata へのアクセスに即時に応答して設定されるだけです。	R	0
hasresethaltreq	このデバッグモジュールが setresethaltreq および clrresethaltreq ビットで制御可能なリセット時停止機能をサポートする場合は 1 です。 それ以外の場合は 0 です。	R	プリセット
confstrptrvalid	0：confstrptr0 - confstrptr3 は設定文字列に関係のない情報を保持します。 1：confstrptr0 - confstrptr3 は設定文字列のアドレスを保持します。	R	プリセット
version	0：デバッグモジュールが存在しません。 1：デバッグモジュールがあり、それはこの仕様のバージョン 0.11 に準拠しています。 2：デバッグモジュールがあり、それはこの仕様のバージョン 0.13 に準拠しています。 15：デバッグモジュールはありますが、この仕様の利用可能なバージョンには準拠していません。	R	2

3.12.2 デバッグモジュール制御（dmcontrol、0x10）

-- 2019/06/09

このレジスタは、hasel で定義されているように、全体のデバッグモジュールと現在選択されているハートを制御します。

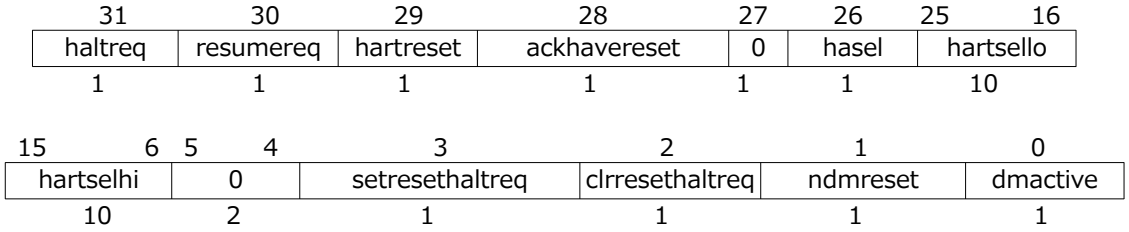
この文書では、hartsel と hartsello を組み合わせた hartsel について説明します。
この仕様では 20 ハートセルビットが許可されています(使用できます)が、実装ではそれより少ない実装を選択することもできます。
ハートセルの実際の幅は HARTSELLEN と呼ばれます。
それは 0 以上 20 以下でなければなりません。
デバッガは（最大サイズを仮定して）すべて 1 を hartsel に書き込み、どのビットが実際に設定されているかを確認するために値を読み返すことによって HARTSELLEN を発見するはずです。(検出する必要があります)
抽象コマンドの実行中にデバッガが hartsel を変更してはいけません。

---

setresethaltreq と clrresethaltreq のビットが別々になっているため、選択したすべてのハートが同じ設定ではない場合でも、選択したハートごとにリセット停止要求ビットを変更せずに dmcontrol を書き込むことができます。

どのような書き込みでも、デバッガは resumereq、hartreset、ackhavereset、setresethaltreq、および clrresethaltreq のうち最大 1 つのビットに 1 を書き込むことしかできません。
他のものは 0 と書く必要があります。

resethaltreq はオプションのパーハート状態の内部ビットで、読み取ることはできませんが、setresethaltreq および clrresethaltreq を使用して書き込むことができます。



領域	説明	アクセス	リセット
haltreq	0 を書き込むと、現在選択されているすべてのハートトの停止要求ビットがクリアされます。 これはそれらのハートのための未解決の停止要求をキャンセルするかもしれません。 1 を書き込むと、現在選択されているすべてのハートの停止要求ビットがセットされます。 実行停止ハートビートは、停止要求ビットが設定されるたびに停止します。 記事は、hartsel と hasel の新しい価値に適用されます。	W	-
resumereq	1 を書き込むと、現在選択されているハートが 1 回再開します（書き込みが発生したときに停止している場合）。 それはまたそれらのハートの履歴書 ack ビットをクリアします。 haltreq が設定されている場合、resumereq は無視されます。 書き込みは、hartsel と hasel の新しい価値に適用されます。	W1	-
hartreset	このオプションのフィールドは、現在選択されているすべてのハートのリセットビットを書き込みます。 リセットを実行するには、デバッガは 1 を書き込み、次に 0 を書き込んでリセット信号をディアサートします。 このビットが 1 の間、デバッガは選択されているハートを変更してはいけません。 この機能が実装されていない場合、ビットは常に 0 のままであるため、1 を書き込んだ後にデバッガはレジスタを読み戻して機能がサポートされているかどうかを確認できます。 記事は、hartsel と hasel の新しい価値に適用されます。	R/W	0
ackhavereset	0：影響ありません。 1：選択したハートのリセットをクリアします。 記事は、hartsel と hasel の新しい価値に適用されます。	W1	–

次のページに続く

領域	説明	アクセス	リセット
hasel	現在選択されているハートの定義を選択します。 0：現在選択されているハートが 1 つあり、それは hartsel によって選択されています。 1：現在選択されているハート（hartsel によって選択されたハートと、ハート配列マスクレジスタによって選択されたハート）が複数存在する可能性があります。 ハート・配列・マスク・レジスターをインプリメントしないインプリメンテーションは、このフィールドを 0 に結合しなければなりません。 ハート・配列・マスク・レジスター機能を使用したいデバッガーは、このビットを設定して、機能がサポートされているかどうかを確認するために読み返す必要があります。	R/W	0
hartsello	hartsel の下位 10 ビット：選択するハートの DM 固有のインデックス。 このハートは常に現在(常時)選択されているハートの一部です。	R/W	0
hartselhi	ハートセルの上位 10 ビット：選択するハートの DM 固有のインデックス。 このハートは常に現在選択されているハートの一部です。	R/W	0
setresethaltreq	このオプションのフィールドは、clrresethaltreq が同時に 1 に設定されていない限り、現在選択されているすべてのハートのリセット時停止要求ビットを書き込みます。 1 に設定されると、選択された各ハートは、次のリセット解除時に停止します。 リセット停止要求ビットは自動的にはクリアされません。 デバッガはそれをクリアするために clrresethaltreq に書き込む必要があります。 書き込みは、hartsel と hasel の新しい価値に適用されます。 hasresethaltreq が 0 の場合、このフィールドは実装されていません。	W1	-
clrresethaltreq	このオプションのフィールドは、現在選択されているすべてのハートのリセット停止要求ビットをクリアします。 記事は、hartsel と hasel の新しい価値に適用されます。	W1	-
ndmreset	このビットは、DM からシステムの他の部分へのリセット信号を制御します。 DM と DM へのアクセスに必要なロジックを除いて、信号はすべてのハートを含むシステムのすべての部分をリセットする必要があります。 システムリセットを実行するには、デバッガは 1 を書き込み、次に 0 を書き込んでリセットをディアサートします。	R/W	0

次のページに続く

領域	説明	アクセス	リセット
dmactive	このビットは、デバッグモジュール自体のリセット信号として機能します。 0：認証メカニズムを含むモジュールの状態はリセット値を取ります（dmactive ビットはリセット値以外に書き込むことができる唯一のビットです）。 1：モジュールは正常に機能しています。 プラットフォーム全体をリセットするグローバルリセット信号を除いて（可能ではありませんが(推奨されませんが)）例外として、電源投入後にデバッグモジュールをリセットする可能性のあるメカニズムは他にはありません。 デバッガはこのビットを Low にしてデバッグモジュールを既知の状態にすることができます。 実装がこのビットに注意を払ってデバッグをさらに支援することができます。たとえば、デバッグがアクティブな間はデバッグモジュールがパワーゲーティングされないようにするなどです。	R/W	0

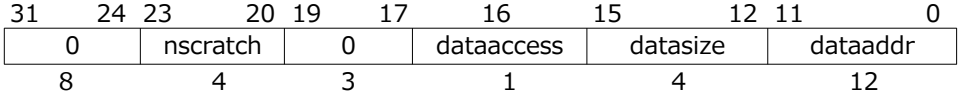
3.12.3 ハート情報（hartinfo、0x12）

このレジスタは、現在 hartsel によって選択されているハートに関する情報を提供します。

このレジスタはオプションです。
存在しない場合は、オールゼロと読みます。(all-0 を読み取る必要があります

このレジスタが含まれている場合、デバッガはデータレジスタまたは dscratch レジスタ、あるいはその両方に明示的にアクセスするプログラムを書くことによってプログラムバッファを使ってより多くのことができます。

このレジスタ全体は読み取り専用です。



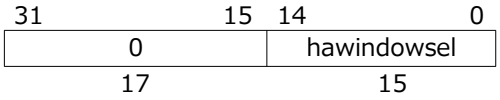
領域	説明	アクセス	リセット
nscratch	dscratch0 から始まる プログラムバッファの実行中にデバッガが使用できる dscratch レジスタの数。 デバッガは、コマンド間でこれらのレジスタの内容について想定することはできません。	R	プリセット

次のページに続く

領域	説明	アクセス	リセット
dataaccess	0：データレジスタは CSRs によってハート内でシャドウされます。 　各 CSRs は DXLEN ビットのサイズで、表 3.1 に従って単一の引数に対応します。 1：データレジスタはハートのメモリマップでシャドウされています。 　各レジスタはメモリマップ内で 4 バイトを占有します。	R	プリセット
datasize	dataaccess が 0 の場合：データレジスタのシャドウイング専用の CSRs の数。 dataaccess が 1 の場合：データレジスタのシャドーイング専用のメモリマップ内の 32 ビットワード数。 最大 12 個のデータレジスタがあるので、このレジスタの値は 12 以下でなければなりません。	R	プリセット
dataaddr	dataaccess が 0 の場合：データレジスタのシャドウイング専用の最初の CSR の番号(数)。 dataaccess が 1 の場合：データレジスタがシャドウされている RAM の符号付きアドレスで、0 を基準にしたアクセスに使用されます。	R	プリセット

3.12.4 ハートアレイウィンドウ選択（hawindowsel、0x14）

このレジスタは、ハート配列マスクレジスタ（セクション 3.3.2 を参照）の 32 ビット部分のどれが hawindow でアクセス可能かを選択します。(32 ビット部分のうち、どの部分にアクセスできるかを選択します。)

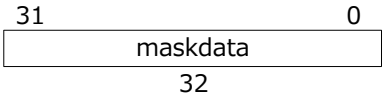


領域	説明	アクセス	リセット
hawindowsel	配列マスクレジスタの大きさによっては、このフィールドの上位ビットを 0 に固定することができます。 例えば。 48 ハートのシステムでは、このフィールドのビット 0 だけが実際に書き込み可能かもしれません。	R/W	プリセット

3.12.5 ハート配列ウィンドウ（hawindow、0x15）

このレジスタはハート・配列・マスク・レジスタの 32 ビット部分への R/W アクセスを提供します（3.3.2 項を参照）。
ウィンドウの位置は hawindowsel によって決定されます。
すなわち ビット 0 はハートの hawowowsel * 32 を表し、ビット 31 はハートの hawindowsel * 32 + 31 を表します。

ハート・アレイ・マスク・レジスター内のいくつかのビットは定数 0 になることがあるので、このウィンドウ内のいくつかのビットは、hawindowsel の現在の値に応じて、定数 0 になることがあります。

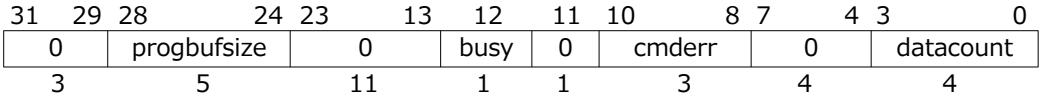


3.12.6 抽象制御とステータス（abstractcs、0x16）

抽象コマンドの実行中にこのレジスタに書き込むと、cmderr が 0 の場合、1（ビジー）に設定されます。

---

datacount は、RV32 ハートをサポートするために少なくとも 1、RV64 ハートをサポートするために 2、または RV128 ハートをサポートするために 4 でなければなりません。



領域	説明	アクセス	リセット
progbufsize	プログラムバッファのサイズ（32 ビットワード）。 有効なサイズは 0 から 16 です。	R	プリセット
busy	1：抽象コマンドを実行中です。 このビットは、コマンドが書き込まれるとすぐに設定され、そのコマンドが完了するまでクリアされません。	R	0

次のページに続く

領域	説明	アクセス	リセット
cmderr	抽象コマンドが失敗した場合に設定されます。 このフィールドのビットは、1 を書き込むことによってクリアされるまでセットされたままです。 値が 0 にリセットされるまで、抽象コマンドは開始されません。 このフィールドは、busy が 0 の場合にのみ有効な値を含みます。 0（なし）：エラーなし 1（ビジー）：command、abstractcs、または abstractauto が書き込まれている間、または data または progbuf レジスタのいずれかが読み取られているか書き込まれているときに、抽象コマンドが実行されていました。 このステータスは、cmderr に 0 が含まれている場合にのみ書き込まれます。 2（未サポート）：ハートが実行中かどうかにかかわらず、要求されたコマンドはサポートされていません。 3（例外）：コマンド実行中（プログラムバッファ実行中など）に例外が発生しました。 4（停止/再開）：ハートが要求された状態（実行中/停止）にないか、または使用不可のため、抽象コマンドを実行できませんでした。 5（bus）：バスエラー（アライメント、アクセスサイズ、タイムアウトなど）のため抽象コマンドが失敗しました。 7（その他）：コマンドが別の理由で失敗しました。	R/W1C	0
datacount	抽象コマンドインタフェースの一部として実装されているデータレジスタの数。 有効なサイズは 1 ～ 12 です。	R	プリセット

