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](mailto: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

1はじめに 1

1.1用語。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 1

1.1.1文脈。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 1

1.1.2バージョン。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 2

1.2この文書について。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 2

1.2.1構造。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 2

1.2.2レジスタ定義フォーマット。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 2

1.2.2.1ロングネーム（ショートネーム、0x123） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 2

1.3背景。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 3

1.4サポートされている機能。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 3

2システム概要 5

3デバッグモジュール（DM） 7

3.1デバッグモジュールインタフェース（DMI）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 8

3.2リセット制御。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 8

3.3ハートの選択。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 9

3.3.1シングルハートの選択。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 9

3.3.2複数のハートの選択。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 9

3.4ハート状態。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 9

3.5実行制御。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 10

3.6抽象コマンド。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 11

3.6.1抽象コマンド一覧。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 12

3.6.1.1アクセスレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 12

3.6.1.2クイックアクセス。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 13

3.6.1.3アクセスメモリ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 14

３.7プログラムバッファ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 15

3.8常体の概要。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 16

3.9システムバスアクセス。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 16

3.10最小限の侵入型デバッグ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 18

3.11セキュリティ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 18

3.12デバッグモジュールレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 19

3.12.1デバッグモジュールステータス（dmstatus、0x11） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 20

3.12.2デバッグモジュール制御（dmcontrol、0x10） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 22

3.12.3 ハート情報（hartinfo、0x12） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 25

3.12.4 ハート アレイ ウィンドウ 選択（hawindowsel、0x14）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 26

3.12.5 ハート アレイ ウィンドウ（hawindow、0x15） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 27

3.12.6抽象制御と常体（abstractcs、0x16） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 27

3.12.7抽象コマンド（command、0x17） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 28

3.12.8抽象コマンドAutoexec（abstractauto、0x18） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 29

3.12.9設定文字列ポインタ0（confstrptr0、0x19）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 29

3.12.10次のデバッグモジュール（nextdm、0x1d） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 30

3.12.11要約データ0（data0、0x04）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 30

3.12.12プログラムバッファ0（progbuf0、0x20） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 30

3.12.13認証データ（authdata、0x30） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 31

3.12.14 停止概要 0（haltsum0、0x40）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 31

3.12.15 停止概要 1（haltsum1、0x13）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 31

3.12.16 停止概要 2（haltsum2、0x34）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 32

3.12.17 停止概要 3（haltsum3、0x35）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 32

3.12.18システムバスのアクセス制御と状態（sbcs、0x38） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 32

3.12.19システムバスアドレス 31: 0（sbaddress0、0x39）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 34

3.12.20システムバスアドレス 63:32（sbaddress1、0x3a） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 35

3.12.21システムバスアドレス 95:64（sbaddress2、0x3b）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 35

3.12.22システムバスアドレス 127:96（sbaddress3、0x37）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 36

3.12.23システムバスデータ31：0（sbdata0、0x3c） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 36

3.12.24システムバスデータ63:32（sbdata1、0x3d） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 37

3.12.25システムバスデータ95:64（sbdata2、0x3e） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 37

3.12.26システムバスデータ127：96（sbdata3、0x3f） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 38

4 RISC-Vデバッグ 39

4.1デバッグモード。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 39

4.2ロード予約/ストアコンディショナル命令。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 40

4.3割り込み命令を待つ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 40

4.4シングルステップ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 40

4.5リセット。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 41

4.6 dretインストラクション。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 41

4.7 XLEN 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 41

4.8コアデバッグレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 41

4.8.1デバッグ制御とステータス（dcsr、0x7b0） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 42

4.8.2 PCのデバッグ（dpc、0x7b1）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 44

4.8.3デバッグスクラッチレジスタ0（dscratch0、0x7b2） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 45

4.8.4デバッグスクラッチレジスタ1（dscratch1、0x7b3）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 45

4.9仮想デバッグレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 45

4.9.1特権レベル（priv、仮想時） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 45

5トリガーモジュール 47

5.1ネイティブMモードトリガ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 48

5.2トリガレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 48

5.2.1トリガ選択（tselect、0x7a0）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 49

5.2.2トリガデータ1（tdata1、0x7a1）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 50

5.2.3トリガデータ2（tdata2、0x7a2）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 50

5.2.4トリガデータ3（tdata3、0x7a3）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 51

5.2.5トリガー情報（tinfo、0x7a4） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 51

5.2.6トリガ制御（tcontrol、0x7a5） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 51

