J-Link RDI

JTAG RDI Interface for J-Link ARM Emulator

Software version 3.78
Document revision 0
Date: December 4, 2007

SEGGER

A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH & Co. KG (the manufacturer) assumes no responsibility for any errors or omissions. The manufacturer makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. The manufacturer specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH & Co. KG

Heinrich-Hertz-Str. 5
D-40721 Hilden

Germany

Tel.+49 2103-2878-0
Fax.+49 2103-2878-28
Email: support@segger.com
Internet: http://www.segger.com

Manual versions

This manual describes the latest software version. If any error occurs, please inform us and we will try to assist you as soon as possible.

For further information on topics or routines not yet specified, please contact us.

Manual version	Date	By	Explanation
9	0709019	SK	Chapter "Configuration": * Section"Configuration file JLinkRDI.ini" included.
8	070803	SK	Chapter "Using J-Link RDI with different debuggers": * Section"ARM's RVDS" updated.
7	070115	SK	Chapter "Using J-Link RDI with different debuggers": * Section "KEIL µVision3 IDE" added. Chapter "Configuration": * Some changes in chapter structure. * Section "Location of Config file" renamed to "Config file" * Section "Config file" enhanced.
6	061221	SK	Preface: Company description added.
5	061106	SK	Section "Using GHS Multi" added. Section "License (J-Link RDI License managment)" added.
4	060801	TQ	Updated list of supported flash devices.
3	060703	OO	Section "Reset strategy": Added listing of available reset types.
2	051111	TQ	Adding description of adaptive clocking. Minor corrections.
1	051028	OO	Initial version.

Software versions

Refers to Release.html for information about the changes of the software versions.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler)
- The C programming language
- The target processor
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend The C Programming Language by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C-programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This document describes J-Link RDI. It provides an overview over the major features of J-Link RDI, gives you some background information about Flash breakpoints and configuration in general and describes using RDI compliant debuggers with J-Link RDI. Finally, the chapter *Support* on page 69 helps to troubleshoot common problems.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command-prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
Reference	Reference to chapters, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections

Table 1.1: Typographic conventions

SEGGER Microcontroller GmbH & Co. KG develops and distributes software development tools and ANSI C software components (middleware) for embedded systems in several industries such as telecom, medical technology, consumer electronics, automotive industry and industrial automation.

SEGGER's intention is to cut software development-time for embedded applications by offering compact flexible and easy to use middleware, allowing developers to concentrate on their application.

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet efficent real-time kernel. emWin, written entirely in ANSI C, can easily be used on any CPU and most any display. It is complemented by the available PC tools: Bitmap Converter, Font Converter, Simulator and Viewer. embOS supports most 8/16/32-bit CPUs. Its small memory footprint makes it suitable for single-chip applications.

Apart from its main focus on software tools, SEGGER developes and produces programming tools for flash microcontrollers, as well as J-Link, a JTAG emulator to assist in development, debugging and production, which has rapidly become the industry standard for debug access to ARM cores.

Corporate Office:
http://www.segger.com

United States Office:
http://www.segger-us.com

EMBEDDED SOFTWARE (Middleware)

emWin
Graphics software and GUI

emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display. Starterkits, eval- and trial-versions are available.

embOS
Real Time Operating System

embOS is an RTOS designed to offer the benefits of a complete multitasking system for hard real time applications with minimal resources. The profiling PC tool embOSView is included.

emFile
File system

emFile is an embedded file system with FAT12, FAT16 and FAT32 support. emFile has been optimized for minimum memory consumption in RAM and ROM while maintaining high speed. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.

emUSB
USB device stack

A USB stack designed to work on any embedded system with a USB client controller. Bulk communication and most standard device classes are supported.

SEGGER TOOLS

Flasher
Flash programmer

Flash Programming tool primarily for microcontrollers.

J-Link
JTAG emulator for ARM cores

USB driven JTAG interface for ARM cores.