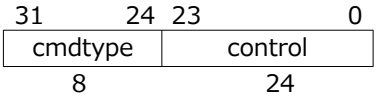
3.12.7 抽象コマンド（command、0x17）

このレジスタに書き込むと、対応する抽象コマンドが実行されます。
抽象コマンドの実行中にこのレジスタに書き込むと、cmderr が 0 の場合、1（ビジー）に設定されます。

cmderr がゼロ以外の場合、このレジスタへの書き込みは無視されます。

---

cmderr は、パフォーマンス上の理由から、cmderr をチェックせずに連続して実行されるいくつかのコマンドを送信するデバッガに対応するための新しいコマンドの起動を禁止します。
1 つのコマンドが失敗したことを心配せずに最後の cmderr をチェックし、その後のコマンド（前のコマンドに依存していた可能性があります）は通過しました。

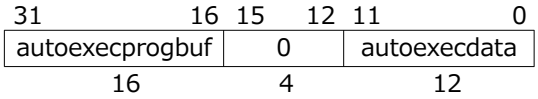


領域	説明	アクセス	リセット
cmdtype	型はこの抽象コマンドの全体的な機能を決定します。	W	0
control	このフィールドは、各抽象コマンドについて説明されているコマンド固有の方法で解釈されます。	W	0

3.12.8 抽象コマンド Autoexec（abstractauto、0x18）

このレジスタはオプションです。
これを含めると、より効率的なバーストアクセスが可能になります。
デバッガは、ビットを設定して読み戻すことで、サポートされているかどうかを検出できます。

抽象コマンドの実行中にこのレジスタに書き込むと、cmderr が 0 の場合、1（ビジー）に設定されます。



領域	説明	アクセス	リセット
autoexecprogbuf	このフィールドのビットが 1 の場合、対応する progbuf ワードへの読み書きアクセスによって command 内のコマンドが再度実行されます。	R/W	0
autoexecdata	このフィールドのビットが 1 の場合、対応するデータワードへの読み書きアクセスによって、command 内のコマンドが再度実行されます。	R/W	0

3.12.9 設定文字列ポインタ 0（confstrptr0、0x19）

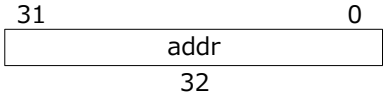
confstrptrvalid が設定されている場合、このレジスタを読み取ると設定文字列ポインタのビット 31：0 が返されます。
他の confstrptr レジスタを読み取ると、アドレスの上位ビットが返されます。

システムバスマスタリングが実装されている場合、これはシステムバスアクセスモジュールで使用できるアドレスでなければなりません。
そうでなければ、これは ID 0 のハートから構成ストリング(文字列)にアクセスするために使用できるアドレスでなければなりません。

confstrptrvalid が 0 の場合、confstrptr レジスタはこの文書ではこれ以上指定されていない識別子情報を保持します。

設定文字列自体は特権仕様に記述されています。

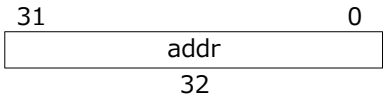
このレジスタ全体は読み取り専用です。



3.12.10 次のデバッグモジュール（nextdm、0x1d）

この DMI でアクセス可能な DM が複数ある場合、このレジスタにはチェーン内の次の DM のベースアドレスが格納されます。チェーン内の最後の DM の場合は 0 が格納されます。

このレジスタ全体は読み取り専用です。



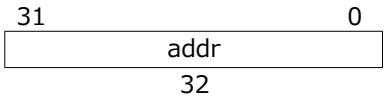
3.12.11 抽象データ 0（data0、0x04）

data0 から data11 は、抽象コマンドによって読み取りまたは変更できる基本的な読み取り/書き込みレジスタです。
datacount は、data0 から数えて、それらがいくつ実装されているかを示します。(data0 から始まって、カウントアップされた実装の数を示します。)
表 3.1 に、抽象コマンドによるこれらのレジスタの使用方法を示します。

抽象コマンドの実行中にこれらのレジスタにアクセスすると、cmderr が 0 の場合、1（ビジー）に設定されます。

busy が設定されている間にそれらを書き込もうとしても、それらの値は変わりません。

これらのレジスタの値は、抽象コマンドの実行後に保存されない可能性があります。
その内容に対する唯一の保証は、問題のコマンドによって提供されるものです。
コマンドが失敗した場合、これらのレジスタの内容については想定できません。

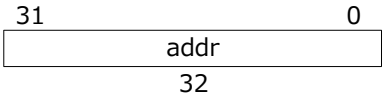


3.12.12 プログラムバッファ 0（progbuf0、0x20）

progbuf0 から progbuf15 は、オプションのプログラムバッファへの読み取り/書き込みアクセスを提供します。
progbufsize は、カウントアップしながら、progbuf0 からいくつ実装されるかを示します。(それらの多くが progbuf0 から始まり、カウントアップを示します。)

抽象コマンドの実行中にこれらのレジスタにアクセスすると、cmderr が 0 の場合、1（ビジー）に設定されます。

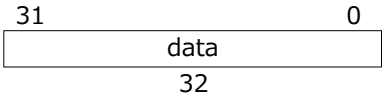
busy が設定されている間にそれらを書き込もうとしても、それらの値は変わりません。



3.12.13 認証データ（authdata、0x30）

このレジスタは、認証モジュールとの間の 32 ビットシリアルポートとして機能します。

authbusy が明確な(クリアな)場合、デバッガはこのレジスタを読み取りもしくは書き込みすることで認証モジュールと通信できます。
オーバーフロー/アンダーフローを通知するための別(個別)のメカニズムはありません。

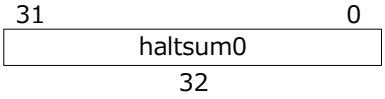


3.12.14 停止 サマリー 0（haltsum0、0x40）

この読み出し専用レジスタの各ビットは、特定の 1 つが停止しているかどうかを示します。
利用できない/存在しないハートは中止されたとは見なされません。

LSB はハート {hartsel [19：5]、5'h0}の停止状態を反映し、MSB は hart {hartsel [19：5]、5'h1f}の停止状態を反映します。

このレジスタ全体は読み取り専用です。

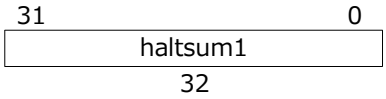


3.12.15 停止サマリー 1（haltsum1、0x13）

この読み出し専用レジスタの各ビットは、ハートグループのいずれかが停止しているかどうかを示します。
利用できない/存在しないハートは停止されたとは見なされません。

このレジスタは、33 ハート未満のシステムには存在しない可能性があります。
LSB はハート{hartsel [19:10]、10'h0}から{hartsel [19:10]、10'h1f}までの停止状態を反映しています。
MSB は、ハートの停止状況{hartsel [19:10]、10'h3e0}から{hartsel [19:10]、10'h3ff}を反映しています。

このレジスタ全体は読み取り専用です。

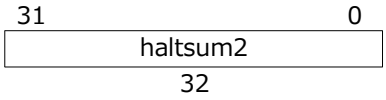


3.12.16 停止サマリー 2（haltsum2、0x34）

この読み出し専用レジスタの各ビットは、ハートグループのいずれかが停止しているかどうかを示します。
利用できない/存在しないハートは中止されたとは見なされません。

このレジスタは 1025 ハート未満のシステムには存在しないかもしれません。

LSB はハートの停止ステータス{hartsel [19:15]、15'h0}から{hartsel [19:15]、15'h3ff}を反映しています。
MSB はハートの停止ステータス{hartsel [19:15]、15'h7c00}から{hartsel [19:15]、15'hfff}を反映しています。
このレジスタ全体は読み取り専用です。

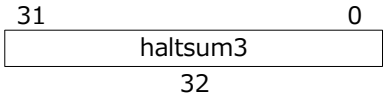


3.12.17 停止サマリー 3（haltsum3、0x35）

この読み出し専用レジスタの各ビットは、ハートグループのいずれかが停止しているかどうかを示します。
利用できない/存在しないハートは中止されたとは見なされません。

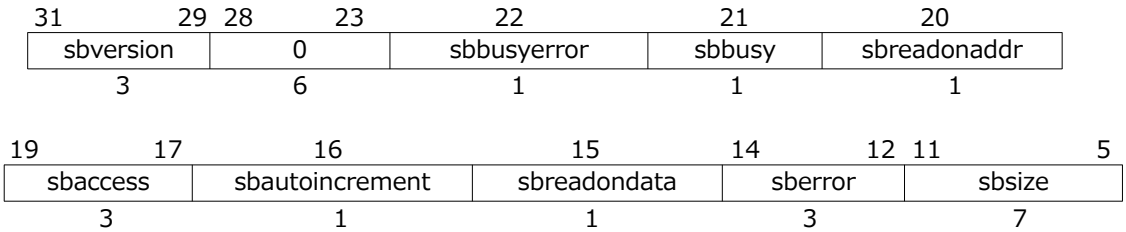
このレジスタは、32769 ハート未満のシステムには存在しない可能性があります。

LSB は、ハート 20'h0 から 20'h7fff の停止状況を反映しています。
MSB は、ハート 20'hf8000 から 20'hfffff の停止状況を反映しています。
このレジスタ全体は読み取り専用です。



3.12.18 システムバスのアクセス制御とステータス（sbcs、0x38）

-- 2019/06/16



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

フィールド	説明	アクセス	リセット
sbversion	0：システムバスインターフェイスは、2018 年 1 月 1 日よりも古いこの仕様のメインラインドラフトに準拠しています。 1：システムバスインターフェイスは、このバージョンの仕様に準拠しています。 他の値は将来のバージョン用に予約されています。	R	1
sbbusyerror	読み取りの進行中にデバッガーがデータの読み取りを試行した場合、または既に進行中のデバッガーが新しいアクセスを開始した場合（sbbusy が設定されている場合）に設定します。 デバッガーによって明示的にクリアされるまで、設定されたままです。 このフィールドが設定されている間、デバッグモジュールはシステムバスアクセスを開始できません。	R/W1C	0
sbbusy	1 の場合、システムバスマスターがビジーであることを示します。（システムバス自体がビジーであるかどうかは関連していますが、同じことではありません。） このビットは、何らかの理由で読み取りまたは書き込みが要求されるとすぐにハイになり、アクセスが完全に完了するまでローになりません。 sbbusy が高いときに sbcs に書き込むと、未定義の動作が発生します。 デバッガーは、sbbusy を 0 として読み取るまで sbcs に書き込まないでください。	R	0
sbreadonaddr	1 の場合、sbaddress0 への書き込みごとに、新しいアドレスでシステムバスの読み取りが自動的にトリガーされます。	R/W	0
sbaccess	システムバスアクセスに使用するアクセスサイズを選択します。 0：8 ビット 1：16 ビット 2：32 ビット 3：64 ビット 4：128 ビット DM がバスアクセスを開始するときに sbaccess にサポートされていない値がある場合、アクセスは実行されず、sberror は 4 に設定されます。	R/W	2
sbautoincrement	1 の場合、sbaddress は、システムバスにアクセスするたびに sbaccess で選択されたアクセスサイズ（バイト単位）ずつ増加します。	R/W	0
sbreadondata	1 の場合、sbdata0 からの読み取りごとに、システムバスの読み取りが（おそらく自動インクリメントされた）アドレスで自動的にトリガーされます。	R/W	0

次のページに続く

-- 2019/07/29

フィールド	説明	アクセス	リセット
sberror	デバッグモジュールのシステムバスマスターでエラーが発生すると、このフィールドが設定されます。 このフィールドのビットは、1 を書き込んでクリアされるまで設定されたままです。 このフィールドはゼロではありませんが、デバッグモジュールはこれ以上システムバスアクセスを開始できません。 実装は、エラー条件について「その他」（7）を報告する場合があります。(報告できます) 0：バスエラーはありませんでした。 1：タイムアウトがありました。 2：不正なアドレスにアクセスしました。 3：位置合わせエラーがありました。 4：サポートされていないサイズのアクセスが要求されました。 7：その他。	R/W1C	0
sbasize	システムバスアドレスの幅（ビット単位）。 （0 はバスアクセスサポートがないことを示します。）	R	事前設定
sbaccess128	128 ビットシステムバスアクセスがサポートされている場合は 1。	R	事前設定
sbaccess64	64 ビットシステムバスアクセスがサポートされている場合は 1。	R	事前設定
sbaccess32	32 ビットシステムバスアクセスがサポートされている場合は 1。	R	事前設定
sbaccess16	16 ビットシステムバスアクセスがサポートされている場合は 1。	R	事前設定
sbaccess8	8 ビットシステムバスアクセスがサポートされている場合は 1。	R	事前設定

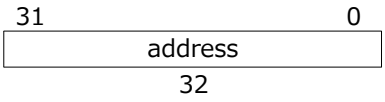
3.12.19 システムバスアドレス 31:0（sbaddress0、0x39）

sbasize が 0 の場合、このレジスタは存在しません。

システムバスマスタがビジーの場合、このレジスタへの書き込みは sbbusyerror を設定し、他には何もしません。

sberror が 0、sbbusyerror が 0、sbreadonaddr が設定されている場合、このレジスタへの書き込みは以下を開始します。

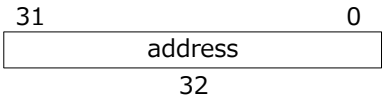
1. sbbusy を設定します。
2. sbaddress の新しい値からバス読み取りを実行します。
3.読み取りが成功し、sbautoincrement が設定されている場合、sbaddress を増やします。
4. sbbusy をクリアします。



フィールド	説明	アクセス	リセット
address	sbaddress の物理アドレスのビット 31：0 にアクセスします。	R/W	0

3.12.20 システムバスアドレス 63:32（sbaddress1、0x3a）

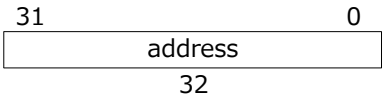
sbasize が 33 未満の場合、このレジスタは存在しません。
システムバスマスタがビジーの場合、このレジスタへの書き込みは sbbusyerror を設定し、他には何もしません。