5.2.7マシンコンテキスト（mcontext、0x7a8） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 52

5.2.8スーパーバイザーコンテキスト（scontext、0x7aa） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 52

5.2.9マッチ制御（mcontrol、0x7a1） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 53

5.2.10命令数（icount、0x7a1） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 58

5.2.11割り込みトリガ（itrigger、0x7a1） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 59

5.2.12例外トリガ（etrigger、0x7a1） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 60

5.2.13追加トリガ（RV32）（textra32、0x7a3）。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 60

5.2.14追加トリガ（RV64）（textra64、0x7a3） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 61

6デバッグトランスポートモジュール（DTM） 62

6.1 JTAGデバッグトランスポートモジュール。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 62

6.1.1 JTAGの背景。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 62

6.1.2 JTAG DTMレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 63

6.1.3 IDCODE（0x01） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 63

6.1.4 DTMの制御とステータス（dtmcs、0x10） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 64

6.1.5デバッグモジュールインタフェースアクセス（dmi、0x11） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 65

6.1.6バイパス（0x1f） 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 66

6.1.7推奨JTAGコネクタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 67

A ハードウェアの実装 69

A.1抽象コマンドベース。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 69

A.2実行ベース。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 69

Bデバッガの実装 71

B.1デバッグモジュールインタフェースアクセス。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 71

B.2停止中のハートの確認。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 72

B.3停止。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 72

B.4ランニング。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 72

B.5シングルステップ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 72

B.6レジスタへのアクセス。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 72

B.6.1抽象コマンドの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 72

B.6.2プログラムバッファの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 73

B.7メモリーの読み取り。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 73

B.7.1システムバスアクセスの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 73

B.7.2プログラムバッファの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 74

B.7.3抽象メモリアクセスの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 75

B.8メモリの書き込み。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 76

B.8.1システムバスアクセスの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 76

B.8.2プログラムバッファの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 76

B.8.3抽象メモリアクセスの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 77

B.9トリガー。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 78

B.10例外処理。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 79

B.11クイックアクセス。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 79

Cバグ修正80

C.1 0.13.1。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 80

C.1.1再開再開ビットは再開後に設定されます。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 80

C.1.2 aamsizeは引数の幅には影響しません。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 80

C.1.3 sbdata0は操作順序を読み取ります。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 80

C.1.4 haltreqが設定されている場合のハートリセットの動作 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 81

C.1.5 mteはaction = 0の場合にのみ適用されます。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 81

C.1.6 sselectはsvalueに適用されます。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 81

C.1.7最後のトリガーの例 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 81

Ｃ.２ ０.１３.２。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 81

インデックス 82

図の一覧

2.1 RISC-Vデバッグシステムの概要。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 6

3.1デバッグステートマシンの実行/停止。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 17

テーブル一覧

1.2アクセス略語の登録。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 3

3.1データレジスタの使用。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 11

3.2 cmdtypeの意味。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 12

3.3抽象レジスタ番号。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 13

3.7システムバスデータビット。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 18

3.8デバッグモジュールデバッグバスレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 20

4.1コアデバッグレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 42

4.3デバッグモード移行時のDPCの仮想アドレス。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 44

4.4仮想コアデバッグレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 45

4.5特権レベルのエンコーディング。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 46

5.1アクションエンコーディング。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 49

5.2トリガレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 49

5.8推奨されるブレークポイントのタイミング。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 53

6.1 JTAG DTM TAPレジスタ。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 63

6.5 MIPI-10コネクタ図。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 67

6.6 MIPI-20コネクタ図。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 67

6.7 JTAGコネクタピン配列。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 68

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）

31 8 7 0

|  |  |
| --- | --- |
| 0 | op |

24 8

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

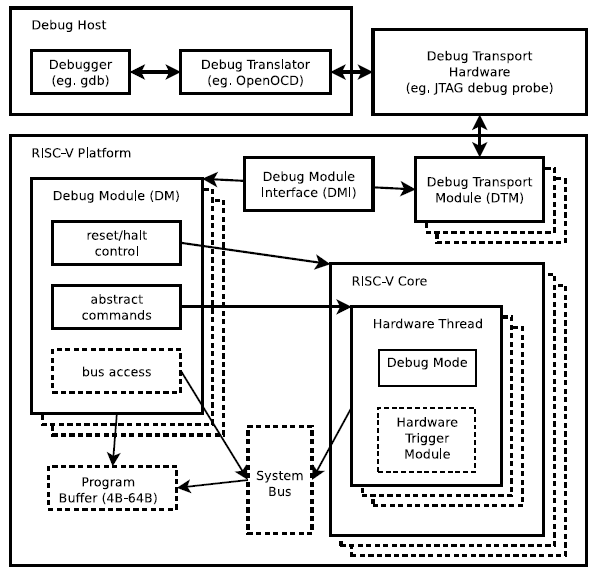


図2.1：RISC-Vデバッグシステムの概要

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220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220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。 |

