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。

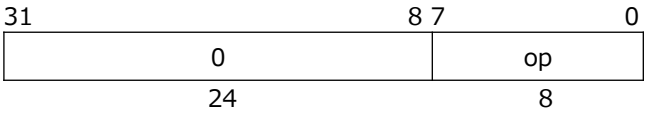
1.2.2 レジスタ定義形式

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。

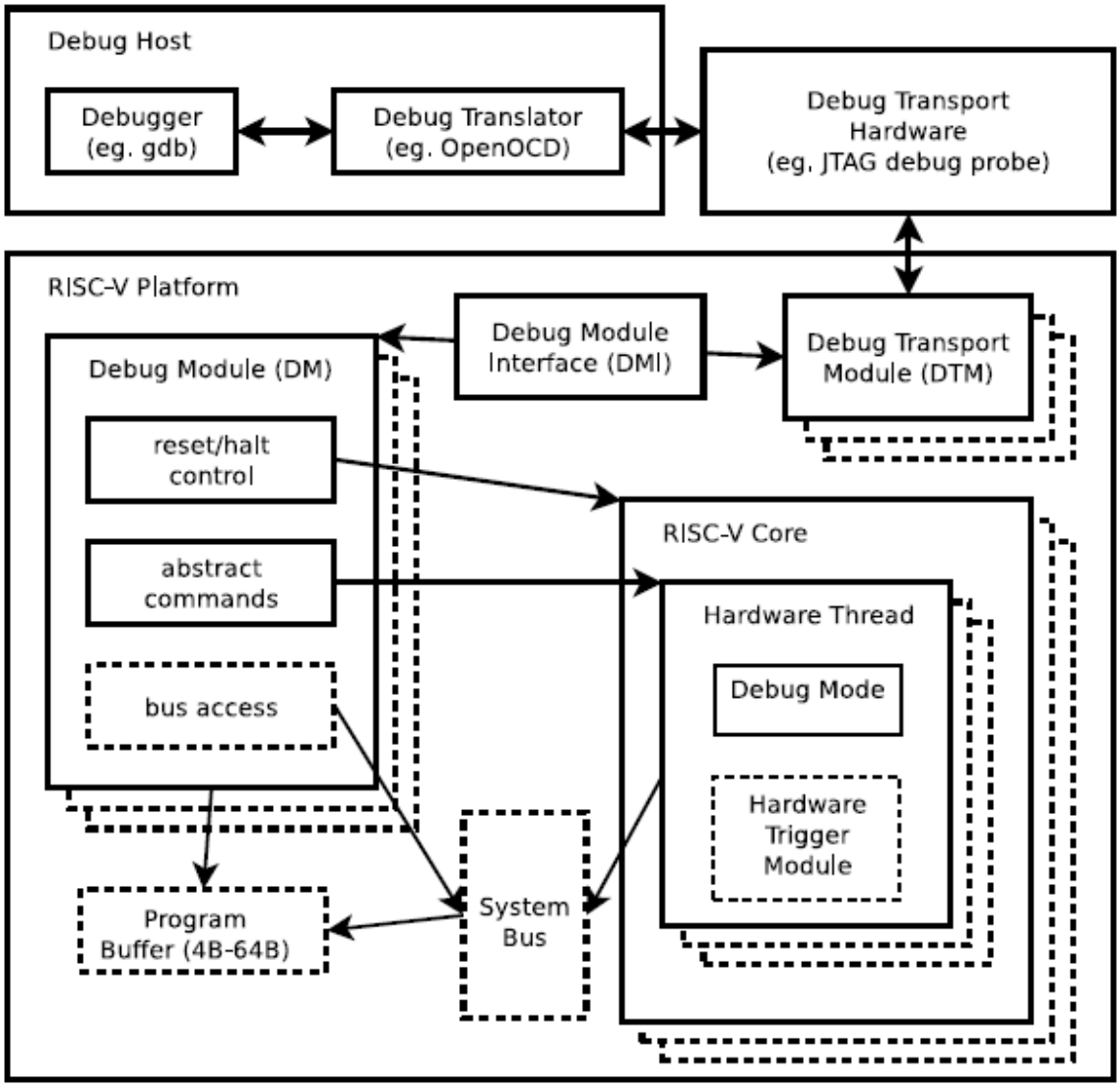


図 2.1：RISC-V デバッグシステムの概要

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

これ以降翻訳未、次回更新