アプリケーションノート 38

ARM7TDMIデバッグ通信チャネルの使用

文書番号：ARM DAI 0038BJ-00

発行：1999**年**11**月**

ENGLAND
ARM Ltd.
90 Fulbourn Road
Cambridge
CB1 4JN
UK
Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: info@arm.com

GERMANY
ARM Ltd.
Otto-Hahn Str. 13b
85521 Ottobrunn-Riemerling
Munich
Germany
Telephone: +49 89 608 75545
Facsimile: +49 89 608 75599
Email: info@arm.com

JAPAN
ARM K.K.
Plustaria Bldg. 4F, 3-1-4 Shinyokohama
Kohoku-ku, Yokohama-shi
Kanagawa
222-0033 Japan
Telephone: +81 45 477 5260
Facsimile: +81 45 477 5261
Email: info@arm.com

USA
ARM, INC.
Suite 5
985 University Avenue
Los Gatos
CA 95030 USA
Telephone: +1 408 399 5199
Facsimile: +1 408 399 8854
Email: info@arm.com

World Wide Web address: http://www.arm.com

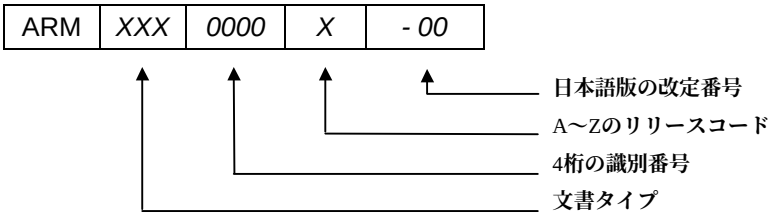
ARM
Advanced RISC Machines

著作権について

記号

文書番号

本書には他の文書と識別するための文書番号が付いています。この文書番号は表紙と各ページの一番下に表示されています。



文書ステータス

文書ステータスは各ページ下のバナーに表示されています。このステータスは、文書の機密性と記載されている情報の現況を示しています。

機密性を表すステータスには以下の4つがあります。

ARM Confidential	ARM社員およびNDA署名者だけが閲覧可。
Named Partner Confidential	上記および指定提携企業の社員だけが閲覧可。
Partner Confidential	ARM社内およびすべての提携企業に配布可。
Open Access	制約なし。

Information status is one of:

Advance	開発予定の製品に関する情報
Preliminary	開発中の製品に関する現時点での情報
Final	開発が終了した製品に関する最終的な情報

改訂記録

英語版

発行	日付	発行者	改訂
B	1998年1月	SKW	初版

日本語版

発行	日付	改訂
BJ-00	1999年11月	初版

目次

1 はじめに

ARM7TDMIのEmbeddedICEマクロセルにはデバッグ通信チャネルが組み込まれています。これにより、JTAGポートとEmbeddedICEインターフェースを使用して、プログラムフローを停止させたり、デバッグ状態に入ったりすることなく、ターゲットとホストデバッガ間でデータの受け渡しを行うことができます。本アプリケーションノートでは、ターゲットで実行中のプログラムとホストデバッガからのデバッグ通信チャネルへのアクセス方法について説明します。

SDT 2.11を使用する場合、デバッグ通信チャネルへのアクセスには以下の2方法があります。

- コマンドラインデバッガ（`armsd`またはWindows対応ARMデバッガのコマンドウィンドウ）
- Windows対応ARMデバッガのチャネルビューワ機能

Note *本アプリケーションノートに記載している機能を利用するには、必ずSDT2.11（またはこれ以降）、EmbeddedICEエージェントソフトウェアVer.2.04（またはこれ以降）、ならびにGAL Ver.EFI-0011Cを使用してください。*

Important Note *現時点において、STD2.11 Windowsツール ARMsd ver.4.48[最終ビルド、1997年9月9日]ならびにADW2.11[最終ビルド、1997年9月9日]では`ccin`および`ccout`をサポートしていません。サポートについては今後のバージョンで変更予定です。*