31 24 23 22 20 19 18 17 16 15 0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cmdtype | 0 | aarsize | aarpostincrement |  | postexec | transfer | write | regno |

8 1 3 1 1 1 1 16

|  |  |
| --- | --- |
| 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。

31 24 23 0

|  |  |
| --- | --- |
| cmdtype | 0 |

8 24

|  |  |
| --- | --- |
| 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。

31 24 23 22 20 19

|  |  |  |  |
| --- | --- | --- | --- |
| cmdtype | aamvirtual | aamsize | aampostincrement |

8 1 3 1

18 17 16 15 14 13 0

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | write | target-specific | 0 |

2 1 2 14

|  |  |
| --- | --- |
| 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。

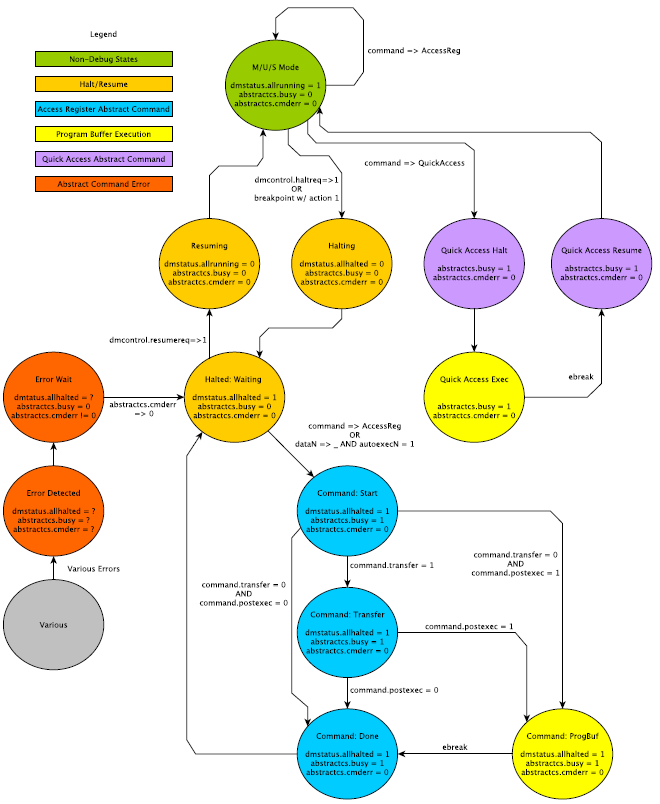


図3.1シングルハートトシステム用の実行/停止デバッグステートマシン

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

17 16 15 14 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| allresumeack | anyresumeack | allnonexistent | anynonexistent | allunavail |

1 1 1 1 1

12 11 10 9 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| anyunavail | allrunning | anyrunning | allhalted | anyhalted |

1 1 1 1 1

7 6 5 4 3 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| authenticated | authbusy | hasresethaltreq | confstrptrvalid | version |

1 1 1 1 4

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 30 29 28 27 26 25 16

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| haltreq | resumereq | hartreset | ackhavereset | 0 | hasel | hartsello |

1 1 1 1 1 1 10

15 6 5 4 3 2 1 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| hartselhi | 0 | setresethaltreq | clrresethaltreq | ndmreset | dmactive |

10 2 1 1 1 1

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 24 23 20 19 17 16 15 12 11 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | nscratch | 0 | dataaccess | datasize | dataaddr |

8 4 3 1 4 12

|  |  |  |  |
| --- | --- | --- | --- |
| 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。)

31 15 14 0

|  |  |
| --- | --- |
| 0 | hawindowsel |

17 15

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| maskdata |

32

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。

31 29 28 24 23 13 12 11 10 8 7 4 3 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | progbufsize | 0 | busy | 0 | cmderr | 0 | datacount |