J-Trace
JTAG emulator with trace

USB driven JTAG interface for ARM cores with Trace memory. supporting the ARM ETM (Embedded Trace Macrocell).

J-Link / J-Trace Related Software

Add-on software to be used with SEGGER's industry standard JTAG emulator, this includes flash programming software and flash breakpoints.

Table of Contents

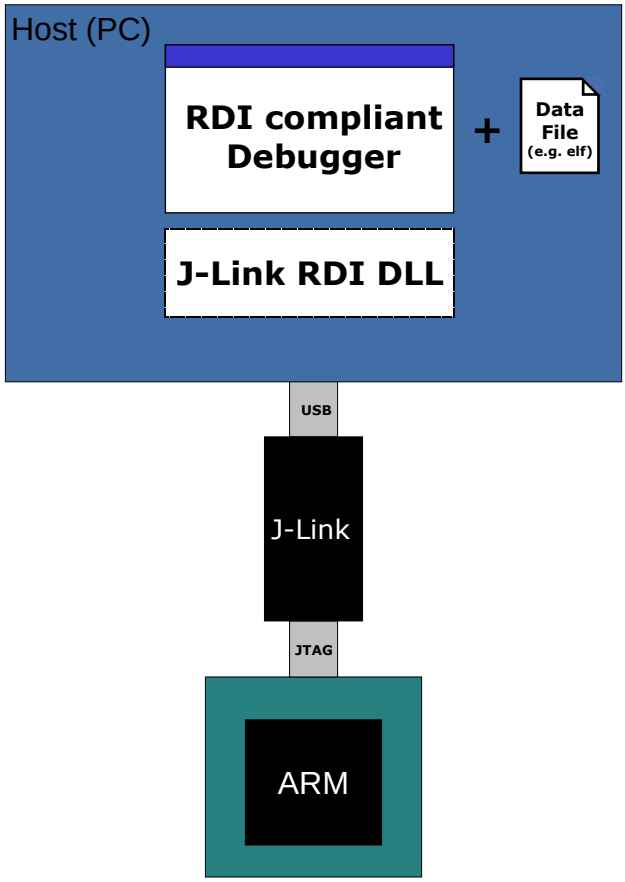
Chapter 1

Introduction

This chapter gives a short overview about the use of J-Link RDI, flash breakpoints and flash download.

1.1 What is RDI?

Remote Debug Interface (RDI) is an Application Programming Interface (API) that defines a standard set of data structures and functions that abstract hardware for debugging purposes. J-Link RDI mainly consists of a DLL designed for ARM cores to be used with any RDI compliant debugger. J-Link RDI offers features like flash breakpoints and flash download.



1.1.1 Features of J-Link RDI

- Usable with every RDI compliant debugger
- Supports more than 2 breakpoints when debugging in flash by using the flash breakpoints feature (add. license required)
- Offers download into flash without the need for a flash loader (add. license required)
- Instruction set simulation (improves debugging performance)
- Any core supported by J-Link ARM (ARM7 / ARM9)
- Easy to use

1.2 Requirements

Host System

In order to use J-Link RDI you need a host system running Windows 2000 or Windows XP with SEGGER's J-Link USB driver and a RDI compliant debugger.

Target System

An ARM7 or ARM9 target system is required. The system should have a 20-pin connector as defined by ARM Ltd. Please note that Segger offers an optional adapter to use J-Link ARM with targets using 14 pin 0.1" mating JTAG connectors.

1.3 Basic principles

The EmbeddedICE logic and the ARM processor debug extensions enable J-Link ARM to debug software running on an ARM processor. The EmbeddedICE is accessed via the JTAG port and described in detail in the technical reference manuals available from ARM. Some basic information can be found in the chapter *Background information* on page 59.