フィールド	説明	アクセス	リセット
address	sbaddress の物理アドレスのビット 63:32 にアクセスします（システムアドレスバスがその幅の場合）。	R/W	0

3.12.21 システムバスアドレス 95:64（sbaddress2、0x3b）

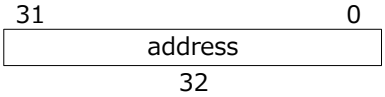
sbasize が 65 未満の場合、このレジスタは存在しません。
システムバスマスタがビジーの場合、このレジスタへの書き込みは sbbusyerror を設定し、他には何もしません。



フィールド	説明	アクセス	リセット
address	sbaddress の物理アドレスのビット 95:64 にアクセスします（システムアドレスバスがその幅の場合）。	R/W	0

3.12.22 システムバスアドレス 127：96（sbaddress3、0x37）

sbasize が 97 未満の場合、このレジスタは存在しません。
システムバスマスタがビジーの場合、このレジスタへの書き込みは sbbusyerror を設定し、他には何もしません。



フィールド	説明	アクセス	リセット
address	sbaddress の物理アドレスのビット 127:96 にアクセスします（システムアドレスバスがその幅の場合）。	R/W	0

3.12.23 システムバスデータ 31:0（sbdata0、0x3c）

sbcs の sbaccess ビットがすべて 0 の場合、このレジスタは存在しません。

システムバスの読み取りが成功すると、sbdata が更新されます。
読み取りアクセスの幅が sbdata の幅より小さい場合、残りの上位ビットの内容は任意の値をとることがあります。

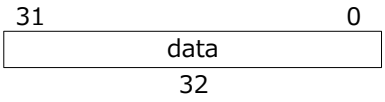
sberror または sbbusyerror の両方が 0 でない場合、アクセスは何もしません。

バスマスタがビジーの場合は、sbbusyerror にアクセスし、他には何もしません。

このレジスタへの書き込みにより、以下が開始されます。
1. sbbusy を設定します。
2. sbdata の新しい値のバス書き込みを sbaddress に実行します。
3.書き込みが成功し、sbautoincrement が設定されている場合、sbaddress をインクリメントします。
4. sbbusy をクリアします。

このレジスタからの読み取りにより、以下が開始されます。
1.データを「返す」。
2. sbbusy を設定します。
3. sbreadondata が設定されている場合、sbaddress に含まれるアドレスからシステムバスの読み取りを実行し、結果を sbdata に配置します。
4. sbautoincrement が設定されている場合、sbaddress を増やします。
5. sbbusy をクリアします。

sbdata0 のみがこの動作を行います。
他の sbdata レジスタには副作用はありません。
32 ビットよりも広いバスを持つシステムでは、デバッガーは他の sbdata レジスタにアクセスした後に sbdata0 にアクセスする必要があります。

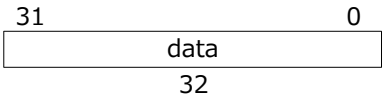


フィールド	説明	アクセス	リセット
data	sbdata のビット 31:0 にアクセスします。	R/W	0

3.12.24 システムバスデータ 63:32（sbdata1、0x3d）

sbaccess64 および sbaccess128 が 0 の場合、このレジスタは存在しません。

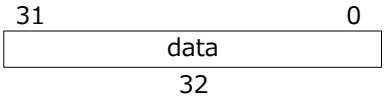
バスマスタがビジーの場合は、sbbusyerror にアクセスし、他には何もしません。



フィールド	説明	アクセス	リセット
data	sbdata のビット 63:32 にアクセスします（システムバスがその幅の場合）。	R/W	0

3.12.25 システムバスデータ 95:64（sbdata2、0x3e）

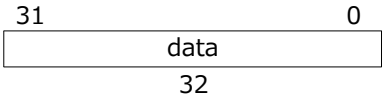
このレジスタは、sbaccess128 が 1 の場合にのみ存在します。
バスマスタがビジーの場合は、sbbusyerror にアクセスし、他には何もしません。



フィールド	説明	アクセス	リセット
data	sbdata のビット 95:64 にアクセスします（システムバスがその幅の場合）。	R/W	0

3.12.26 システムバスデータ 127：96（sbdata3、0x3f）

このレジスタは、sbaccess128 が 1 の場合にのみ存在します。
バスマスタがビジーの場合は、sbbusyerror にアクセスし、他には何もしません。



フィールド	説明	アクセス	リセット
data	sbdata のビット 127:96 にアクセスします（システムバスがその幅の場合）。	R/W	0

第 4 章

RISC-V デバッグ

デバッグをサポートするための RISC-V コアの変更は最小限に抑えられています。
特別な実行モード（デバッグモード）といくつかの追加の CSR があります。
DM は残りを処理します。

この仕様に準拠するには、実装は明示的にオプションとしてリストされていないこのセクションで説明されているすべてを実装する必要があります。

4.1 デバッグモード

デバッグモードは、外部デバッグのためにハートが停止した場合にのみ使用される特別なプロセッサモードです。
デバッグモードの実装方法はここでは指定しません(指定されていません)。

オプションのプログラムバッファーからコードを実行すると、ハートはデバッグモードのままになり、以下が適用されます。

1. mstatus の MPRV が mprven に従って無視される場合を除き、すべての操作はマシンモード特権レベルで実行されます。
2. すべての割り込み（NMI を含む）はマスクされます。
3. 例外はレジスタを更新しません。
   これには、cause、epc、tval、dpc、および mstatus が含まれます。
   プログラムバッファの実行を終了します。
4. トリガーが一致した場合、アクションは実行されません。
5. dcsr の停止カウントに応じて、カウンターが停止する場合があります。
6. dcsr の停止時間に応じて、タイマーが停止する場合があります。
7. wfi 命令は nop として機能します。
8. 特権レベルを変更するほとんどすべての命令には、未定義の動作があります。
   これには、ecall、mret、sret、および uret が含まれます。
   （特権レベルを変更するために、デバッガーは dcsr に prv を書き込むことができます）。
   唯一の例外は ebreak です。
   それがデバッグモードで実行されると、dpc または dcsr を更新せずにハートが再び停止します。
9. プログラムバッファの実行の完了は、fence 命令の目的での出力と見なされます。
10. すべてのコントロール転送命令は、宛先がプログラムバッファにある場合、不正な命令として機能する場合があります。
    そのような命令の 1 つが違法な命令として動作する場合、そのような命令はすべて違法な命令として動作する必要があります。

11. すべてのコントロール転送命令は、宛先がプログラムバッファ外にある場合、不正な命令として機能する場合があります。
    そのような命令の 1 つが違法な命令として動作する場合、そのような命令はすべて違法な命令として動作する必要があります。
12. PC の値に依存する命令（auipc など）は、違法な命令として機能する場合があります。
13. 有効な XLEN は DXLEN です。

一般に、デバッガーは MPRV のすべての効果をシミュレートできることが期待されています。
例外は、34 ビットの物理アドレスにアクセスするために MPRV 機能が必要な Sv32 システムの場合です。
他のシステムは mprven を 0 に結びつける可能性があります。

4.2 ロード予約/ストア条件付き命令

メモリアドレスの lr 命令によって登録された予約は、デバッグモードに入るとき、またはデバッグモード中に失われる場合があります。
これは、lr と sc のペアの間にデバッグモードが開始された場合、前方に進行しない可能性があることを意味します。

これは、デバッグユーザーが認識する必要がある動作です。
lr と sc のペアの間にブレークポイントが設定されている場合、またはそのようなコードをステップ実行している場合、sc は決して成功しません。(sc は成功しない可能性があります。)
幸いなことに、一般的な使用では、このようなシーケンスでの命令は非常に少なく、それをデバッグする人はすぐに予約が発生していないことに気付くでしょう。
その場合の解決策は、sc の後の最初の命令にブレークポイントを設定して実行することです。
高レベル(上位)のデバッガーは、これを自動化することを選択できます。

4.3 割り込み命令を待つ

wfi の実行中に停止が要求された場合、ハートはストール(停止)状態を終了し、この命令の実行を完了してから、デバッグモードに入る必要があります。

4.4 シングルステップ

デバッガーは、停止された hart に単一の命令を実行させ、resumereq を設定する前に step を設定することにより、デバッグモードに再び入ることができます。

その命令を実行またはフェッチして例外が発生した(発生する)場合、PC が例外ハンドラーに変更され、適切な tval および原因レジスタが更新された直後に、デバッグモードに再び入ります。

命令を実行またはフェッチしてトリガーが起動する場合、そのトリガーが起動した直後にデバッグモードに再び入ります。
その場合、原因は 4（単一ステップ）ではなく 2（トリガー）に設定されます。
命令が実行されるかどうかは、トリガーの特定の構成に依存します。

実行された命令により、PC が命令フェッチにより例外が発生するアドレスに変更される場合、その例外は、次にハートが再開されるまで発生しません。

同様に、ハートが実際にその命令を実行しようとするまで、新しいアドレスでのトリガーは起動しません。

ステップオーバーされる命令が wfi であり、通常はハートがストール(停止)する場合、代わりに命令は nop として扱われます。

4.5 リセット

ハートがリセットから抜けたときに、ホールト信号（デバッグモジュールのハートのホールトリクエストビットによって駆動される）または resethaltreq がアサートされる場合、ハートは、命令を実行する前に、通常は最初の命令が実行される前に発生する初期化を実行した後に、デバッグモードに入る必要があります。

4.6 dret 命令

デバッグモードから戻るために、新しい命令 dret が定義されています。
0x7b200073 のエンコーディングがあります。
この命令をサポートするハートでは、デバッグモードで dret を実行すると、pc が dpc に保存されている値に変更されます。
現在の特権レベルは、dcsr の prv で指定されたレベルに変更されます。
ハートはデバッグモードではなくなりました。

デバッグモード以外で dret を実行すると、不正な命令例外が発生します。

デバッグモジュールは必要に応じて実行されることを保証するため、デバッガーは実装が dret をサポートしているかどうかを知る必要はありません。
この仕様で定義されているのは、オペコードを予約し、再利用可能なデバッグモジュールの実装を可能にするためだけです。

4.7 XLEN

デバッグモードでは、XLEN は DXLEN です。
通常のプログラム実行中に XLISA を（misa を調べることによって）決定し、これをユーザーに明確に伝えるのはデバッガー次第です。

4.8 コアデバッグレジスタ

サポートされているコアデバッグレジスタは、デバッグ可能な各ハートに対して実装する必要があります。
これらは CSR であり、RISC-V csr オペコードを使用してアクセスでき、オプションで抽象デバッグコマンドを使用します。

これらのレジスタは、デバッグモードからのみアクセスできます。

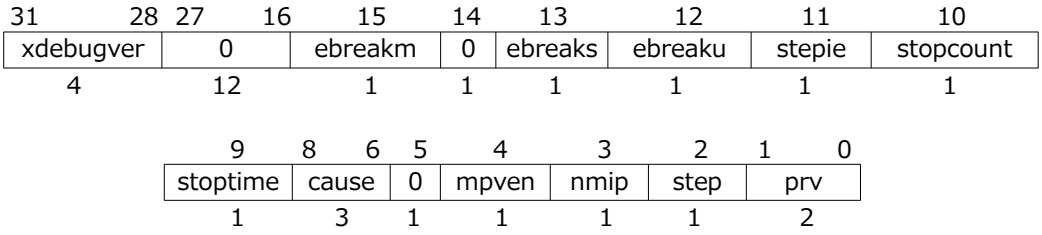
表 4.1：コアデバッグレジスタ

アドレス	名前	ページ
0x7b0	デバッグ制御と状態 (dcsr)	42
0x7b1	デバッグ PC (dpc)	44
0x7b2	デバッグ スクラッチ レジスタ 0 (dscratch0)	45
0x7b3	デバッグ スクラッチ レジスタ 1 (dscratch1)	45

4.8.1 デバッグ制御とステータス（dcsr、0x7b0）

---

cause の優先順位は、最も予測不可能なイベントが最高の優先順位を持つように割り当てられます。



フィールド	説明	アクセス	リセット
xdebugver	0：外部デバッグのサポートはありません。 4：このドキュメントで説明されているように、外部デバッグサポートが存在します。 15：外部デバッグサポートがありますが、この仕様の利用可能なバージョンに準拠していません。	R	事前設定
ebreakm	0：M モードの ebreak 命令は、特権仕様で説明されているように動作します。 1：M モードの ebreak 命令がデバッグモードに入ります。	R/W	0
ebreaks	0：S モードの ebreak 命令は、特権仕様で説明されているように動作します。 1：S モードの ebreak 命令はデバッグモードに入ります。	R/W	0
ebreaku	0：U モードの ebreak 命令は、特権仕様で説明されているように動作します。 1：U モードの ebreak 命令はデバッグモードに入ります。	R/W	0
stepie	0：シングルステップ中に割り込みが無効になります。 1：シングルステッピング中に割り込みが有効になります。 実装では、このビットを 0 に固定する場合があります。 その場合、割り込み動作はデバッガーによってエミュレートできます。 デバッガーは、ハートの実行中にこのビットの値を変更してはなりません。	WARL	0