3 5 11 1 1 3 4 4

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 24 23 0

|  |  |
| --- | --- |
| cmdtype | control |

8 24

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 16 15 12 11 0

|  |  |  |
| --- | --- | --- |
| autoexecprogbuf | 0 | autoexecdata |

16 4 12

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| addr |

32

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。

31 0

|  |
| --- |
| addr |

32

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。

31 0

|  |
| --- |
| addr |

32

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。

31 0

|  |
| --- |
| addr |

32

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。

31 0

|  |
| --- |
| data |

32

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。

31 0

|  |
| --- |
| haltsum0 |

32

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。

31 0

|  |
| --- |
| haltsum1 |

32

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。

31 0

|  |
| --- |
| haltsum2 |

32

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。

31 0

|  |
| --- |
| haltsum3 |

32

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

31 29 28 23 22 21 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sbversion | 0 | sbbusyerror | sbbusy | sbreadonaddr |

3 6 1 1 1

19 17 16 15 14 12 11 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sbaccess | sbautoincrement | sbreadondata | sberror | sbsize |

3 1 1 3 7

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。

31 0

|  |
| --- |
| address |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| address |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| address |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| address |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| data |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| data |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| data |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 0

|  |
| --- |
| data |

32

|  |  |  |  |
| --- | --- | --- | --- |
| 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  0x7b1  0x7b2  0x7b3 | デバッグ制御と状態 (dcsr)  デバッグ PC (dpc)  デバッグ スクラッチ レジスタ 0 (dscratch0)  デバッグ スクラッチ レジスタ 1 (dscratch1) | 42  44  45  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。

31 28 27 16 15 14 13 12 11 10

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| xdebugver | 0 | ebreakm | 0 | ebreaks | ebreaku | stepie | stopcount |

4 12 1 1 1 1 1 1

9 8 6 5 4 3 2 1 0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| stoptime | cause | 0 | mpven | nmip | step | prv |

1 3 1 1 1 1 2

|  |  |  |  |
| --- | --- | --- | --- |
| 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。)

DXLEN-1 0

|  |
| --- |
| dpc |

DXLEN

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。

1 0

|  |
| --- |
| prv |

2

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  1  3 | ユーザー / アプリケーション  監督者  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、2n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2m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  0x7a1  0x7a1  0x7a1  0x7a1  0x7a1  0x7a2  0x7a3  0x7a3  0x7a3  0x7a4  0x7a5  0x7a8  0x7aa | トリガー選択 (tselect)  トリガーデータ1 (tdata1)  一致制御 (mcontrol)  命令カウント (icount)  割り込みトリガー (itrigger)  例外トリガー (etrigger)  トリガーデータ2 (tdata2)  トリガーデータ3 (tdata3)  トリガー追加 (RV32) (textra32)  トリガー追加 (RV64) (textra64)  トリガー情報 (tinfo)  トリガー制御 (tcontrol)  マシンコンテキスト(属性) (mcontext)  スーパーバイザ(監督者)コンテキスト(属性) (scontext) | 49  50  53  58  59  60  50  51  60  61  51  51  52  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。

XLEN-1 0

|  |
| --- |
| index |

XLEN

– 2019/09/01

5.2.2トリガーデータ1（tdata1、0x7a1）

XLEN-1 XLEN-4 XLEN-5 XLEN-6 0

|  |  |  |
| --- | --- | --- |
| type | dmode | data |

4 1 XLEN-5

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

XLEN-1 0

|  |
| --- |
| data |

XLEN

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。

XLEN-1 0

|  |
| --- |
| data |

XLEN

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。

XLEN-1 16 15 0

|  |  |
| --- | --- |
| 0 | info |

XLEN-16 16

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

XLEN-1 8 7 6 4 3 2 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | mpte | 0 | mte | 0 |

XLEN-8 1 3 1 3

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

XLEN-1 0

|  |
| --- |
| mcontext |

XLEN

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

XLEN-1 0

|  |
| --- |
| data |

XLEN

|  |  |  |  |
| --- | --- | --- | --- |
| 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  命令を実行する  アドレス+命令の実行  ロードアドレス  データを読み込む  アドレス+データの読み込み  ストアアドレス  データを保存する  ストアアドレス+データ | 前  前  前  前  後  後  前  前  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263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。

XLEN-1 XLEN-4 XLEN-5 XLEN-6 XLEN-7 10 9 8 7 6 5 0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| type | dmode | hit | 0 | m |  | 0 | s | u | action |

4 1 1 XLEN-16 1 1 1 1 6

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

XLEN-1 XLEN-4 XLEN-5 XLEN-6 XLEN-7 10 9 8 7 6 5 0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| type | dmode | hit | 0 | m |  | 0 | s | u | action |