ARM7TDMIでEmbeddeICEが提供するデバッグ機能について詳しくは、以下の資料を参照してください。

- *Application Note 28: The ARM7TDMI Debug Architecture* (ARM DAI 0028)
- *Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 7 EmbeddedICE***

2 コマンドラインからのデバッグコマンド

コマンドラインからデバッグ通信チャネルにアクセスするには、以下のコマンドを使用します。

`ccin <filename>` 読み出し用通信チャネルデータを含むファイルを選択します。同時に、ホストからターゲットへの通信チャネル通信もイネーブルします。

`ccout <filename>` 書き込み用通信チャネルデータを含むファイルを選択します。同時に、ターゲットからホストへの通信チャネル通信もイネーブルします。

Note *SDT 2.11では、ccinを使用してもデバッグ通信チャネルが正しくイネーブルされません。これを避けるため、アプリケーションがターゲットからホストへの通信を要求しない場合でも、ccinを使用する際は必ずccoutコマンドも使用してください。*

3 Windows対応ARMデバッガ(ADW: ARM Debugger for Windows)のチャネルビューワ

3.1 チャネルビューワの起動

ADWにおいてデバッグ通信チャネルビューワを起動するには：

1. ADW起動後、**[Options]→[Configure Debugger]**を選択します。
2. 接続リストから、**Remote_A** RDI DLLを選択します。
3. **[Configure]**ボタンをクリックしてRDI接続設定を変更します。ダイアログボックスの一番下に、チャネルビューワの欄があります(***図1：[Angel Remote Configuration]ダイアログボックス***参照)。

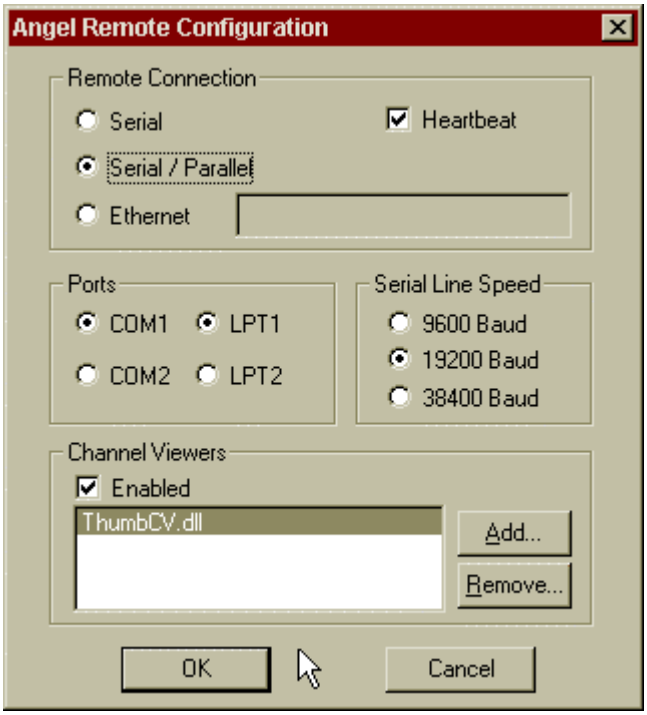


図1：[Angel Remote Configuration]ダイアログボックス

4. チャネルビューワDLLを追加する場合は、**[Add]**ボタンをクリックして適切なDLLを選択し、**[OK]**をクリックします。
5. リストからチャネルビューワDLLを削除する場合は、削除したいDLLを反転表示させ、**[Remove]**をクリックします。
6. チャネルビューワDLLをイネーブルするには、必ず**[Enabled]**ボックスにチェックマークを入れ、リスト内の適切なDLLが反転表示されていることを確認してください。
7. [Angel Remote Configuration]ダイアログボックスと[Debugger Configuration]ダイアログボックスの両方で**[OK]**をクリックすると、チャネルビューワがアクティブな状態でADWが再起動します。