次のページに続く

フィールド	説明	アクセス	リセット
stopcount	0：カウンタを通常どおりインクリメントします。 1：デバッグモード中、またはデバッグモードへの移行を引き起こす ebreak 命令中は、カウンターをインクリメントしないでください。 これらのカウンターには、cycle および instret CSRs が含まれます。 これは、ほとんどのデバッグシナリオに適しています。 実装では、このビットを 0 または 1 に固定(ハードワイヤー)する場合があります。	WARL	事前設定
stoptime	0：通常どおりタイマーをインクリメントします。 1：デバッグモードでは、ハートローカルタイマーをインクリメントしないでください。 実装では、このビットを 0 または 1 に固定(ハードワイヤー)する場合があります。	WARL	事前設定
cause	デバッグモードに入った理由を説明します。 単一サイクルでデバッグモードに入る理由が複数ある場合、ハードウェアは原因を最も優先度の高い原因に設定する必要があります。 1：ebreak 命令が実行されました。 （優先度 3） 2：トリガーモジュールがブレークポイント例外を引き起こしました。（優先度 4、最高） 3：デバッガーは、haltreq を使用してデバッグモードへのエントリを要求しました。 （優先度 1） 4：ステップが設定されたため、ハートがシングルステップしました。（優先度 0、最低） 5：ハートは、resethaltreq によりリセットから直接停止しました。 これが発生したときに 3 を報告することもできます。 （優先度 2） 他の値は将来の使用のために予約されています。	R	0
mprven	0：mstatus の MPRV は、デバッグモードでは無視されます。 1：mstatus の MPRV は、デバッグモードで有効になります。 このビットの実装はオプションです。 0 または 1 に結び付けられます。(結びつけてもよい)	WARL	事前設定
nmip	設定すると、ハートに対して保留中のマスク不能割り込み（NMI）があります。 NMI はハードウェアエラー状態を示すことがあるため、このビットが設定されると、信頼性の高いデバッグができなくなる可能性があります。 これは実装に依存します。	R	0

次のページに続く

フィールド	説明	アクセス	リセット
step	デバッグモードではなく設定されている場合、ハートは 1 つの命令のみを実行し、デバッグモードに入ります。 例外のために命令が完了しない場合、ハートは、トラップハンドラを実行する前に、適切な例外レジスタを設定してすぐにデバッグモードに入ります。 デバッガーは、ハートの実行中にこのビットの値を変更してはなりません。	R/W	0
prv	デバッグモードに入ったときにハートが動作していた特権レベルが含まれます。 エンコードについては、表 4.5 で説明しています。 デバッガーはこの値を変更して、デバッグモードを終了するときにハートの特権レベルを変更できます。 すべてのハートですべての特権レベルがサポートされているわけではありません。 書き込まれたエンコーディングがサポートされていない場合、またはデバッガーがそれへの変更を許可されていない場合、ハートはサポートされている任意の特権レベルに変更できます。	R/W	3

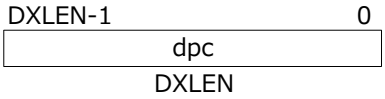
4.8.2 PC のデバッグ（dpc、0x7b1）

デバッグモードに入ると、dpc は次に実行される命令の仮想アドレスで更新されます。
動作については、表 4.3 で詳しく説明します。

表 4.3：デバッグモードエントリ時の DPC の仮想アドレス

原因	DPC の仮想アドレス
ebreak	ebreak 命令のアドレス
single step シングルステップ	デバッグが行われていない場合に次に実行される命令のアドレス。すなわち。 プログラムフローを変更しない 32 ビット命令の場合は pc + 4、取られたジャンプ/ブランチの宛先 PC などです。
trigger module トリガーモジュール	timing が 0 の場合、トリガーを起動させた(原因となった)命令のアドレス。 timing が 1 の場合、デバッグモードに入ったときに実行される次の命令のアドレス。
halt request ハート リクエスト	デバッグモードに入ったときに実行される次の命令のアドレス。

再開すると、ハートの PC は dpc に保存されている仮想アドレスに更新されます。
デバッガーは dpc を作成して、ハートが再開する場所を変更できます。
(デバッガは、ハートが再開する場所を変更するために dpc を書き込む場合があります。)



4.8.3 デバッグスクラッチレジスタ 0（dscratch0、0x7b2）

必要な実装で使用できるオプションのスクラッチレジスタ。
デバッガーは、hartinfo が明示的に言及しない限り、このレジスターに書き込むことはできません (書き込んではいけません)（デバッグモジュールはこのレジスターを内部で使用する場合があります(内部的に使用できます)）。

4.8.4 スクラッチレジスタ 1 のデバッグ（dscratch1、0x7b3）

必要な実装で使用できるオプションのスクラッチレジスタ。
デバッガーは、hartinfo が明示的に言及しない限り、このレジスターに書き込むことはできません (書き込んではいけません)（デバッグモジュールはこのレジスターを内部で使用する場合があります(内部的に使用できます)）。

4.9 仮想デバッグレジスタ

仮想レジスタとは、ハードウェアに直接存在するものではありませんが、デバッガが存在するかのように公開するものです。(デバッガが公開する仮想レジスタのことです。)
デバッグソフトウェアはそれらを実装する必要がありますが、ハードウェアはこのセクションをスキップできます。
仮想レジスタは、デバッガがデバッグレジスタも同じレジスタにアクセスしている間に、デバッグレジスタを慎重に変更することなく、標準デバッガの一部ではない機能にユーザーがアクセスできるようにするために存在します。

表 4.4：仮想コアデバッグレジスタ

アドレス	名前	ページ
仮想	特権レベル（priv）	45

4.9.1 特権レベル（priv、仮想）

ユーザーはこのレジスタを読み取って、ハートが停止したときにハートが実行されていた特権レベルを検査できます。
ユーザーは、このレジスタを記述して、再開時にハートが実行される特権レベルを変更できます。

このレジスタには、dcsr からの prv が含まれていますが、ユーザーがアクセスすると予想される場所にあります。
ユーザーは dcsr に直接アクセスしないでください。デバッガーに干渉する可能性があるためです。

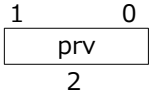


表 4.5：特権レベルのエンコード

エンコーディング	特権レベル
0	ユーザー / アプリケーション
1	監督者
3	マシン

フィールド	説明	アクセス	リセット
prv	デバッグモードに入ったときにハートが動作していた特権レベルが含まれます。 エンコードは表 4.5 で説明されており、特権仕様の特権レベルのエンコードと一致しています。 ユーザーはこの値を記述して、デバッグモードを終了するときにハートの特権レベルを変更できます。	R/W	0

第 5 章

トリガーモジュール

トリガーは、特別な命令を実行することなく、ブレークポイント例外、デバッグモードへの移行、またはトレースアクションを引き起こす可能性があります。
これにより、ROM からコードをデバッグする際に非常に役立ちます。
特定のメモリアドレスでの命令の実行、またはロード/ストアのアドレス/データでトリガーできます。
これらはすべて、デバッグモジュールが存在しなくても役立つ機能であるため、トリガーモジュールは、個別に実装できる個別のピ部品として分割されています。

ハートは、トリガー機能をまったく実装せずにこの仕様に準拠できますが、実装する場合は、このセクションに準拠する必要があります。

デバッグモードではトリガーは起動しません。

各トリガーはさまざまな機能をサポートします。(サポートする場合があります)
デバッガーは、次のようにすべてのトリガーとその機能のリストを作成できます。

1. tselect に 0 を書き込みます。
2. tselect を読み戻し、書き込まれた値が含まれていることを確認します。
   そうでない場合は、ループを終了します。
3. tinfo を読みます。
4.例外が発生した場合、デバッガーは tdata1 を読み取って型を検出する必要があります。
（タイプが 0 の場合、このトリガーは存在しません。ループを終了します。）
5. info が 1 の場合、このトリガーは存在しません。ループを終了します。
6.それ以外の場合、選択したトリガーは info で検出されたタイプをサポートします。
7.繰り返し、tselect の値を増やします。

上記のアルゴリズムは tselect を読み戻すので、2^n トリガーを持つ実装は tselect の n ビットのみを実装する必要があります。
アルゴリズムは、実装に m ビットの tselect があるが 2^m 未満のトリガーがある場合に、tinfo と type をチェックします。

「デバッグモードに入る」アクション（1）を持つトリガーと「ブレークポイント例外を発生させる」アクション（0）を持つ別のトリガーを同時に起動することができます。(起動する可能性があります)
推奨される動作は、両方のアクションを実行することです。
2 つのうちどちらが最初に起こるかは実装に依存します。
これにより、外部デバッガーの存在が実行に影響を与えず、ユーザーコードによって設定されたトリガーが外部デバッガーに影響を与えないことが保証されます。
これが実装されていない場合、ハートはデバッグモードに入り、ブレークポイント例外を無視する必要があります。
後者の場合、アクションが 0 のトリガーのヒットを設定する必要があり、デバッガーにこのケースを処理する機会を与えます。

異なるアクションを持つトリガーが起動されたときにトレースアクションで何が起こるかは、トレース仕様に任されています。

5.1 ネイティブ M モードトリガー

トリガーはネイティブデバッグに使用できます。
フル機能のシステムでは、トリガーは u または s を使用して設定され、起動時にブレークポイント例外がより特権モードにトラップされる可能性があります。
M モードで起動するようにトリガーをネイティブに設定することもできます。
その場合、トラップする上位の特権モードはありません。
そのようなトリガーがトラップハンドラー内に既にあるときにブレークポイント例外を発生させると、システムは通常の実行を再開できなくなります。

フル機能のシステムでは、これはおそらく無視できるリモートコーナーケースです。
ただし、M モードのみを実装するシステムでは、この問題に対する 2 つのソリューションのいずれかを実装することをお勧めします。
このようにトリガーは、M モードコードのネイティブデバッグにも役立ちます。

簡単な解決策は、M モードで mstatus の MIE が 0 の場合に、action= 0 のトリガーがハードウェアで起動しないようにすることです。
その制限は、ユーザーがトリガーを起動したい場合に、割り込みが無効になる場合があることです。(割り込みを無効にする可能性がある場合です。)

より複雑なソリューションは、tcontrol に mte と mpte を実装することです。
このソリューションには、トラップハンドラー中にトリガーのみを無効にするという利点があります。(トラップ ハンドラ中にのみトリガーを無効にする利点があります。)

ブレークポイント例外を引き起こす M モードトリガーを設定するユーザーは、作業中の特定のシステムで発生する可能性のある問題を認識する必要があります。

5.2 トリガーレジスタ

これらのレジスタは CSR であり、RISC-V csr オペコードを使用して、オプションで抽象デバッグコマンドを使用してアクセスできます。

ほとんどのトリガー機能はオプションです。
すべての tdata レジスタは、write-any-read-legal セマンティクスに従います。
デバッガーがサポートされていない構成を書き込むと、レジスターはサポートされている値（単に無効なトリガーである可能性があります）を読み戻します。
つまり、デバッガは、サポートされているものを既に知っている場合を除き、tdata レジスタに書き込む値を常に読み戻す必要があります。
1 つの tdata レジスタへの書き込みは、他の tdata レジスタの内容や、現在選択されているもの以外のトリガーの構成を変更することはできません。

トリガーレジスタは、信頼されていないユーザーコードが OS の許可なしにデバッグモードに入るのを防ぐために、マシンおよびデバッグモードでのみアクセスできます。

このセクションでは、XLEN は M モードの場合は MXLEN、デバッグモードの場合は DXLEN を意味します。
これにより、現在の実行モードと MXLEN の値に基づいて、tdata1 のいくつかのフィールドが移動することに注意してください。

表 5.1：アクションのエンコード

値	説明
0	ブレークポイント例外を発生させます。 （ソフトウェアが外部デバッガーを接続せずにトリガーモジュールを使用する場合に使用します。）
1	デバッグモードに入ります。（トリガーの dmode が 1 の場合のみサポートされます。）
2 - 5	トレース仕様で使用するために予約されています。
それ以外	将来の使用のために予約されています。

表 5.2：トリガーレジスタ

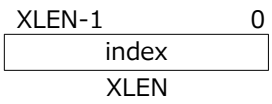
アドレス	名前	ページ
0x7a0	トリガー選択 (tselect)	49
0x7a1	トリガーデータ 1 (tdata1)	50
0x7a1	一致制御 (mcontrol)	53
0x7a1	命令カウント (icount)	58
0x7a1	割り込みトリガー (itrigger)	59
0x7a1	例外トリガー (etrigger)	60
0x7a2	トリガーデータ 2 (tdata2)	50
0x7a3	トリガーデータ 3 (tdata3)	51
0x7a3	トリガー追加 (RV32) (textra32)	60
0x7a3	トリガー追加 (RV64) (textra64)	61
0x7a4	トリガー情報 (tinfo)	51
0x7a5	トリガー制御 (tcontrol)	51
0x7a8	マシンコンテキスト(属性) (mcontext)	52
0x7aa	スーパーバイザ(監督者)コンテキスト(属性) (scontext)	52

5.2.1 トリガー選択（tselect、0x7a0）

このレジスタは、他のトリガーレジスタを介してアクセスできるトリガーを決定します。
アクセス可能なトリガーのセットは 0 から始まり、連続している必要があります。

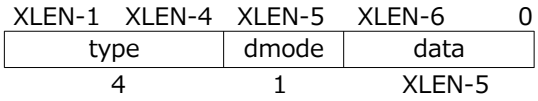
サポートされているトリガーの数以上の値を書き込むと、書き込まれた値とは異なる値がこのレジスターに書き込まれる可能性があります。
作成した内容が有効なインデックスであることを確認するために、デバッガーは値を読み戻し、tselect が作成した内容を保持していることを確認できます。

トリガーはデバッグモードと M モードの両方で使用できるため、デバッガーはこのレジスタを変更した場合、このレジスタを復元する必要があります。



– 2019/09/01

5.2.2 トリガーデータ 1（tdata1、0x7a1）

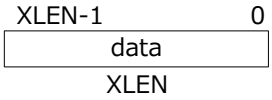


フィールド	説明	アクセス	リセット
type	0：この tselect にはトリガーがありません。 1：トリガーは、従来の SiFive アドレス一致トリガーです。 　これらは実装されるべきではなく、ここではさらに文書化されません。 2：トリガーはアドレス/データ一致トリガーです。 　このレジスタの残りのビットは、mcontrol で説明されているように機能します。 3：トリガーは命令カウントトリガーです。 　このレジスタの残りのビットは、icount で説明されているように機能します。 4：トリガーは割り込みトリガーです。 　このレジスタの残りのビットは、itrigger で説明されているように機能します。 5：トリガーは例外トリガーです。 　このレジスタの残りのビットは、etrigger で説明されているように機能します。 15：このトリガーは存在します（したがって、列挙は終了しません）が、現在使用できません。 他の値は将来の使用のために予約されています。	R/W	事前設定
dmode	0：デバッグモードと M モードの両方で、選択した tselect で tdata レジスタを書き込むことができます。 1：選択された tselect で tdata レジスタに書き込むことができるのは、デバッグモードのみです。 　他のモードからの書き込みは無視されます。 このビットは、デバッグモードからのみ書き込み可能です。	R/W	0
data	トリガー固有のデータ。	R/W	事前設定