1.4 Purchase

The J-Link ARM software and documentation pack includes the J-Link RDI software. You can download the J-Link ARM software and documentation pack from:

http://www.segger.com/downloads.html

1.5 Licensing

The software is licensed on a per J-Link basis. It requires different licenses for different parts of the software. In general, the following items are required to use the software:

1. J-Link ARM
2. RDI license

In addidtion to that, "flash download" and "flash breakpoints" are not part of the standard RDI software and require add. licenses. Free trial licenses are avaliable upon request from www.segger.com.

Chapter 2

Using J-Link RDI with different debuggers

This chapter describes how to use J-Link ARM with different debuggers via RDI.

The J-Link RDI software is an ARM Remote Debug Interface (RDI) for J-Link ARM. It makes it possible to use J-Link ARM with any RDI compliant debugger. The package consists of 2 DLLs, which need to be copied to the same folder. In order to use these DLLs, they need to be selected in the debugger. J-Link RDI is a separate item and not included in the J-Link ARM software.

2.1 IAR Embedded Workbench IDE

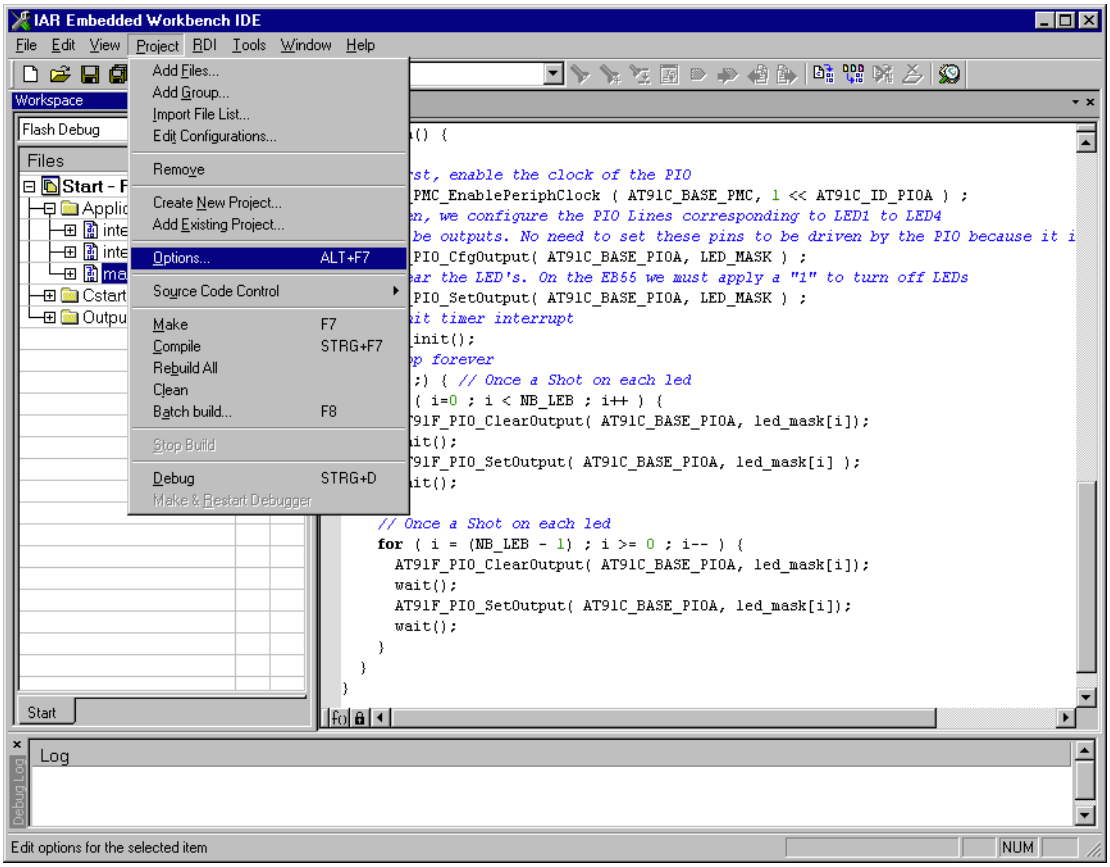
2.1.1 Software version

The `JLinkRDI.dll` has been tested with IAR Embedded Workbench IDE version 4.40. There should be no problems with other versions of IAR Embedded Workbench IDE.