3.2 ユーザーインターフェース

デバッグ通信チャネルビューワには以下のメニューがあります。

Control	**Start Viewer**	チャネルビューの開始
	Pause Viewer	チャネルビューの停止
	Exit	チャネルビューの終了
Options	**Clear Display**	出力表示の消去
	Clear Send Buffer	送信バッファの消去
	Save Contents	表示内容のファイルへの保存
	Change Font	表示フォントの変更

ウィンドウの一番下には出し入れ可能なダイアログバーがあり、ホストデバッガからターゲットで実行中のプログラムへのデータ送信ができます。***図2：ADWチャネルビューワ***を参照してください。

図2：ADWチャネルビューワ

3.3 ターゲットからデバッガへ (データの受信)

チャネルビューワがアクティブのとき、チャネルビューワが32ビットワードとして受信したデータはASCII文字コードに変換され、ウィンドウ内にテキストで表示されます。

しかし、ワード「`0xffffffff`」を受信すると、その後のワードはASCIIテキストではなく、16進数で表示されます。

3.4 デバッガからターゲットへ (データの送信)

[Edit]ボックスにテキストを入力して**[Send]**ボタンをクリック(または**[Return]**キーを押す)と、テキストを32ビットデータとしてバッファに保存できます。デバッガは通信データ書き込みレジスタが空であることを検出すると、このデータを1ワードずつ送信します。**[Left to Send]**カウンタはバッファに残っているバイト数を表示し、テキストを32ビットワードに変換します。このデータはターゲットの要求に応じて送信されます。

4 ターゲットのデータ転送

ARM7TDMIデバッグ通信チャネルは、MCRとMRCのARM命令を使用して、ARM7TDMIコア上のコプロセッサ14としてターゲットからアクセスできます。データ転送には以下の2つのレジスタを使用します。

通信データ読み出しレジスタ デバッガからデータを受信する32ビット幅レジスタ。以下のARM命令は、この読み出しレジスタの値をRdに返します。

```
MRC p14, 0, Rd, c1, c0
```

通信データ書き込みレジスタ デバッガへデータを送信する32ビット幅レジスタ。以下のARM命令は、この書き込みレジスタにRnの値を書き込みます。

```
MCR p14, 0, Rn, c1, c0
```

5 デバッグ通信のポーリング

デバッグ通信チャネルでは、通信データ読み出し/書き込みレジスタに加え、通信データ制御レジスタも使用できます。

制御レジスタの値をRdに返すには以下の命令を使用します。

```
MRC p14, 0, Rd, c0, c0
```

この制御レジスタの2ビットが、ターゲットとホストデバッガ間の同期ハンドシェークを行います。

ビット1(Wビット)通信データ書き込みレジスタが(ターゲットから見て)空いているかどうかを示します。

- W = 0 ターゲットアプリケーションが新しいデータを書き込むことができます。
- W = 1 ホストデバッガが書き込みレジスタから新しいデータをスキャンできます。

ビット0(Rビット)通信データ読み出しレジスタに(ターゲットから見て)新しいデータが存在するかどうかを示します。

- R = 1 ターゲットアプリケーションが新しいデータを読み出すことができます。
- R = 0 ホストデバッガが読み出しレジスタへの新しいデータをスキャンできます。

Note *デバッガが、コプロセッサ14を使って直接デバッグ通信チャネルにアクセスすることはできません。それはデバッガにとって無意味です。その代わり、デバッガはスキャンチェインを使用して、デバッグ通信チャネルレジスタとの間で読み出し、書き込みを行うことができます。デバッグ通信チャネルのデータレジスタおよび制御レジスタは、EmbeddedICEマクロセル内のアドレスにマップされます。*

5.1 ターゲットからデバッガへの通信

以下に、ARM7TDMIコア上で動作中のアプリケーションがホスト上で動作中のデバッガと通信する際のプロセスを示します。

1. ターゲットアプリケーションが、デバッグ通信チャネル書き込みレジスタが空いているかどうかをチェックします。すなわち、MRC命令を使用してデバッグ通信チャネル制御レジスタを読み、Wビットがクリアかどうかを確認します。
2. Wビットがクリアなら、デバッグ通信チャネル書き込みレジスタも空いていますから、アプリケーションはコプロセッサ14へのMCR命令によってワードを書き込みます。