5.2.3 トリガーデータ 2（tdata2、0x7a2）

トリガー固有のデータ。

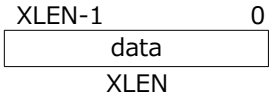
XLEN が DXLEN より小さい場合、このレジスタへの書き込みは符号拡張されます。



5.2.4 トリガーデータ 3（tdata3、0x7a3）

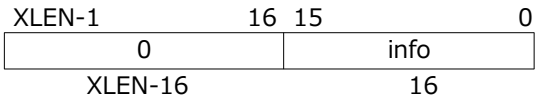
トリガー固有のデータ。

XLEN が DXLEN より小さい場合、このレジスタへの書き込みは符号拡張されます。



5.2.5 トリガー情報（tinfo、0x7a4）

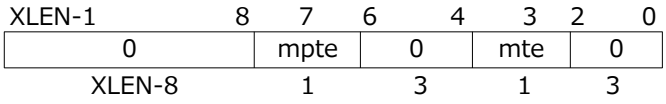
このレジスタ全体は読み取り専用です。



フィールド	説明	アクセス	リセット
info	tdata1 に列挙されている可能な type ごとに 1 ビット。 ビット N はタイプ N に対応します。 ビットが設定されている場合、そのタイプは現在選択されているトリガーによってサポートされます。 現在選択されているトリガーが存在しない場合、このフィールドには 1 が含まれます。 type が書き込み可能でない場合、このレジスタは実装されていない可能性があり、その場合、それを読み取ると不正な命令例外が発生します。 この場合、デバッガは tdata1 からサポートされている唯一のタイプを読み取ることができます。	R	事前設定

5.2.6 トリガー制御（tcontrol、0x7a5）

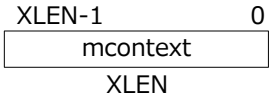
このオプションのレジスタは、M モードトラップハンドラーで action = 0 リングを持つトリガーに関する問題の解決策の 1 つです。詳細については、セクション 5.1 を参照してください。



フィールド	説明	アクセス	リセット
impte	M モード前トリガー有効フィールド。 M モードへのトラップが取得されると、mpte は mte の値に設定されます。	R/W	0
mte	M モードトリガーイネーブルフィールド。 0：ハートが M モードの場合(間)、action = 0 のトリガーは一致/発動しません。 1：ハートが M モードのときにトリガーが一致/発動します。 M モードへのトラップが取得されると、mte は 0 に設定されます。 mret が実行されると、mte は mpte の値に設定されます。	R/W	0

5.2.7 マシンコンテキスト（mcontext、0x7a8）

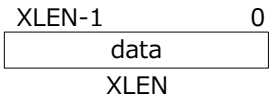
このレジスタは、M モードおよびデバッグモードでのみ書き込み可能です。



フィールド	説明	アクセス	リセット
mcontext	マシンモードソフトウェアは、このレジスタにコンテキスト番号を書き込むことができます。このレジスタを使用して、その特定のコンテキストでのみ起動するトリガーを設定できます。 実装では、このフィールドの任意の数の上位ビットを 0 に関連付けることができます。 RV32 では 6 ビット以下、RV64 では 13 ビット以下を実装することをお勧めします。	R/W	0

5.2.8 スーパーバイザーコンテキスト（scontext、0x7aa）

このレジスタは、S モード、M モード、およびデバッグモードでのみ書き込み可能です。



フィールド	説明	アクセス	リセット
data	スーパーバイザモードソフトウェアは、このレジスタにコンテキスト番号を書き込むことができます。このレジスタを使用して、その特定のコンテキストでのみ起動するトリガーを設定できます。 実装では、このフィールドの任意の数の上位ビットを 0 に関連付けることができます。 RV32 では 16 ビット以下、RV64 では 34 ビット以下を実装することをお勧めします。	R/W	0

5.2.9 一致制御（mcontrol、0x7a1）

type が 2 の場合、このレジスタは tdata1 としてアクセスできます。

アドレスおよびデータトリガーの実装は、プロセッサコアの実装方法に大きく依存しています。
さまざまな実装に対応するため、アドレス/データトリガーの実行、ロード、および保存は、実装にとって最も都合の良い時点で起動する場合があります。
デバッガーは、timing で説明されている特定のタイミングを要求する場合があります。
表 5.8 は、最高のユーザーエクスペリエンスのためのタイミングを示しています。

表 5.8：推奨されるブレークポイントタイミング

一致	推奨されるトリガータイミング
実行アドレス	前
命令を実行する	前
アドレス+命令の実行	前
ロードアドレス	前
データを読み込む	後
アドレス+データの読み込み	後
ストアアドレス	前
データを保存する	前
ストアアドレス+データ	前

このトリガータイプは、アドレス比較（選択は常に 0）のみに制限される場合があります。
その場合、tdata2 はすべての有効な仮想アドレスを保持できる必要がありますが、他の値を保持できる必要はありません。

XLEN-1	XLEN-4	XLEN-5	XLEN-6	XLEN-11	XLNE-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

フィールド	説明	アクセス	リセット
maskmax	match が 1 の場合にハードウェアがサポートする最大の自然に整列した 2 のべき乗（NAPOT）範囲を指定します。 値は、その範囲のバイト数の 2 を底とする対数です。 値 0 は、完全に一致する値のみがサポートされることを示します（1 バイト範囲）。 値 63 は、NAPOT の最大範囲に対応し、サイズは 2^{63} バイトです。	R	事前設定
sizehi	このフィールドは、XLEN が 32 より大きい場合にのみ存在します。 その場合、サイズが拡張されます。 存在しない場合、ハードウェアはフィールドに 0 が含まれているかのように動作します。	R/W	0
hit	このオプションのビットが実装されている場合、ハードウェアはこのトリガーが一致したときにそれを設定します。 トリガーのユーザーはいつでもトリガーを設定またはクリアできます。 どのトリガーが一致したかを判別するために使用されます。 ビットが実装されていない場合、ビットは常に 0 であり、ビットを書き込んでも効果はありません。	R/W	0
select	0：仮想アドレスで一致を実行します。 1：ロードまたは保存されたデータ値、または実行された命令で一致を実行します。	R/W	0

次のページに続く

type と dmode の説明が無いけど、5.2.2 トリガーデータ 1（tdata1、0x7a1）の type,dmode と同じでいいのかな。

フィールド	説明	アクセス	リセット
timing	0：このトリガーのアクションは、トリガーした命令が実行される直前に実行されますが、先行するすべての命令がコミットされた後です。 1：このトリガーのアクションは、トリガーした命令が実行された後に実行されます。 次の命令が実行される前に取得する必要がありますが、トリガーを実装し、その提案を実装しないほうが、それらをまったく実装しないよりも優れています。 ほとんどのハードウェアは、選択、実行、ロード、およびストアに依存する可能性がある、いずれかのタイミングのみを実装します。 このビットは主に、ハードウェアが何が起こるかをデバッガと通信するために存在します。 ハードウェアは、完全に書き込み可能なビットを実装できます。その場合、デバッガーはもう少し制御があります。 タイミングが 0 のデータロードトリガーは、デバッガーがハートの実行を許可すると、同じロードが再び発生します。 データロードトリガーの場合、デバッガーは最初に 1 のタイミングでブレークポイントを設定しようとする必要があります。 すべてが同じタイミング値を持たないトリガーのチェーンは、連続した命令が適切なトリガーに一致しない限り起動しません。 タイミングが 0 のトリガーが一致する場合、タイミングが 1 のトリガーも一致するかどうかは実装に依存します。	R/W	0

次のページに続く

フィールド	説明	アクセス	リセット
sizelo	このフィールドには、size の下位 2 ビットが含まれます。 上位ビットは sizehi から来ています。 結合された値は次のように解釈されます。 0：トリガーは、任意のサイズのアクセスに対して一致を試みます。 select = 0 の場合、またはアクセスサイズが XLEN の場合にのみ、動作が明確に定義されます。 1：トリガーは 8 ビットのメモリアクセスに対してのみ一致します。 2：トリガーは、16 ビットメモリアクセスまたは 16 ビット命令の実行に対してのみ一致します。 3：トリガーは、32 ビットメモリアクセスまたは 32 ビット命令の実行に対してのみ一致します。 4：トリガーは、48 ビット命令の実行に対してのみ一致します。 5：トリガーは、64 ビットメモリアクセスまたは 64 ビット命令の実行に対してのみ一致します。 6：トリガーは 80 ビット命令の実行に対してのみ一致します。 7：トリガーは、96 ビット命令の実行に対してのみ一致します。 8：トリガーは 112 ビット命令の実行に対してのみ一致します。 9：トリガーは、128 ビットメモリアクセスまたは 128 ビット命令の実行に対してのみ一致します。	R/W	0
axtion	トリガーが起動したときに実行するアクション。 値は表 5.1 で説明されています。	R/W	0

次のページに続く

フィールド	説明	アクセス	リセット
chain	0：このトリガーが一致すると、構成されたアクションが実行されます。 1：このトリガーは一致しませんが、次のインデックスを持つトリガーは一致しません。(一致するのを防ぎます) トリガーチェーンは、chain = 0 のトリガーの後、chain = 1 の最初のトリガー、または chain = 1 の場合は最初のトリガーから開始します。 chain = 0(を待つ)の最初のトリガーで終了します。 この最後のトリガーはチェーンの一部です。 最終トリガーを除くすべてのアクションは無視されます。 その最終トリガーに対するアクションは、チェーン内のすべてのトリガーが同時に一致する場合にのみ実行されます。 チェーンは次のトリガーに影響するため、ハードウェアは、次のトリガーの dmode が 1 の場合、dmode を 0 に設定する mcontrol への書き込みでゼロにする必要があります。 さらに、前のトリガーの dmode が 0 でチェーンが 1 の場合、ハードウェアは dmode を 1 に設定する mcontrol への書き込みを無視する必要があります。 デバッガーは、mcontrol を記述している場合、前のトリガーのチェーンをチェックすることにより、後者のケースを回避する必要があります。 トリガーチェーンの最大長を制限したい実装（タイミング要件を満たすなど）は、mcontrol への書き込みでチェーンをゼロにすることで、チェーンが長くなりすぎる可能性があります。	R/W	0
match	0：値が tdata2 と等しい場合に一致します。 1：値の上位 M ビットが tdata2 の上位 M ビットと一致する場合に一致します。 　M は XLEN-1 から tdata2 に 0 を含む最下位ビットのインデックスを引いたものです。 2：値が tdata2 より大きい（符号なし）または等しい場合に一致します。 3：値が（符号なし）tdata2 より小さい場合に一致します。 4：値の下半分が tdata2 の上半分と AND された後、値の下半分が tdata2 の下半分に等しい場合に一致します。 5：値の上半分が tdata2 の上半分と AND された後、値の上半分が tdata2 の下半分と等しい場合に一致します。 他の値は将来の使用のために予約されています。	R/W	0
m	設定すると、このトリガーを M モードで有効にします。	R/W	0
s	設定すると、このトリガーを S モードで有効にします。	R/W	0
u	設定すると、このトリガーを U モードで有効にします。	R/W	0

次のページに続く

フィールド	説明	アクセス	リセット
execute	設定すると、トリガーは、実行される命令の仮想アドレスまたはオペコードで起動します。	R/W	0
store	設定すると、トリガーはストアの仮想アドレスまたはデータで起動します。	R/W	0
load	設定すると、トリガーは仮想アドレスまたはロードのデータで起動します。	R/W	0

5.2.10 命令カウント（icount、0x7a1）

type が 3 の場合、このレジスタは tdata1 としてアクセスできます。

このトリガータイプは、外部デバッガーとソフトウェアモニタープログラムの両方に役立つ単一のステップとして使用することを目的としています。
その場合、1 より大きいカウントをサポートする必要はありません。
これらのシナリオで有用なモードビットの 2 つの組み合わせは、u 自体、または m、s、u がすべて設定されている場合のみです。

ハードウェアがカウントを 1 に制限し、カウントを減らす代わりにモードビットを変更する場合、このレジスタは 2 ビットだけで実装できます。
1 つは u 用、もう 1 つは m と s 用です。
外部デバッガーのみ、またはソフトウェアモニターのみをサポートする必要がある場合は、1 ビットで十分です。

XLEN-1	XLEN-4	XLEN-5	XLEN-6 25	24	23 10	9	8	7	6	5 0
type		dmode	0	hit	count	m	0	s	u	action
4		1	XLEN-30	1	14	1	1	1	1	6

フィールド	説明	アクセス	リセット
hit	このオプションのビットが実装されている場合、ハードウェアはこのトリガーが一致したときにそれを設定します。 トリガーのユーザーはいつでもトリガーを設定またはクリアできます。 どのトリガーが一致したかを判別するために使用されます。 ビットが実装されていない場合、ビットは常に 0 であり、ビットを書き込んでも効果はありません。	R/W	0
count	カウントが 0 まで減少されると、トリガーが起動します。 カウントを 1 から 0 に変更する代わりに、ハードウェアが m、s、および u をクリアすることも許容されます。 これにより、このレジスタが単一ステップでのみ存在する場合、カウントを 1 に固定(ハードワイヤード)できます。	R/W	1
m	設定すると、すべての命令が完了するか、M モードで例外が発生すると 1 ずつカウントされます。	R/W	0
s	設定すると、すべての命令が完了するか、S モードで例外が発生すると 1 ずつカウントされます。	R/W	0
u	設定すると、すべての命令が完了するか、U モードで例外が発生すると 1 ずつカウントされます。	R/W	0