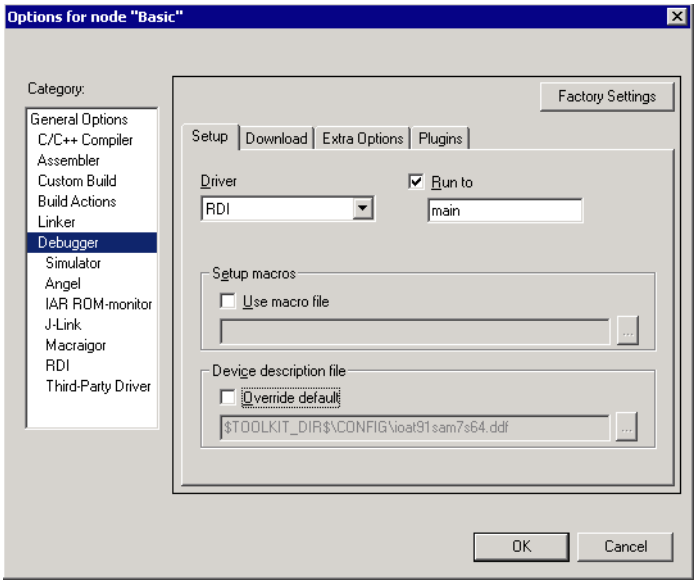
All screenshots are taken from IAR Embedded Workbench version 4.40.

2.1.2 Configuring to use J-Link RDI

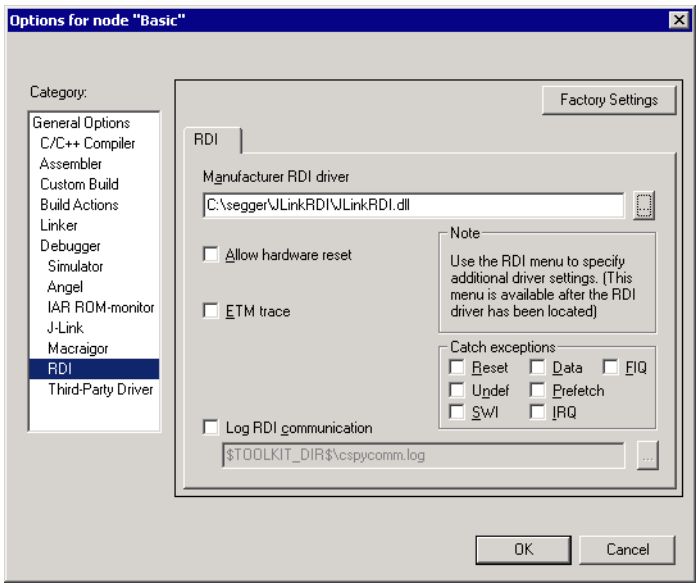
1. Start the IAR Embedded Workbench and open the tutor example project or your desired project. This tutor project has been preconfigured to use the simulator driver. In order to run the J-Link RDI you must change the driver.



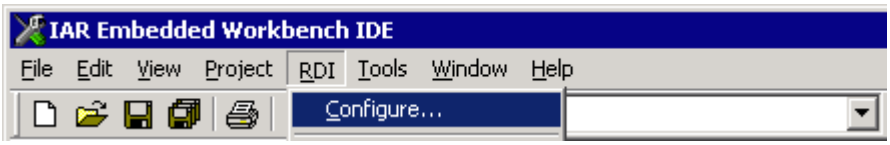
2. Choose **Project** | **Options** and select the **Debugger** category. Change the **Driver** option to RDI.