   レジスタへの書き込み動作は、自動的にWビットをセットします。Wビットがセットされていれば、デバッガがデバッグ通信チャネル書き込みレジスタを空けていないことになります。アプリケーションが別のワードを送信する必要があるときは、Wビットがクリアになるまでポーリングを続けなくてはなりません。
3. デバッガはスキャンチェイン2を介してデバッグ通信制御レジスタにポーリングを行います。デバッガは、Wビットがセットされていることを検出すると、デバッガ通信チャネルデータレジスタを読み、アプリケーションが送信したメッセージを読み出すことができます。データの読み出しにより、デバッグ通信制御レジスタ内のWビットは自動的にクリアされます。

以下のターゲットアプリケーションコードの一部は、前述のプロセスを示しています。

```
        AREA   OutChannel, CODE, READONLY
        ENTRY
        MOV    r1,#4         ; Number of words to send
        ADR    r2, outdata   ; Address of data to send
pollout
        MRC    p14,0,r0,c0,c0 ; Read control register
        TST    r0, #2
        BNE    pollout       ; if W set, register
                             ;  still full
write
        LDR    r3,[r2],#4    ; Read word from outdata
                             ;  into r3 and update the
                             ;  pointer
        MCR    p14,0,r3,c1,c0 ; Write word from r3
        SUBS   r1,r1,#1      ; Update counter
        BNE    pollout       ; Loop if more words to
                             ;  be written
        MOV    r0, #0x18     ; Angel_SWIreason_ReportException
        LDR    r1, =0x20026  ; ADP_Stopped_ApplicationExit
        SWI    0x123456      ; Angel semihosting SWI
outdata
        DCB    "Hello there!"
        END
```

4 このコードは、以下のコマンドを使用してアセンブルし、リンクできます。

```
armasm -g outchan.s

armlink outchan.o -o outchan
```

5.1.1 コマンドラインを使用する場合

1 以下のコマンドを使用して、イメージをarmsdにロードします。

```
armsd -li -adp -port s=1 outchan
```

2 通信をイネーブルして出力ファイルを開き、プログラムを実行します。

```
ccout output

go
```

3 実行が終了したらarmsdを閉じます。ファイルを表示させ、転送が行われたことを確認できます。

5.1.2 ADWチャネルビューワを使用する場合

1 上記で作成したイメージをWindows対応ARMデバッガにロードし、チャネルビューワを起動します(P.4「*3.1 チャネルビューワの起動* 参照)。

2 チャネルビューワウィンドウで、メニューから**[Control]→[Start Viewer]**を選択してデバッグ通信チャネルをイネーブルします。

3 メニューから**[Execute]→[Go]**を選択し、ADW内でプログラムを実行します。

ターゲットから送信されたデータ(上記の例では `Hello there!`)が、チャネルビューワウィンドウに表示されます。

5.2 デバッガからターゲットへの通信

以下に、ホスト上で動作中のデバッガからコア上で動作中のアプリケーションへメッセージを転送する際のプロセスを示します。

1 デバッガが、デバッグ通信制御レジスタのRビットをポーリングします。Rビットがクリアなら、デバッグ通信読み出しレジスタもクリアですから、データを書き込んでターゲットアプリケーションに読み出させることができます。

2 デバッガはスキャンチェイン2を介してデバッグ通信読み出しレジスタに入れるデータをスキャンします。これで、デバッグ通信制御レジスタのRビットが自動的にセットされます。

3 ターゲットアプリケーションが、デバッグ通信制御レジスタのRビットをポーリングします。このビットがセットされていれば、デバッグ通信読み出しレジスタにデータが存在し、アプリケーションはMRC命令を使用してコプロセッサ14から読み出すことができます。この読み出し命令により、Rビットはクリアされます。