次のページに続く

フィールド	説明	アクセス	リセット
action	トリガーが起動したときに実行するアクション。 値は表 5.1 で説明されています。	R/W	0

5.2.11 割り込みトリガー（itrigger、0x7a1）

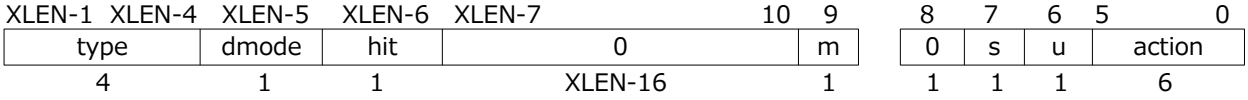
タイプが 4 の場合、このレジスタは tdata1 としてアクセスできます。

このトリガーは、mie で構成可能な割り込みのいずれかで起動する場合があります（特権仕様で説明されています）。
発火する割り込みは、割り込みを有効にするために mie で設定されるのと同じビットを tdata2 に設定することで構成されます。

ハードウェアは、このトリガーの割り込みのサブセットのみをサポートする場合があります。
デバッガは、要求された機能が実際にサポートされていることを確認するために、書き込み後に tdata2 を読み戻す必要があります。

トリガーは、割り込みのためにハートがトラップを取得した場合にのみ起動します。
（たとえば、タイマー割り込みが発生しても起動しませんが、その割り込みは mie で有効になっていません。）

トリガーが起動すると、すべての CSR が特権仕様の定義に従って更新され、要求されたアクションは、割り込み/例外ハンドラーの最初の命令が実行される直前に実行されます。



フィールド	説明	アクセス	リセット
hit	このオプションのビットが実装されている場合、ハードウェアはこのトリガーが一致したときにそれを設定します。 トリガーのユーザーはいつでもトリガーを設定またはクリアできます。 どのトリガーが一致したかを判別するために使用されます。 ビットが実装されていない場合、ビットは常に 0 であり、ビットを書き込んでも効果はありません。	R/W	0
m	設定すると、M モードから取得される割り込みに対してこのトリガーを有効にします。	R/W	0
s	設定すると、S モードから取得される割り込みに対してこのトリガーを有効にします。	R/W	0
u	設定すると、U モードから取得される割り込みに対してこのトリガーを有効にします。	R/W	0
action	トリガーが起動したときに実行するアクション。 値は表 5.1 で説明されています。	R/W	0

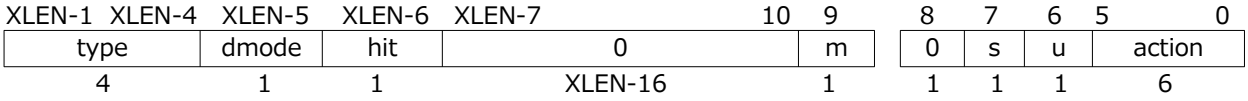
5.2.12 例外トリガー（トリガー、0x7a1）

タイプが 5 の場合、このレジスタは tdata1 としてアクセスできます。

このトリガーは、mcause で定義された例外コードの XLEN まで起動できます（Interrupt = 0 の特権仕様で説明されています）。
これらの原因は、tdata2 に対応するビットを書き込むことで構成されます。
（たとえば、不正な命令をトラップするために、デバッガーは tdata2 のビット 2 を設定します。）

ハードウェアは例外のサブセットのみをサポートする場合があります。
デバッガは、要求された機能が実際にサポートされていることを確認するために、書き込み後に tdata2 を読み戻す(返す)必要があります。

トリガーが起動すると、すべての CSR が特権仕様の定義に従って更新され、要求されたアクションは、割り込み/例外ハンドラーの最初の命令が実行される直前に実行されます。

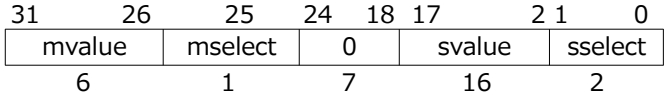


フィールド	説明	アクセス	リセット
hit	このオプションのビットが実装されている場合、ハードウェアはこのトリガーが一致したときにそれを設定します。 トリガーのユーザーはいつでもトリガーを設定またはクリアできます。 どのトリガーが一致したかを判別するために使用されます。 ビットが実装されていない場合、ビットは常に 0 であり、ビットを書き込んでも効果はありません。	R/W	0
m	設定すると、M モードから取得される割り込みに対してこのトリガーを有効にします。	R/W	0
s	設定すると、S モードから取得される割り込みに対してこのトリガーを有効にします。	R/W	0
u	設定すると、U モードから取得される割り込みに対してこのトリガーを有効にします。	R/W	0
action	トリガーが起動したときに実行するアクション。 値は表 5.1 で説明されています。	R/W	0

5.2.13 トリガーエクストラ（RV32）（textra32、0x7a3）

タイプが 2、3、4、または 5 の場合、このレジスタは tdata3 としてアクセスできます。

このレジスタのすべての機能はオプションです。
値ビットは、任意の数の上位ビットを 0 に関連付けることができます。
選択ビットは 0（無視）のみをサポートします。

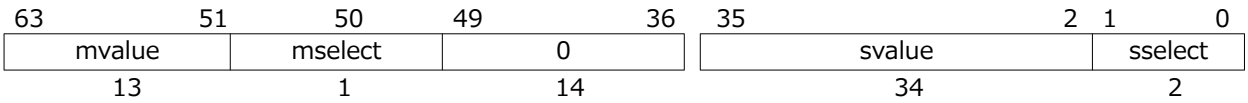


フィールド	説明	アクセス	リセット
mvalue	mselect とともに使用されるデータ。	R/W	0
mselect	0：mvalue を無視します。 1：このトリガーは、mcontext の下位ビットが mvalue と等しい場合にのみ一致します。	WARL	0
svalue	select と共に使用されるデータ。	R/W	0
sselect	0：svalue を無視します。 1：このトリガーは、scontext の下位ビットが svalue に等しい場合にのみ一致します。 2：このトリガーは、satp の ASID が svalue の下位 ASIDMAX（特権仕様で定義）ビット値と等しい場合にのみ一致します。	WARL	0

WARL って、Write any, read legal. だったよね

5.2.14 トリガーエクストラ（RV64）（textra64、0x7a3）

XLEN が 64 の場合、これは textra のレイアウトです。
フィールドは、上記の textra32 で定義されています。



第 6 章

デバッグトランスポートモジュール（DTM）

デバッグトランスポートモジュールは、1 つ以上のトランスポート（JTAG や USB など）を介して DM へのアクセスを提供します。

1 つのプラットフォームに複数の DTM が存在する場合があります。
理想的には、外部と通信するすべてのコンポーネントに DTM が含まれており、サポートするすべてのトランスポートを介してプラットフォームをデバッグできます。
たとえば、USB コンポーネントには DTM を含めることができます。
これにより、任意のプラットフォームを USB 経由で簡単にデバッグできます。
必要なのは、すでに使用されている USB モジュールがデバッグモジュールインターフェイスにもアクセスできることだけです。

複数の DTM を同時に使用することはサポートされていません。
これが起こらないようにすることはユーザーに任されています。

この仕様では、セクション 6.1 で JTAG DTM を定義しています。
この仕様の将来のバージョンでは、追加の DTM が追加される可能性があります。

実装は、このセクションを実装しなくても、この仕様に準拠できます。
その場合、「RISC-V デバッグ仕様 0.13.2、カスタム DTM に準拠」として広告(アドバタイズ)する必要があります。
ここで説明する JTAG DTM を実装する場合は、「RISC-V デバッグ仕様 0.13.2、JTAG DTM に準拠」として広告(アドバタイズ)する必要があります。

6.1 JTAG デバッグトランスポートモジュール

このデバッグトランスポートモジュールは、通常の JTAG テストアクセスポート（TAP）に基づいています。
JTAG TAP では、最初に JTAG 命令レジスタ（IR）を使用して 1 つを選択し、次に JTAG データレジスタ（DR）を介してアクセスすることにより、任意の JTAG レジスタにアクセスできます。

6.1.1 JTAG の背景

JTAG は IEEE Std 1149.1-2013 を指します。
集積回路に含まれるテストロジックを定義する標準であり、集積回路間の相互接続をテストし、集積回路自体をテストし、コンポーネントの通常動作中に回路アクティビティを観察または変更します。

この仕様は後者の機能を使用します。
JTAG 標準は、コンポーネントのデバッグハードウェアとの通信に使用できるいくつかのカスタムレジスタの読み取りと書き込みに使用できるテストアクセスポート（TAP）を定義しています。

6.1.2 JTAG DTM レジスタ

DTM として使用される JTAG TAP には、少なくとも 5 ビットの IR が必要です。
TAP がリセットされると、IR はデフォルトで 00001 になり、IDCODE 命令が選択されます。
JTAG レジスタの完全なリストとそのエンコーディングを表 6.1 に示します。
IR が実際に 5 ビットを超える場合、表 6.1 のエンコーディングは、最上位ビットに 0 を使用して拡張する必要があります。
デバッガが使用する可能性のある通常の JTAG レジスタは BYPASS と IDCODE だけですが、この仕様は他の多くの標準 JTAG 命令用の IR スペースを残しています。
未実装の命令は BYPASS レジスタを選択する必要があります。

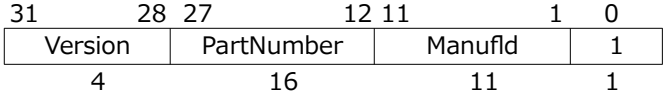
表 6.1：JTAG DTM TAP レジスタ

アドレス	名前	説明	頁
0x00	BYPASS　バイパス	JTAG はこのエンコードを推奨しています	
0x01	IDCODE　ID コード	JTAG はこのエンコードを推奨しています	
0x10	DTM 制御と状態(dtcs)	デバッグ用	
0x11	デバッグモジュールインターフェースアクセス(dmi)	デバッグ用	
0x12	予約 (BYPASS バイパス)	将来の RISC-V デバッグ用に予約済み	
0x13	予約 (BYPASS バイパス)	将来の RISC-V デバッグ用に予約済み	
0x14	予約 (BYPASS バイパス)	将来の RISC-V デバッグ用に予約済み	
0x15	予約 (BYPASS バイパス)	将来の RISC-V 標準用に予約済み	
0x16	予約 (BYPASS バイパス)	将来の RISC-V 標準用に予約済み	
0x17	予約 (BYPASS バイパス)	将来の RISC-V 標準用に予約済み	
0x1f	BYPASS バイパス	JTAG にはこのエンコードが必要です	

6.1.3 IDCODE（0x01 で）

TAP ステートマシンがリセットされると、このレジスタが（IR で）選択されます。
その定義は、IEEE Std 1149.1-2013 で定義されているとおりです。

このレジスタ全体は読み取り専用です。



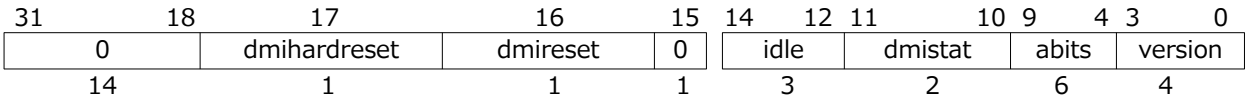
フィールド	説明	アクセス	リセット
Version	この部分のリリースバージョンを識別します。	R	事前設定

次のページに続く

フィールド	説明	アクセス	リセット
PartNumber	この部品の設計者の部品番号を識別します。	R	事前設定
Manufld	この部品の設計者/製造者を識別します。 ビット 6：0 は、JEDEC 標準 JEP106 によって割り当てられた設計者/製造業者の識別コードのビット 6：0 でなければなりません。 ビット 10：7 には、同じ識別コードの継続文字数（0x7f）のモジュロ 16 カウントが含まれます。	R	事前設定

6.1.4 DTM の制御とステータス（dtmcs、0x10）

このレジスタのサイズは、デバッガが常に DTM のバージョンを判断できるように、将来のバージョンでは一定のままです。



フィールド	説明	アクセス	リセット
dmihardreset	このビットに 1 を書き込むと、DTM のハードリセットが行われ、DTM は未処理の DMI トランザクションを忘れます。 一般に、これは、未処理の DMI トランザクションが決して完了しないと予想する理由がある場合にのみ使用してください。 （例：リセット条件により、実行中の DMI トランザクションがキャンセルされました）。	W1	-
dmireset	このビットに 1 を書き込むと、スティッキーエラー状態がクリアされ、DTM が前のトランザクションを再試行または完了することができます。	W1	-
idle	これは、デバッガが「ビジー」リターンコード（dmistat 3）を回避するために、DMI スキャンのたびに Run-Test / Idle で費やす必要のある最小サイクル数のヒントです。 デバッガーは必要に応じて dmistat をチェックする必要があります。 0：Run-Test / Idle を入力する必要はまったくありません。 1：Run-Test / Idle を入力し、すぐに終了します。 2：Run-Test / Idle を入力し、1 サイクルの間そこにとどまってから出発します。 等々。	R	事前設定

次のページに続く

フィールド	説明	アクセス	リセット
dmistat	0：エラーなし。 1：予約済み。 2 と同じように解釈します。 2：操作が失敗しました（op が 2 になりました(op2 の結果)）。 3：DMI アクセスがまだ進行中に操作が試行されました（結果として op が 3(op3 の結果)）。	R	0
abits	dmi のアドレスのサイズ。	R	事前設定
version	0：仕様バージョン 0.11 で説明されているバージョン。 1：仕様バージョン 0.13 で説明されているバージョン。 15：この仕様の利用可能なバージョンでは説明されていないバージョン。	R	1