3. Go to the RDI page of the Debugger options, select the manufacturer driver (`JLinkRDI.dll`) and click **OK**.



4. Now an extra menu, RDI, has been added to the menu bar.
Choose **RDI** | **Configure** to configure the J-Link. For details refer to the configuration chapter.



J-Link may also be selected directly in the debugger of the IAR Embedded Workbench IDE; RDI can be used, but is not necessary to use the IAR Embedded Workbench IDE with J-Link ARM unless you want to use one of the features offered by J-Link RDI (e.g. flash breakpoints or flash download).

2.1.3 Limitations

Their are no known limitations. All features including download into flash (add. license) and breakpoints in flash memory (add. license) can be used.

2.2 ARM's AXD (ARM Developer Suite, ADS)

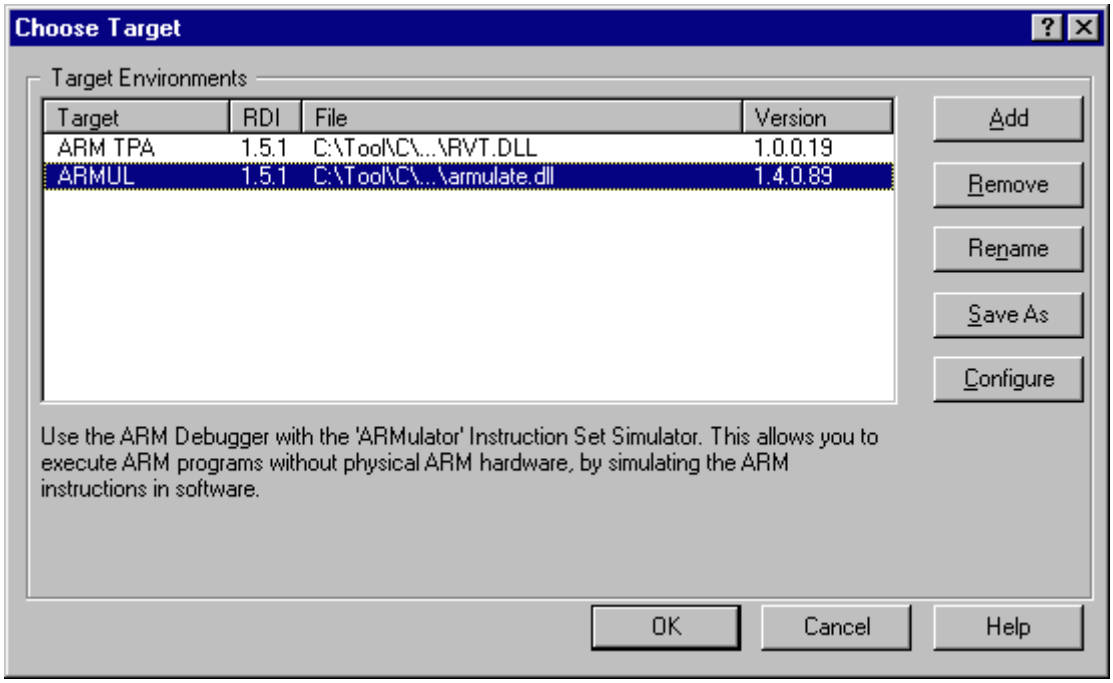
2.2.1 Software version

The `JLinkRDI.dll` has been tested with ARM's AXD version 1.2.0 and 1.2.1. There should be no problems with other versions of ARM's AXD.

