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。

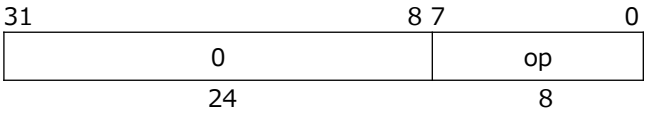
1.2.2 レジスタ定義形式

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。

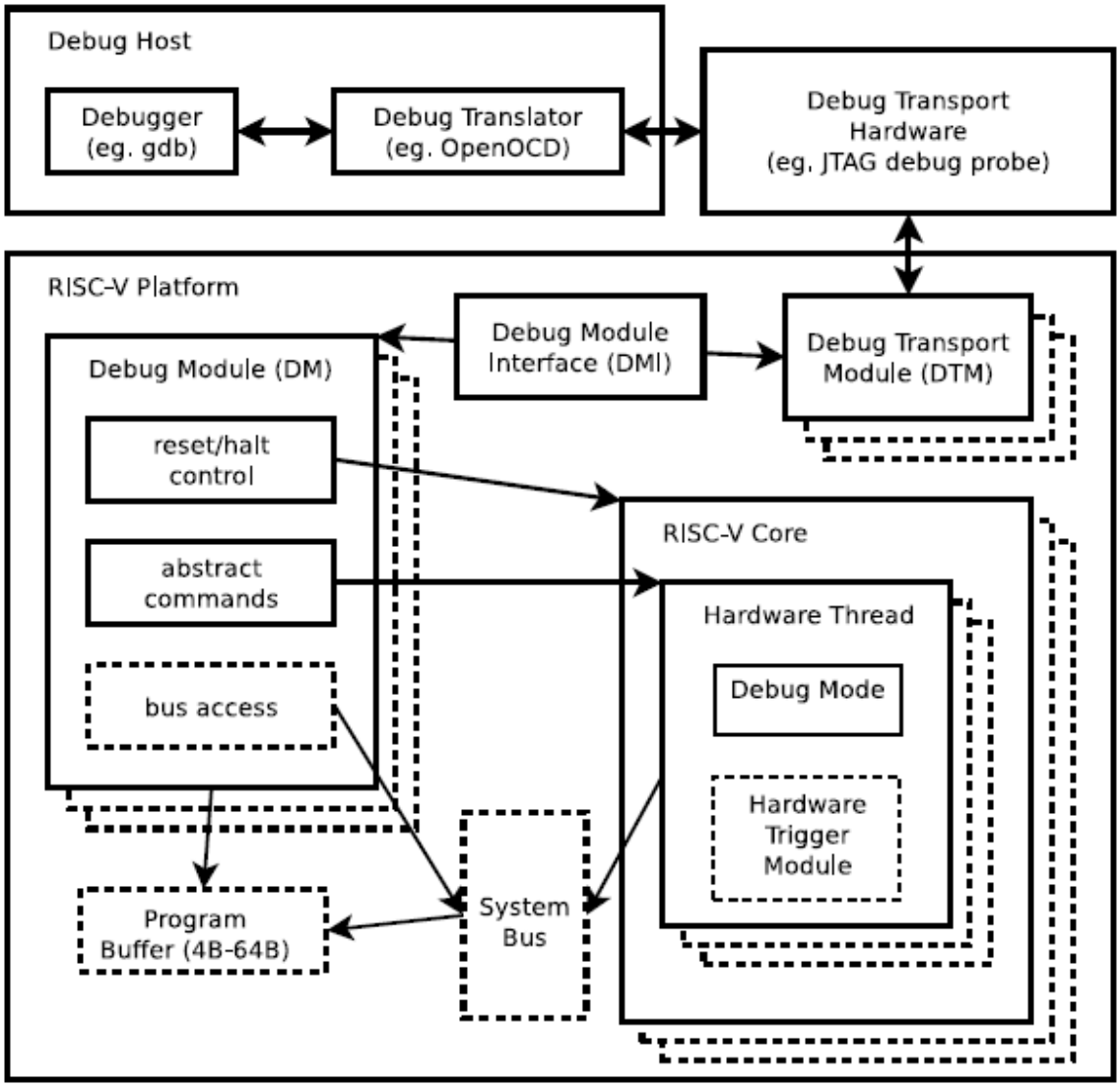


図2.1：RISC-Vデバッグシステムの概要

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。

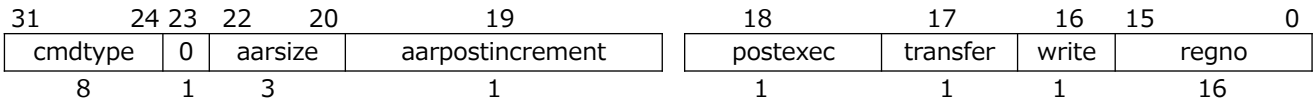
---

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