6.1.5 デバッグモジュールインターフェイスアクセス（dmi、0x11）

このレジスタにより、デバッグモジュールインターフェイス（DMI）にアクセスできます。

Update-DR では、op で報告された現在のステータスがスティッキーでない限り、DTM は op で指定された操作を開始します。

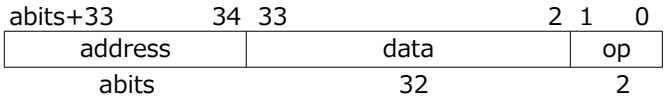
Capture-DR では、DTM はその操作の結果でデータを更新し、現在の操作がスティッキーでない場合は op を更新します。

この使用方法の例については、セクション B.1 および表??を参照してください。

Table ?? 表 ?? ってどれだろ？

まだ進行中の状態(ステータス)は、多くのスキャンをまとめてバッチ処理するデバッガーに対応するためにスティッキーです。これらはすべて、問題が発生したらすぐに実行または停止する必要があります。

たとえば、一連のスキャンでデバッグプログラムを作成して実行できます。
書き込みの 1 つが失敗しても実行が継続すると、デバッグプログラムがハングするか、その他の予期しない副作用が発生する可能性があります。



フィールド	説明	アクセス	リセット
address	DMI アクセスに使用されるアドレス。 Update-DR では、この値は DMI を介して DM にアクセスするために使用されます。	R/W	0
data	Update-DR 中に DMI を介して(通じて)DM に送信するデータ、および前の操作の結果として DM から返されたデータ。	R/W	0

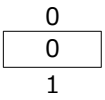
次のページに続く

-- 2019/09/23

フィールド	説明	アクセス	リセット
op	デバッガがこのフィールドに書き込むとき、次の意味があります。 0：データとアドレスを無視します。 （nop） Update-DR 中に DMI 経由で何も送信しないでください。 この操作により、ビジーまたはエラー応答が発生することはありません。 次の Capture-DR で報告されるアドレスとデータは未定義です。 1：アドレスから読み取り。 （read） 2：アドレスにデータを書き込みます。 （write） 3：予約済み。 デバッガがこのフィールドを読み取るとき、次のことを意味します。 0：前の操作は正常に完了しました。 1：予約済み。 2：前の操作が失敗しました。 このアクセスで dmi にスキャンされたデータは無視されます。 このステータスはスティッキーであり、dtmcs に dmireset を書き込むことでクリアできます。 これは、DM 自体がエラーで応答したことを示します。 DM がエラーで応答する特定のケースはなく、DMI はエラーを返すことをサポートする必要はありません。 3：DMI 要求がまだ進行中に操作が試行されました。 このアクセスで dmi にスキャンされたデータは無視されます。 このステータスはスティッキーであり、dtmcs に dmireset を書き込むことでクリアできます。 デバッガーがこのステータスを確認した場合、Update-DR と Capture-DR の間のターゲットにより多くの TCK エッジを与える(増やす)必要があります。 これを行う最も簡単な方法は、Run-Test / Idle に追加の遷移を追加することです。	R/W	0

6.1.6 バイパス（0x1f）

効果のない 1 ビットのレジスタ。 デバッガーがこの TAP と通信したくない場合に使用されます。
このレジスタ全体は読み取り専用です。



6.1.7 推奨 JTAG コネクタ

デバッグハードウェアを簡単に入手できるようにするため、この仕様では、デバッグおよびトレース コネクタに関する MIPI アライアンス勧告バージョン 1.10.00、2011 年 3 月 16 日で説明されているように、MIPI-10 .05 インチコネクタ仕様と互換性のあるコネクタを推奨しています。

コネクタの間隔は.05 インチ、金メッキのオスヘッダーで、厚さは.016 インチの硬化銅またはベリリウムブロンズの正方形の柱（SAMTEC FTSH または同等品）です。
メスコネクタは、互換性のある 20µm ゴールドコネクタです。

オスヘッダーを上から見ると（ピンが目を指している）、ターゲットのコネクタは表 6.5 のように見えます。
各ピンの機能を表 6.7 で説明します。

表 6.5：MIPI-10 コネクタ図

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

プラットフォームで nTRST が必要な場合、nRESET ピンを nTRST 信号として再利用することができます。
プラットフォームでシステムリセットと TAP リセットの両方が必要な場合は、MIPI-20 コネクタを使用する必要があります。
その物理コネクタは、MIPI-10 とほぼ同じですが、2 倍の長さであり、2 倍のピンをサポートしています。
そのコネクタを表 6.6 に示します。

表 6.6：MIPI-20 コネクタ図

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

同じコネクタを 2 線式 cJTAG に使用できます。
その場合、TMC は TMSC に使用され、TCK は TCKC に使用されます。

表 6.7：JTAG コネクタのピン配列

1	VREF DEBUG	ロジックハイの基準電圧。
2	TMS	デバッグアダプターによって駆動される JTAG TMS 信号。
4	TCK	デバッグアダプターによって駆動される JTAG TCK 信号。
6	TDO	ターゲットによって駆動される JTAG TDO 信号。
7	GND or KEY	ヘッダーが常に正しく差し込まれていることを確認するために、このピンをオスで切り、メスのヘッダーに差し込むことができます。 ただし、TCK の速度を最速にするために、このピンを追加のグランドとして使用することをお勧めします。 ケーブルが誤って差し込まれないように、シュラウドコネクタを使用する必要があります。
8	TDI	デバッグアダプターによって駆動される JTAG TDI 信号。
10	nRESET	デバッグアダプタによって駆動されるアクティブローのリセット信号。 リセットをアサートすると、RISC-V コアと PCB 上のその他の周辺機器がリセットされます。 デバッグロジックをリセットしないでください。 このピンはオプションですが、強くお勧めします。 必要に応じて、このピンを代わりに nTRST として使用できます。 nRESET を TAP リセットに接続しないでください。そうしないと、デバッガーがリセットを介してデバッグしてクラッシュの原因を発見したり、リセット後に実行制御を維持したりできない場合があります。
12	RTCK	ターゲットによって駆動されるテストクロックを返します。 ターゲットは、TCK 信号を処理した後、ここで TCK 信号を中継し、デバッガが応答して TCK 周波数を調整できるようにします。
14	nTRST_PD	デバッグアダプターによって駆動されるテストリセットプルダウン（オプション）。 nTRST と同じ機能ですが、ターゲットにプルダウン抵抗があります。
16	nTRST	テストリセット（オプション）、デバッグアダプターによって駆動されます。 JTAG TAP コントローラーをリセットするために使用されます。
18	TRIGIN	使用されず、デバッグアダプターによって Low に駆動されます。
20	TRIGOUT	使用されず、ターゲットによって駆動されます。

ここに出てくる KEY って、オスコネクタのピン折って、メスコネクタの穴をつぶしておいて、その向きでしか入らないようにするためのものだよね。(ただのピンコネクタは 180 度回転させてもそのまま入る。)
昔のフロッピーディスクコネクタとか、IDE のコネクタにあったようなやつ。(うろおぼえ)

付録 A

ハードウェア実装

以下に 2 つの可能な実装を示します。
デザイナーは、1 つを選択し、組み合わせて、または独自のデザインを思いつく(考え出す)ことができます。

A.1 抽象コマンドベース

停止は、ハート実行パイプラインをストール(停止)させることで発生します。

レジスタファイルの Muxs により、アクセスレジスタ抽象コマンドを使用して GPR および CSR にアクセスできます。

メモリへのアクセスには、抽象アクセスメモリコマンドを使用するか、システムバスアクセスを使用します。

この実装により、デバッガーは、ハートが命令を実行できない場合でも、ハートから情報を収集できます。

A.2 実行ベース

この実装は、停止したハートの GPR に対してアクセスレジスタ抽象コマンドのみを実装し、他のすべての操作についてはプログラムバッファーに依存します。
ハートの既存のパイプラインと、任意のメモリ位置から実行する機能を使用して、ハートのデータパスへの変更を回避します。

停止要求ビットが設定されると、デバッグモジュールは選択されたハートに対して特別な割り込みを発生させます。
この割り込みにより、各ハートはデバッグモードに入り、DM によって処理される定義済みのメモリ領域にジャンプします。
この例外を取得すると、pc は dpc に保存され、原因は dcsr で更新されます。

デバッグモジュールのコードにより、ハートは「パークループ」を実行します。
パークループでは、ハートはその mhartid をデバッグモジュール内のメモリ位置に書き込み、停止したことを示します。
DM が停止した複数のハートのうちの 1 つを個別に制御できるように、各ハートは DM が制御するメモリ位置のフラグをポーリングして、デバッガがプログラムバッファを実行するか再開するかを決定します。

抽象コマンドを実行するために、DM は最初にコマンドに従ってプログラムバッファーの内部ワードを入力します。
転送が設定されると、DM はこれらのワードに lw <gpr>、0x400（zero）または sw 0x400（zero）、<gpr>を取り込みます(設定します)。
64 ビットと 128 ビットのアクセスは、それぞれ ld / sd と lq / sq を使用します。
転送が設定されていない場合、DM はこれらの指示を nops として設定します。
実行が設定されている場合、実行はデバッガー制御のプログラムバッファーまで続行します。そうでない場合、DM は ebreak を直ちに実行します。

ebreak が実行されると（プログラムバッファーコードの終了を示す）、ハートはそのパークループに戻ります。
例外が発生した場合、ハートはデバッグモジュール内のデバッグ例外アドレスにジャンプします。
そのアドレスのコードにより、ハートは例外を示すデバッグモジュール(内)のアドレスに書き込みます。
このアドレスは、フェンス指示の I/O と見なされます（39 ページの＃9 を参照）。
その後、ハートはパークループに(飛び)戻ります。
DM は書き込みから例外があったと推測し、cmderr を適切に設定します。

実行を再開するために、デバッグモジュールは hart に dret を実行させるフラグを設定します。
dret が実行されると、pc は dpc から復元され、prv によって設定された特権で通常の実行が再開されます。

data0 などは、12 ビットの imm のみを使用して、ゼロに関連するアドレスで通常のメモリにマップされます。
正確なアドレスは、デバッガーが依存してはならない実装の詳細です。
たとえば、データレジスタは 0x400 にマップされる場合があります。

さらに柔軟性を高めるため、プログラムの実行またはデータ転送に使用できる連続したメモリ領域を形成するために、progbuf0 などが data0 の直前の通常のメモリにマップされます。

付録 B

デバッガーの実装

このセクションでは、セクション 6.1 で説明した JTAG DTM を使用して、外部デバッガーが説明したデバッグインターフェイスを使用して RISC-V コアでいくつかの一般的な操作を実行する方法について詳しく説明します。
これらの例はすべて 32 ビットコアを想定していますが、サンプルを 64 ビットまたは 128 ビットコアに容易に適合させる必要があります。

例を読みやすくするために、すべてが成功すると想定し、デバッガーが次のアクセスを実行できるよりも速く完了すると想定しています。
これは、一般的な JTAG セットアップの場合です。
ただし、デバッガは一連のアクションを実行した後、常にスティッキーエラーステータスビットをチェックする必要があります。
設定されているものが見つかった場合は、遅延を追加したり、ステータスビットを明示的にチェックしたりしながら、同じアクションを再試行する必要があります。

B.1 デバッグモジュールインターフェイスアクセス

任意のデバッグモジュールレジスタを読み取るには、dmi を選択し、op を 1 に設定して値をスキャンし、アドレスを目的のレジスタアドレスに設定します。
Update-DR では操作が開始され、Capture-DR では結果がデータにキャプチャされます。
操作が時間内に完了しなかった場合、op は 3 になり、データの値は無視する必要があります。
dtmcs に dmireset を書き込むことにより、ビジー状態をクリアしてから、2 回目のスキャンスキャンを再度実行する必要があります。
op が 0 を返すまで、このプロセスを繰り返す必要があります。
後の操作で、デバッガは Capture-DR と Update-DR の間の時間を長くする(より多くの時間を確保する)必要があります。

任意のデバッグバスレジスタを書き込むには、dmi を選択し、op を 2 に設定して値をスキャンし、アドレスとデータを目的のレジスタアドレスとデータにそれぞれ設定します。
それ以降は、読み取りの代わりに書き込みが実行されることを除いて、すべてが読み取りの場合とまったく同じように行われます。

通常の JTAG 使用の非効率性の大部分を回避するために、IR をスキャンする必要はほとんどありません。

B.2 停止したハートの確認

ユーザーは、ハートが停止したとき（たとえば、ブレークポイントが原因）、できるだけ早く知りたいと思うでしょう。
多数のハートがあるときにどのハートが停止したかを効率的に判断するために、デバッガは haltsum レジスタを使用します。
最大数のハートが存在すると仮定すると、最初に haltsum3 をチェックします。
そこで設定された各ビットに対して、hartsel を書き込み、haltsum2 をチェックします。
このプロセスは、haltsum1 と haltsum0 まで繰り返されます。
存在するハートの数に応じて、プロセスは、より低い haltsum レジスタの 1 つ(いずれか)で開始する必要があります。

B.3 停止

1 つまたは複数のハートを停止するには、デバッガーがそれらを選択し、haltreq を設定してから、ハートが停止したことを示すために allhalted を待機します。
次に、haltreq を 0 にクリアするか、ハイのままにして、停止中にリセットされるハートをキャッチします。

B.4 ランニング

まず、デバッガーは上書きされたレジスターを復元する必要があります。
その後、resumereq を設定して、選択したハートを実行させることができます。
allresumeack が設定されると、デバッガーはハートが再開したことを認識し、resumereq をクリアできます。
ハーツは再開後に非常に急速に停止する場合があり（ソフトウェアブレークポイントにヒットするなど）、デバッガーは allhalted / anyhalted を使用してハートが再開したかどうかを確認できません。

