Application Note 38

Using the ARM7TDMI Debug Comms Channel

Document number: ARM DAI 0038B

Issued: January 1998

Copyright Advanced RISC Machines Ltd (ARM) 1998

ENGLAND

Advanced RISC Machines Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
UK
Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: info@arm.com

GERMANY

Advanced RISC Machines Limited
Otto-Hahn Str. 13b
85521 Ottobrunn-Riemerling
Munich
Germany
Telephone: +49 89 608 75545
Facsimile: +49 89 608 75599
Email: info@arm.com

JAPAN

Advanced RISC Machines K.K.
KSP West Bldg, 3F 300D, 3-2-1 Sakado
Takatsu-ku, Kawasaki-shi
Kanagawa
213 Japan
Telephone: +81 44 850 1301
Facsimile: +81 44 850 1308
Email: info@arm.com

USA

ARM USA Incorporated
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

Open Access

Proprietary Notice

ARM and the ARM Powered logo are trademarks of Advanced RISC Machines Ltd.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

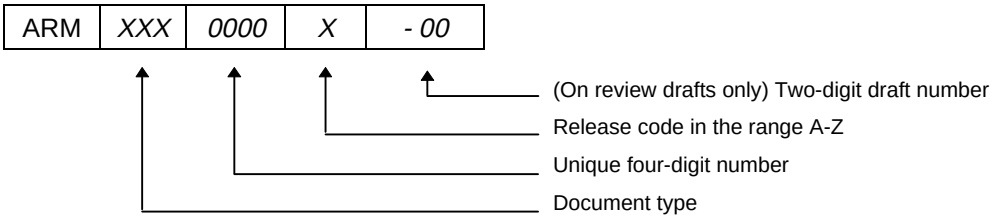
The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties or merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Ltd shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Key

Document Number

This document has a number which identifies it uniquely. The number is displayed on the front page and at the foot of each subsequent page.



Document Status

The document's status is displayed in a banner at the bottom of each page. This describes the document's confidentiality and its information status.

Confidentiality status is one of:

ARM Confidential	Distributable to ARM staff and NDA signatories only
Named Partner Confidential	Distributable to the above and to the staff of named partner companies only
Partner Confidential	Distributable within ARM and to staff of all partner companies
Open Access	No restriction on distribution

Information status is one of:

Advance	Information on a potential product
Preliminary	Current information on a product under development
Final	Complete information on a developed product

Change Log

Issue	Date	By	Change
B	January 1998	SKW	Released

Table of Contents

Open Access

1 Introduction

The EmbeddedICE macrocell in the ARM7TDMI contains a debug communication channel. This allows data to be passed between the target and the host debugger using the JTAG port and an EmbeddedICE interface, without stopping the program flow or entering debug state. This Application Note examines how the debug communication channel can be accessed by a program running on the target and by the host debugger.

With SDT 2.11, there are two methods of accessing the debug communication channel:

- The command line debugger (`armsd` or the command window in the ARM Debugger for Windows)
- The Channel Viewer mechanism in the ARM Debugger for Windows.

Note *If you wish to make use of the facilities described in this Application Note, ensure that you are using SDT2.11 or later, EmbeddedICE agent software version 2.04 or later, and GAL version EFI-0011C.*

Important Note *`ccin` and `ccout` are currently not supported for the SDT2.11 Windows tools ARMsd version 4.48 [last build 9 Sept 1997] and ADW2.11 [last build 9 Sept 1997]. This will be changed in a future release.*

For further information on the debug facilities provided by EmbeddedICE on the ARM7TDMI, see:

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

Open Access

2 Command Line Debugging Commands

To access the debug communication channel from the command line, use the following commands:

ccin <filename>	Selects a file containing comms channel data for reading. This command also enables host to target comms channel communication.
ccout <filename>	Selects a file where comms channel data is written. This command also enables target to host comms channel communication.

Note *In SDT 2.11, `ccin` does not correctly enable the debug communication channel. As a workaround, when using `ccin`, you must also use a `ccout` command, even if Target to Host communication is not required by the application.*

Open Access

3 ARM Debugger for Windows Channel Viewer

3.1 Activating a channel viewer

To activate the debug communication channel viewer in ADW:

1. After starting ADW, select **Options** then **Configure Debugger**.
2. Select the **Remote_A** RDI DLL from the connections list.
3. Select the **Configure** button to change the RDI connection settings. At the bottom of this dialog there is a section for channel viewers, as shown in ***Figure 1: Angel Remote Configuration dialog***.