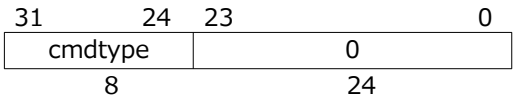
3.6.1.2 クイックアクセス

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。

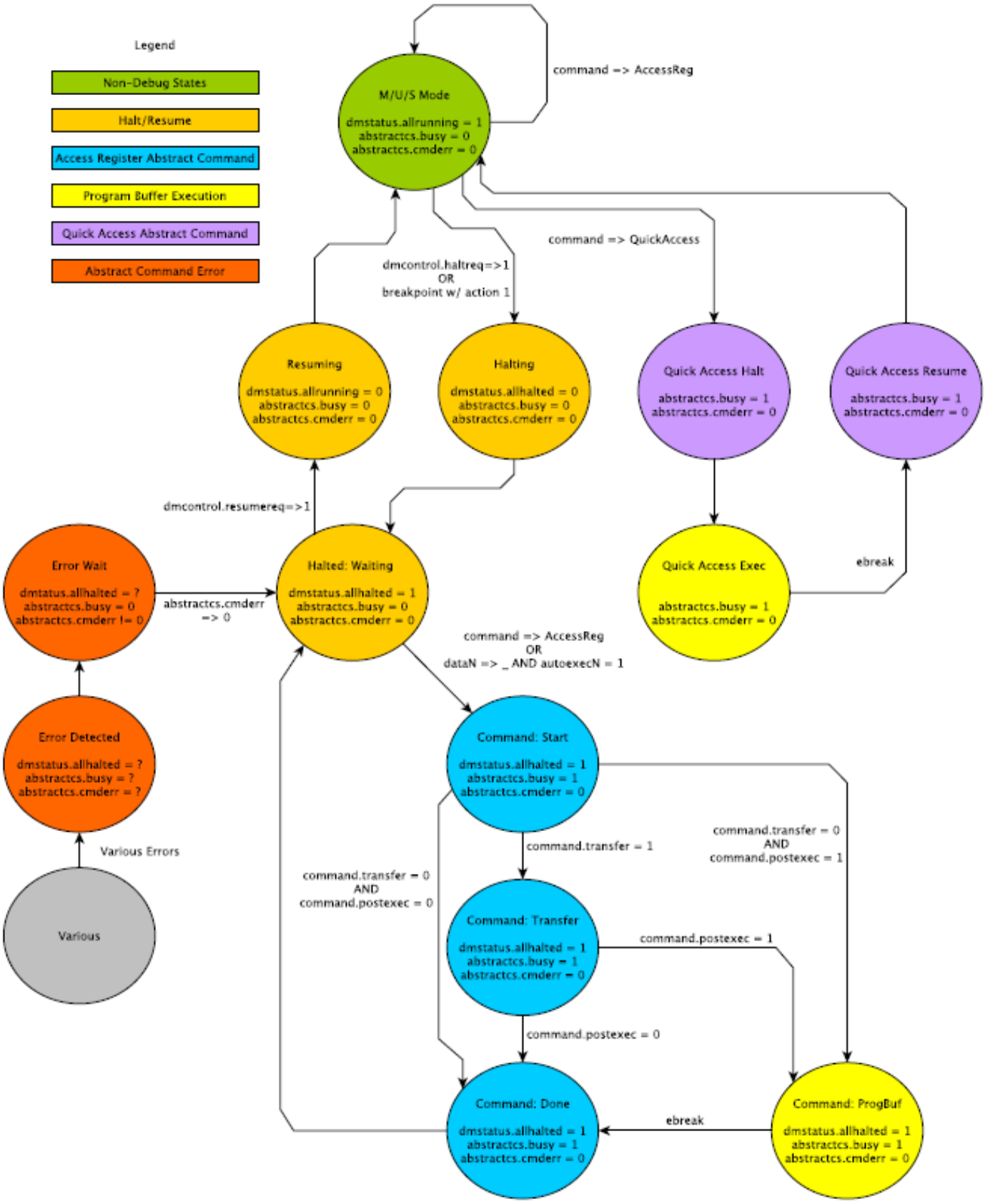


図 3.1 シングルハートトシステム用の実行/停止デバッグステートマシン
ごくわずかな状態しかデバッガに表示されないため、状態と遷移は概念的です。

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。

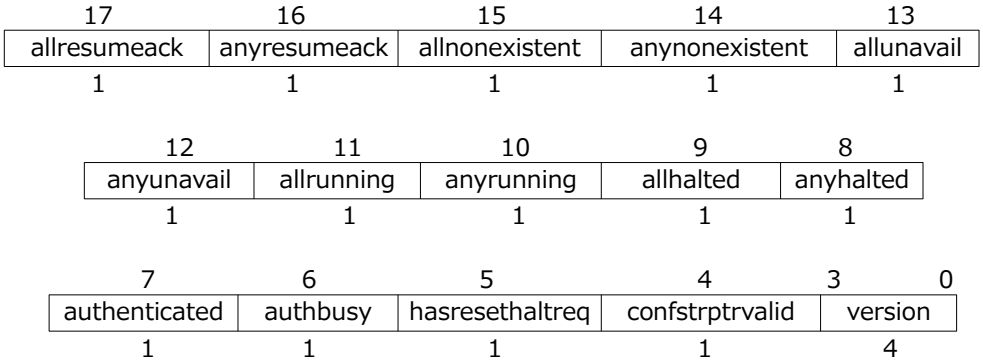
表 3.8 : デバッグモジュールデバッグバスレジスタ

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

これ以降翻訳未、次回更新