B.5 シングルステップ

ハードウェアシングルステップ機能の使用は、通常の実行とほぼ同じです。
デバッガは、ハートを実行させる前に dcsr にステップを設定するだけです。
ハートは実行中の場合とまったく同じように動作しますが、割り込みは無効になり（stepie によって異なります）、デバッグモードに入る前に 1 つの命令のみをフェッチして実行します。

B.6 レジスタへのアクセス

B.6.1 抽象コマンドの使用

抽象コマンドを使用して s0 を読み取ります。 :

Op	アドレス	値	コマンド
ライト	command	aarsize = 2, transfer, regno = 0x1008	s0 を読む
リード	data0	-	s0 にあった値を返します

抽象コマンドを使用して mstatus を記述します。 :

Op	アドレス	値	コマンド
ライト	data0	新値	
ライト	command	aarsize = 2, transfer, write, regno = 0x300	mstatus に書き込む

B.6.2 プログラムバッファーの使用

抽象コマンドは、GPR とデータを交換するために使用されます。
このメカニズムを使用すると、値を GPR に出入りさせることで、他のレジスタにアクセスできます。

プログラムバッファーを使用して mstatus を書き込みます。 :

Op	アドレス	値	コマンド
ライト	progbuf0	csrw s0, MSTATUS	
ライト	progbuf1	ebreak	
ライト	data0	新値	
ライト	command	Aarsize = 2, postexec, transfer, write, regno = 0x1008	s0 を書き込んでから、プログラムバッファを実行します

プログラムバッファを使用して f1 を読み取ります。 :

Op	アドレス	値	コマンド
ライト	progbuf0	fmv.x.s s0, f1	
ライト	progbuf1	ebreak	
ライト	command	postexec	プログラムバッファを実行
ライト	command	Transfer, regno = 0x1008	s0 を読む
リード	data0	-	f1 にあった値を返します

B.7 メモリの読み取り

B.7.1 システムバスアクセスの使用

システムバスアクセスでは、アドレスは物理システムバスアドレスです。

システムバスアクセスを使用してメモリから単語(ワード)を読み取ります。 :

Op	アドレス	値	コマンド
ライト	sbcs	sbaccess = 2, sbreadonaddr	設定
ライト	sbaddress0	address	
リード	sbdata0	-	メモリから読み取った値

システムバスアクセスを使用してメモリブロックを読み取ります。 :

Op	アドレス	値	コマンド
ライト	sbcs	sbaccess = 2, sbreadonaddr, sbreadondata, sbautoincrement	自動読み取りと自動インクリメントをオンにする
ライト	sbaddress0	address	書き込みアドレストリガーの読み取りとインクリメント
リード	sbdata0	-	メモリから読み取った値
リード	sbdata0	-	メモリから読み取られる次の値
…	…	…	…
ライト	sbcs	0	自動読み取りを無効にする
リード	sbdata0	-	メモリから最後に読み取った値を取得します。

B.7.2 プログラムバッファーの使用

プログラムバッファを介して、ハートはメモリアクセスを実行します。
アドレスは物理的または仮想的です（mprven および他のシステム構成に依存）。

プログラムバッファを使用してメモリから単語を読み取ります。：

Op	アドレス	値	コマンド
ライト	progbuf0	lw s0, 0(s0)	
ライト	progbuf1	ebreak	
ライト	data0	address	
ライト	command	write, postexec, regno = 0x1008	s0 を書き込んでから、プログラムバッファを実行します
ライト	command	regno = 0x1008	s0 を読む
リード	data0		メモリから読み取った値

プログラムバッファを使用してメモリブロックを読み取ります。：

Op	アドレス	値	コマンド
ライト	progbuf0	lw s1, 0(s0)	
ライト	progbuf1	addi s0, s0, 4	
ライト	progbuf2	ebreak	
ライト	data0	address	
ライト	command	write, postexec, regno = 0x1008	s0 を書き込んでから、プログラムバッファを実行します
ライト	command	postexec, regno = 0x1009	s1 を読み取り、プログラムバッファを実行
ライト	abstractauto	autoexecdata[0]	autoexecdata[0]設定
リード	data0	-	メモリから値を読み取り、プログラムバッファを実行します
リード	data0	-	メモリから読み取られた次の値を取得し、プログラムバッファを実行します
...	...	...	...
ライト	abstractauto	0	autoexecdata[0]クリア
リード	data0	-	メモリから最後に読み取った値を取得します。

B.7.3 抽象メモリアクセスの使用

抽象メモリアクセスは、実際の実装は異なる場合がありますが、ハートによって実行されるかのように機能します。

抽象メモリアクセスを使用してメモリから単語を読み取ります。 :

Op	アドレス	値	コマンド
ライト	data1	address	
ライト	command	cmdbype = 2, aamsize = 2	
リード	data0	-	メモリから読み取った値

抽象メモリアクセスを使用してメモリブロックを読み取ります。 :

Op	アドレス	値	コマンド
ライト	abstractauto	1	data0 にアクセスしたときにコマンドを再実行します
ライト	data1	address	
ライト	command	cmdtype = 2, aamsize = 2, aampostincrement = 1	
リード	data0	-	値を読み取り、次のアドレスの読み取りをトリガーします
...	...	...	...
ライト	abstractauto	0	自動実行を無効にする
リード	data0	-	メモリから最後に読み取った値を取得します。

B.8 メモリーの書き込み

B.8.1 システムバスアクセスの使用

システムバスアクセスでは、アドレスは物理システムバスアドレスです。

システムバスアクセスを使用してメモリにワードを書き込みます。：

Op	アドレス	値	コマンド
ライト	sbaddress0	address	
ライト	sbdata0	value	

システムバスアクセスを使用してメモリブロックを書き込みます。：

Op	アドレス	値	コマンド
ライト	sbcs	sbaccess = 2, sbautoincrement	自動インクリメントをオンにする
ライト	sbaddress0	address	
ライト	sbdata0	Value0	
ライト	sbdata0	value1	
...	...	...	...
ライト	sbdata0	valueN	

B.8.2 プログラムバッファーの使用

プログラムバッファを介して、ハートはメモリアクセスを実行します。
アドレスは物理的または仮想的です（mprven および他のシステム構成に依存）。

プログラムバッファを使用してメモリにワードを書き込みます。：

Op	アドレス	値	コマンド
ライト	progbuf0	sw s1, 0(s0)	
ライト	progbuf1	ebreak	
ライト	data0	address	
ライト	command	write, regno = 0x1008	s0 に書き込み
ライト	data0	value	
ライト	command	write, postexec, regno = 0x1009	s1 を書き込んでから、プログラムバッファを実行します

プログラムバッファを使用してメモリブロックを書き込みます。：

Op	アドレス	値	コマンド
ライト	progbuf0	SW s1, 0(s0)	
ライト	progbuf1	addi s0, s0, 4	
ライト	progbuf2	ebreak	
ライト	data0	address	
ライト	command	write, regno = 0x1008	s0 を書き込む
ライト	data0	value0	
ライト	command	Write, postexec, regno = 0x1009	s1 を書き込んでから、プログラムバッファを実行します
ライト	abstractauto	autoexecdata[0]	autoexecdata [0]を設定します
ライト	data0	value1	
...	...	...	...
ライト	data0	valueN	
ライト	abstructauto	0	autoexecdata [0]をクリアします

B.8.3 抽象メモリアクセスの使用

抽象メモリアクセスは、実際の実装は異なる場合がありますが、ハートによって実行されるかのように機能します。

抽象メモリアクセスを使用して単語をメモリに書き込みます。：

Op	アドレス	値	コマンド
ライト	data1	address	
ライト	data0	value	
ライト	command	cmdtype = 2, aamsize = 2, write = 1	

抽象メモリアクセスを使用してメモリブロックを書き込みます。：

Op	アドレス	値	コマンド
ライト	data1	address	
ライト	data0	value0	
ライト	command	cmdtype = 2, aamsize = 2, write = 1, aampostincrement = 1	
ライト	abstractauto	1	data0 にアクセスしたときにコマンドを再実行します
ライト	data0	value1	
ライト	data0	value2	
...	...	...	
ライト	data0	valueN	
ライト	abstractauto	0	自動実行を無効にする

B.9 トリガー

デバッガーは、特定のイベントが発生したときにハードウェアトリガーを使用してハートを停止できます。
以下にいくつかの例を示しますが、ハートによって実装されるトリガーの機能の数に要件はないため、これらの例はすべての実装に適用できるわけではありません。
デバッガーがトリガーを設定する場合、目的の構成を書き込み、その構成がサポートされているかどうかを確認します。

0x80001234 の命令が実行される直前にデバッグモードに入り、ROM の命令ブレークポイントとして使用します。：

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

0x80007f80 の値が読み取られた直後にデバッグモードに入ります。：

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

0x80007c80 から 0x80007cef（両端を含む）のアドレスへの書き込みの直前にデバッグモードに入ります。

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 (execlusive)

0x81230000 から 0x8123ffff までのアドレスへの書き込みの直前にデバッグモードに入ります。：

tdata1	0x10da	taction = 1, match = 1, m = 1, s = 1, u = 1, store = 1
tdata2	0x81237fff	16bit to match exactly, then 0, then all ones. 完全に一致する 16 ビット、次に 0、次にすべて 1。

0x86753090 から 0x8675309f まで、または 0x96753090 から 0x9675309f までのアドレスからの読み取り直後にデバッグモードに入ります。：

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 下位半分をマスクしてから、下位半分を一致させる
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 上位半分をマスクし、上位半分を一致させる

B.10 例外の処理

一般に、デバッガーは、作成するプログラムに注意することで例外を回避できます。
ただし、ユーザーがメモリまたは実装されていない CSR にアクセスするように要求した場合などやむを得ない場合もあります。
典型的(一般的)なデバッガーは、何が起こるかを知るのにプラットフォームについて十分に知らないので、結果を決定するためにアクセスを試みなければなりません。

プログラムバッファの実行中に例外が発生すると、cmderr が設定されます。
デバッガはこのフィールドをチェックして、プログラムで例外が発生したかどうかを確認できます。
例外があった場合、何が原因であるに違いないかを知るのはデバッガーに任されています。

B.11 クイックアクセス

GPR とデータレジスタ間でデータを転送するためのさまざまな命令(手順)があります。
それらは、ロード/ストアまたは CSR 読み取り/書き込みのいずれかです。
特定のアドレスも異なります。
これはすべて、hartinfo で指定されています。
ここの例では、これらすべてのオプションを表すために、疑似 op dest、src を使用しています。

単一のメモリ書き込みを実行するために、最小時間ハートを停止します。：

Op	アドレス	値	コマンド
ライト	progbuf0	transfer arg2, s0	s0 を保存
ライト	progbuf1	transfer s0, arg0	最初の引数（アドレス）を読み取る
ライト	progbuf2	transfer arg0, s1	s1 を保存
ライト	progbuf3	transfer s1, arg1	2 番目の引数（データ）を読み取る
ライト	progbuf4	sw s1, 0(s0)	
ライト	progbuf5	transfer s1, arg0	s1 を復元
ライト	progbuf6	transfer s0, arg2	s0 を復元
ライト	progbuf7	ebreak	
ライト	data0	address	
ライト	data1	data	
ライト	command	0x10000000	クイックアクセスを実行する

これは、M モードでハードウェアブレークポイントを有効にするために mcontrol で m ビットを設定する例を示しています。
ここで有効にするトリガーを設定するために、以前に同様のクイックアクセス手順を使用できました。：
(同様のクイック アクセス命令は、ここで有効になっているトリガーを構成するために以前に使用された可能性があります。：)

Op	アドレス	値	コマンド
ライト	progbuf0	transfer arg0, s0	s0 を保存
ライト	progbuf1	li s0, (1 << 6)	m ビットのマスクを形成します
ライト	progbuf2	csrrs x0, tdata1, s0	マスクを mcontrol に適用する
ライト	progbuf3	Transfer s0, arg2	s0 を復元
ライト	progbuf4	ebreak	
ライト	command	0x10000000	クイックアクセスを実行する

付録 C

バグの修正

C.1 0.13.1

0.13 の批准以来、0.13.1 では次のバグが修正されました。

C.1.1 再開後、再開 ack ビットが設定される

セクション 3.5 の 3 番目の段落には誤りがあります。
そこで説明されているプロセスの最後に、再開 ack ビットが設定されます。

C.1.2 aamsize は引数の幅に影響しません

セクション 3.7.1.3 で定義されている Access Memory 抽象コマンドの引数の幅は、amsize ではなく DXLEN によって決定されます。

C.1.3 sbdata0 が操作の順序を読み取る

セクション 3.12.23 にリストされている、sbdata0 からの読み取りを説明する操作の順序は正しくありません。
次のようになります。

このレジスタからの読み取りにより、以下が開始されます。

1.データを「返す」。
2. sbbusy を設定します。
3. sbreadondata が設定されている場合、別のシステムバス読み取りを実行します。
4. sbautoincrement が設定されている場合、sbaddress を増やします。
5. sbbusy をクリアします。

C.1.4 haltreq が設定されている場合のハートリセット動作

ハートがリセットから抜け、haltreq が設定されると、ハートはすぐにデバッグモードに入ります。

C.1.5 mte は、action = 0 の場合にのみ適用されます

セクション 5.2.6 の mte の定義では、mte はアクションが 0 であるトリガーにのみ影響を与えると述べています。(影響することを明記する必要があります。)

C.1.6 sselect は svalue に適用されます

セクション 5.2.13 では、sselect が 0 の場合、svalue は無視されます。

C.1.7 最後のトリガーの例

セクション B.9 の最後の例では、tdata2 1 の値は 0xefff8675 である必要があります。

C.2 0.13.2

クイックアクセスの説明の手順 1 がドキュメントにない原因となる書式設定の問題を修正しました。

索引

--2019/09/23
--翻訳完了