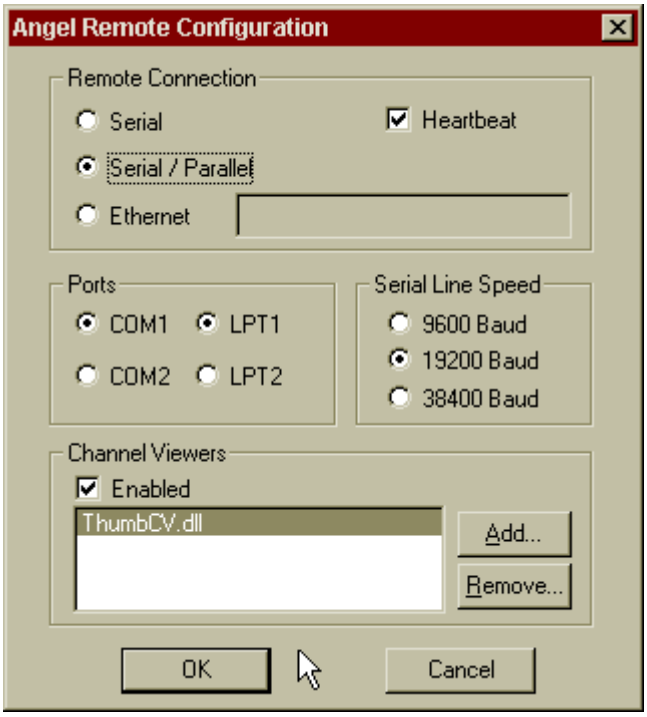


Figure 1: Angel Remote Configuration dialog

4. To add a channel viewer DLL, click the **Add** button, select the appropriate DLL, and click **OK**.
5. To remove a channel viewer DLL from the list, highlight the DLL that you wish to remove and click the **Remove** button.
6. To enable a channel viewer DLL, ensure that the **Enabled** box is checked, and that the appropriate DLL in the list is highlighted.
7. Click **OK** for both the Angel Remote Configuration dialog and the Debugger Configuration dialog. ADW restarts with an active channel viewer.

Open Access

3.2 The user interface

The debug communication channel viewer has the following menu structure:

Control	**Start Viewer**	starts viewing the channel
	Pause Viewer	stops viewing the channel
	Exit	quits the channel viewer
Options	**Clear Display**	clears the output display
	Clear Send Buffer	clears the send buffer
	Save Contents	saves the contents of the window to a file
	Change Font	changes the font in the window

The window has a dockable dialog bar at the bottom of the window, which is used to send data from the host debugger to a program running on the target. The ADW Channel Viewer is shown in ***Figure 2: ADW Channel Viewer***.

Figure 2: ADW Channel Viewer

3.3 Target to debugger (receiving data)

The data that is received by the Channel Viewer as 32-bit words is converted into ASCII character codes and displayed in the window as text, if the channel viewers are active.

However, if the word `0xffffffff` is received, the following word is displayed as a hexadecimal number, not as ASCII text.

3.4 Debugger to target (sending data)

Type text in the **Edit** box and click the **Send** button (or press **Return**) to store the text in a buffer as 32-bit data. The data is sent a word at a time when the debugger detects that the comms data write register is free. The **Left to Send** counter displays the number of bytes that are left in the buffer, and the text is converted into 32-bit words. This data is sent when requested by the target.

Open Access

4 Target Transfer of Data

The ARM7TDMI debug communication channel is accessed by the target as coprocessor 14 on the ARM7TDMI core using the ARM instructions `MCR` and `MRC`. Two registers are provided to transfer data:

Comms data read register	A 32-bit wide register used to receive data from the debugger. The following ARM instruction returns the read register value in `Rd`: `MRC p14, 0, Rd, c1, c0`
Comms data write register	A 32-bit wide register used to send data to the debugger. The following instruction writes the value in `Rn` to the write register: `MCR p14, 0, Rn, c1, c0`

Open Access

5 Polled Debug Communications

In addition to the comms data read and write registers, a comms data control register is provided by the debug communication channel.

The following instruction returns the control register value in Rd:

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

Two bits in this control register provide synchronized handshaking between the target and the host debugger:

Bit 1 (W bit) Denotes whether the comms data write register is free (from the target's point of view):

- W = 0 New data may be written by the target application.
- W = 1 The host debugger can scan new data out of the write register.

Bit 0 (R bit) Denotes whether there is some new data in the comms data read register (from the target's point of view):

- R = 1 New data is available to be read by the target application.
- R = 0 The host debugger can scan new data into the read register.

Note *The debugger cannot use coprocessor 14 to access the debug communication channel directly, as this has no meaning to the debugger. Instead, the debugger can read from and write to the debug communication channel registers using the scan chain. The debug communication channel data and control registers are mapped into addresses in the EmbeddedICE macrocell.*

5.1 Target to debugger communication

This is the sequence of events for an application running on the ARM7TDMI core to communicate with the debugger running on the host:

1. The target application checks if the debug communication channel write register is free for use. It does this using the MRC instruction to read the debug communication channel control register to check that the W bit is clear.
2. If the W bit is clear, the debug communication write register is clear and the application writes a word to it using the MCR instruction to coprocessor 14.