以下のターゲットアプリケーションコードは、上記のプロセスを示しています。

```
        AREA   InChannel, CODE, READONLY
        ENTRY
        MOV    r1,#4         ; Number of words to read
        LDR    r2, =indata   ; Address to store data
                             ;  read
pollin
        MRC    p14,0,r0,c0,c0 ; Read control register
        TST    r0, #1
        BEQ    pollin        ; If R bit clear then
                             ;  loop
read
        MRC    p14,0,r3,c1,c0 ; read word into r3
        STR    r3,[r2],#4    ; Store to memory and
                             ;  update pointer
        SUBS   r1,r1,#1      ; Update counter
        BNE    pollin        ; Loop if more words to
                             ;  read
        MOV    r0, #0x18     ; Angel_SWIreason_ReportException
        LDR    r1, =0x20026  ; ADP_Stopped_ApplicationExit
        SWI    0x123456      ; Angel ARM semihosting
                             ;  SWI

        AREA Storage, DATA, READWRITE
indata
        DCB    "Duffmessage#"
        END
```

4 一例として「And goodbye!」を含む入力ファイルをホスト上で作成します。

5 このコードを以下のコマンドを用いてアセンブルし、リンクさせます。

```
armasm -g inchan.s

armlink inchan.o -o inchan
```

5.2.1 コマンドラインを使用する場合

1 以下のコマンドを用いてイメージをarmsdにロードします。

```
armsd -li -adp -port s=1 inchan
```

メモリ領域indataを表示させると、初期のランダムな内容を見ることができます。

```
examine indata
```

2 通信をイネーブルにして入力ファイルを開き、プログラムを実行します。

```
ccin input
ccout output
go
```

3 実行終了時に再度メモリを表示させると、以下で入力が読み出されたことを確認することができます。

```
examine indata
```

Note *これはホストからターゲットへの通信ですが、デバッグ通信チャネルを正しく開くには ccoutコマンドを使用する必要があります。*

5.2.2 ADWチャネルビューワを使用する場合

1 上記で作成したイメージをWindows対応ARMデバッガにロードし、チャネルビューワを起動します(P.4「*3.1 チャネルビューワの起動*」参照)。

2 チャネルビューワウィンドウで、メニューから**[Control]→[Start Viewer]**を選択してデバッグ通信チャネルをイネーブルします。

3 チャネルビューワのダイアログバーにある**[Edit]**ボックスに**「And goodbye」**を入力し、**[Send]**ボタンをクリックします。**[Left to Send]**カウンタは、ターゲットに送信するためにストア中のバイト数を表示します。

メモリ領域indataを表示させると、初期の内容を見ることができます。

```
examine indata
```

4 メニューから**[Execute]→[Go]**を選択し、ADW内でプログラムを実行します。

5 実行終了時に再度メモリを表示させると、入力が読み込まれたことを確認できます。

```
examine indata
```

6 割り込み駆動型デバッグ通信

これまではポーリングによる通信例を見てきましたが、ARM7TDMIコアから割り込みコントローラにCOMMRXおよびCOMM信号を接続することによって、割り込み駆動型に変換することもできます。

割り込み駆動型の通信では、前述のreadおよびwriteコードは割り込みハンドラ内に移動することができます。

割り込みハンドラの記述に関する詳しい情報については、*Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 11 Exception Handling*** を参照してください。

7 Thumb状態からのアクセス

Thumb命令セットにはコプロセッサ命令が含まれていないため、コアがThumb状態にあるときはデバッグ通信チャネルを使用できません。

これには以下の3つの解決策があります。

- 各ポーリングルーチンを、ARM状態またはThumb状態のどちらでも実行可能なSWI(ソフトウェア割り込み)として記述します。SWIハンドラに入るとコアはただちにARM状態に入り、コプロセッサ命令を使用できるようになります。SWIの詳細については、*Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 11 Exception Handling*** を参照してください。
- Thumbコードは、ポーリングを行うARMサブルーチンへのインターワーキング呼び出しを行うことができます。ARMおよびThumbの混合コードについては、*Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 12 Interworking ARM and Thumb*** を参照してください。
- ポーリング型通信ではなく、割り込み駆動型通信を用います。割り込みハンドラをARM命令で記述すれば、コプロセッサ命令に直接アクセスできます。