4 1 1 XLEN-16 1 1 1 1 6

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 26 25 24 18 17 2 1 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mvalue | mselect | 0 | svalue | sselect |

6 1 7 16 2

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

63 51 50 49 36 35 2 1 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mvalue | mselect | 0 |  | svalue | sselect |

13 1 14 34 2

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

31 28 27 12 11 1 0

|  |  |  |  |
| --- | --- | --- | --- |
| Version | PartNumber | Manufld | 1 |

4 16 11 1

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

31 18 17 16 15 14 12 11 10 9 4 3 0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | dmihardreset | dmireset | 0 |  | idle | dmistat | abits | version |

14 1 1 1 3 2 6 4

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

abits+33 34 33 2 1 0

|  |  |  |
| --- | --- | --- |
| address | data | op |

abits 32 2

|  |  |  |  |
| --- | --- | --- | --- |
| 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。

0

|  |
| --- |
| 0 |

1

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

aampostincrement, 15

aamsize, 15

aamvirtual, 15

aarpostincrement, 13

aarsize, 13

abits, 65

abstractauto, 29

abstractcs, 27

Access Memory, 14

Access Register, 12

ackhavereset, 23

action, 56, 59, 60

address, 35, 36, 65

allhalted, 21

allhavereset, 21

allnonexistent, 21

allresumeack, 21

allrunning, 21

allunavail, 21

anyhalted, 21

anyhavereset, 21

anynonexistent, 21

anyresumeack, 21

anyrunning, 21

anyunavail, 21

authbusy, 22

authdata, 31

authenticated, 22

autoexecdata, 29

autoexecprogbuf, 29

busy, 27

BYPASS, 66

cause, 43

chain, 57

clrresethaltreq, 24

cmderr, 28

cmdtype, 13{15, 29

command, 28

confstrptr0, 29

confstrptrvalid, 22

control, 29

count, 58

data, 37, 38, 50, 53, 65

data0, 30

dataaccess, 26

dataaddr, 26

datacount, 28

datasize, 26

dcsr, 42

dmactive, 25

dmcontrol, 22

dmi, 65

dmihardreset, 64

dmireset, 64

dmistat, 65

dmode, 50

dmstatus, 20

dpc, 44

dscratch0, 45

dscratch1, 45

dtmcs, 64

ebreakm, 42

ebreaks, 42

ebreaku, 42

etrigger, 60

execute, 58

field, 2

haltreq, 23

haltsum0, 31

haltsum1, 31

haltsum2, 32

haltsum3, 32

hartinfo, 25

hartreset, 23

hartsel, 22

hartselhi, 24

hartsello, 24

hasel, 24

hasresethaltreq, 22

hawindow, 26

hawindowsel, 26

hit, 54, 58-60

icount, 58

IDCODE, 63

idle, 64

impebreak, 21

info, 51

itrigger, 59

load, 58

m, 57-60

ManufId, 64

maskmax, 54

match, 57

mcontext, 52

mcontrol, 53

mprven, 43

mpte, 52

mselect, 61

mte, 52

mvalue, 61

ndmreset, 24

nextdm, 30

nmip, 43

nscratch, 25

op, 66

PartNumber, 64

postexec, 13

priv, 45

progbuf0, 30

progbufsize, 27

prv, 44, 46

Quick Access, 14

regno, 13

resethaltreq, 22

resumereq, 23

s, 57-60

sbaccess, 33

sbaccess128, 34

sbaccess16, 34

sbaccess32, 34

sbaccess64, 34

sbaccess8, 34

sbaddress0, 34

sbaddress1, 35

sbaddress2, 35

sbaddress3, 36

sbasize, 34

sbautoincrement, 33

sbbusy, 33

sbbusyerror, 33

sbcs, 32

sbdata0, 36

sbdata1, 37

sbdata2, 37

sbdata3, 38

sberror, 34

sbreadonaddr, 33

sbreadondata, 33

sbversion, 33

scontext, 52

select, 54

setresethaltreq, 24

shortname, 2

sizehi, 54

sizelo, 56

sselect, 61

step, 44

stepie, 42

stopcount, 43

stoptime, 43

store, 58

svalue, 61

target-specific, 15

tcontrol, 51

tdata1, 50

tdata2, 50

tdata3, 51

textra32, 60

textra64, 61

timing, 55

tinfo, 51

transfer, 13

tselect, 49

type, 50

u, 57-60

Version, 63

version, 22, 65

write, 13, 15

xdebugver, 42

--2019/09/23

--翻訳完了