   The action of writing to the register automatically sets the W bit. If the W bit is set, the debug communication write register has not been emptied by the debugger. If the application needs to send another word, it must poll the W bit until it is clear.
3. The debugger polls the debug communication control register via scan chain 2. If the debugger sees that the W bit is set, it can read the debug communication channel data register to read the message sent by the application. The process of reading the data automatically clears the W bit in the debug communication control register.

Open Access

The following piece of target application code shows this in action:

```
        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 Assemble and link this code using the following commands:

```
armasm -g outchan.s

armlink outchan.o -o outchan
```

5.1.1 Using the command line

1 Load the image into `armsd` using the following command:

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

2 Enable communication and open the output file, then execute the program:

```
ccout output

go
```

3 Quit `armsd` when execution finishes. You should be able to view the file and see that transfer has occurred.

5.1.2 Using the ADW Channel Viewer

1 Load the image created above into the ARM Debugger for Windows, and activate the Channel Viewer, as described in ***3.1 Activating a channel viewer*** on page 4.

2 In the Channel Viewer window, select **Control** then **Start Viewer** from the menu, to enable the debug communication channel.

3 Select **Execute** then **Go** from the menu to execute the program in ADW.

The data sent from the target (in this example, `Hello there!`) should now be displayed in the Channel Viewer window.

Open Access

5.2 Debugger to target communication

This is the sequence of events for message transfer from the debugger running on the host to the application running on the core:

1. The debugger polls the debug communication control register R bit. If the R bit is clear, the debug communication read register is clear and data can be written there for the target application to read.
2. The debugger scans the data into the debug communication read register via scan chain 2. The R bit in the debug communication control register is automatically set by this.
3. The target application polls the R bit in the debug communication control register. If it is set, there is data in the debug communication read register that can be read by the application, using the MRC instruction to read from coprocessor 14. The R bit is cleared as part of the read instruction.

The following piece of target application code shows this in action:

```
        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. Create an input file on the host containing, for example, `And goodbye!`.
5. Assemble and link this code using the following commands:

```
armasm -g inchan.s

armlink inchan.o -o inchan
```


Open Access

5.2.1 Using the command line

1 Load the image into `armsd` using the following command:

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

If you view the area of memory `indata`, you see its initial random contents:

```
examine indata
```

2 Enable communication and open the input file, then execute the program:

```
ccin input
ccout output
go
```

3 When execution completes, view memory again and you can see the input has been read in:

```
examine indata
```

Note *A `ccout` command is required, even though this is Host to Target communication, in order to open up the debug communication channel correctly.*

5.2.2 Using the ADW Channel Viewer

1 Load the image created above into the ARM Debugger for Windows, and activate the Channel Viewer (as described in ***3.1 Activating a channel viewer*** on page 4).

2 In the Channel Viewer window, select **Control** then **Start Viewer** from the menu to enable the debug communication channel.

3 In the **Edit** box on the dialog bar of the Channel Viewer, type **And goodbye**, and click the **Send** button. The **Left to Send** counter should show the number of bytes stored for sending to the target.

If you view the area of memory `indata`, you see its initial contents:

```
examine indata
```

4 Execute the program in ADW by selecting **Execute** then **Go** from the menu.

5 When execution is complete, view memory again and you can see that the input has been read in:

```
examine indata
```


Open Access

6 Interrupt-Driven Debug Communications

The examples given above are polled. It is also possible to convert these to interrupt-driven examples by connecting up **COMMRX** and **COMMTX** signals from the ARM7TDMI core to your interrupt controller.

The `read` and `write` code given above could then be moved into an interrupt handler.

For information on writing interrupt handlers refer to the *Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 11 Exception Handling***.

Open Access

7 Access from Thumb State

As the Thumb instruction set does not contain coprocessor instructions, you cannot use the debug communication channel while the core is in Thumb state.

There are three possible ways around this:

- You can write each polling routine as a SWI (Software Interrupt), which can then be executed while in either ARM or Thumb state. Entering the SWI handler immediately puts the core into ARM state where the coprocessor instructions are available. Refer to the *Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 11 Exception Handling*** for further information on SWIs.
- Thumb code can make interworking calls to ARM subroutines which implement the polling. Refer to the *Software Development Toolkit User Guide* (ARM DUI 0040), ***Chapter 12 Interworking ARM and Thumb*** for further information on mixing ARM and Thumb code.
- Use interrupt-driven communication rather than polled communication. The interrupt handler would be written in ARM instructions, so the coprocessor instructions can be accessed directly.

Open Access