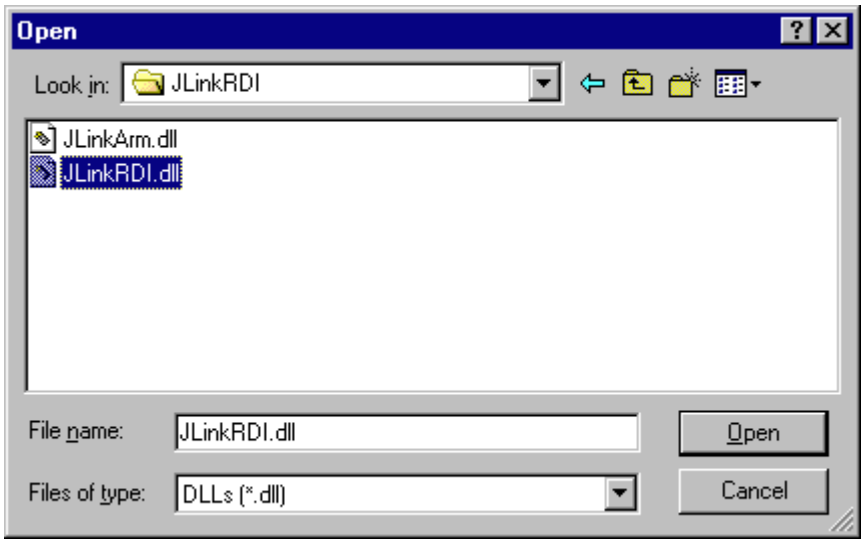
All screenshots are taken from ARM's AXD version 1.2.0.

2.2.2 Configuring to use J-Link RDI

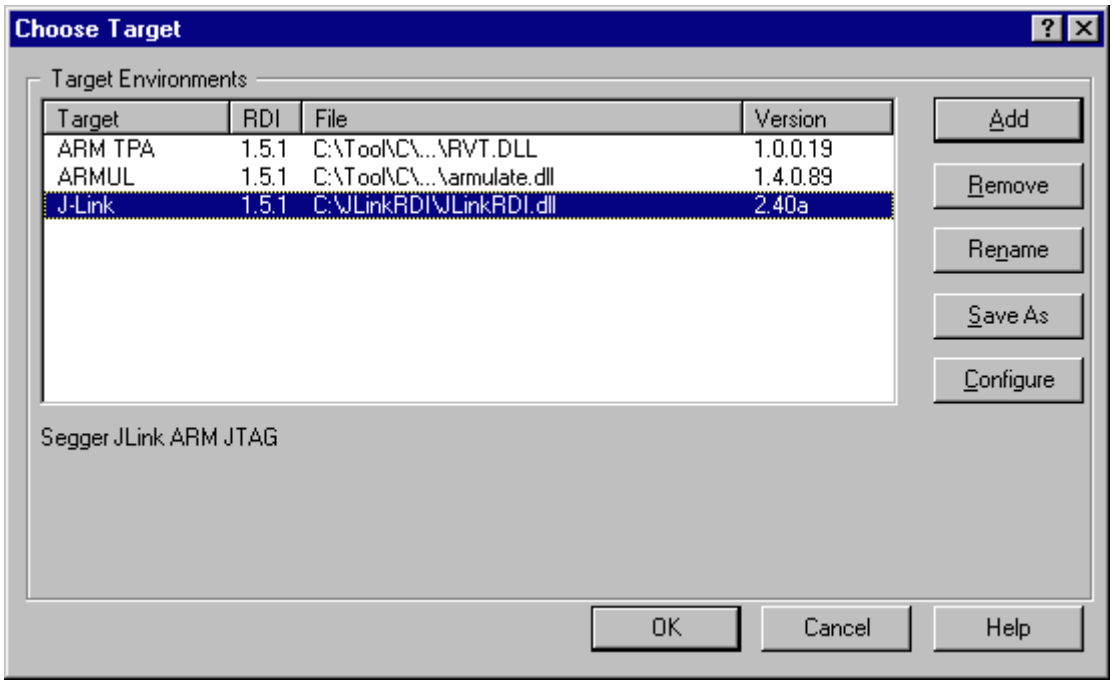
1. Start the ARM debugger and select **Options** | **Configure Target...**. This opens the **Choose Target** dialog box:



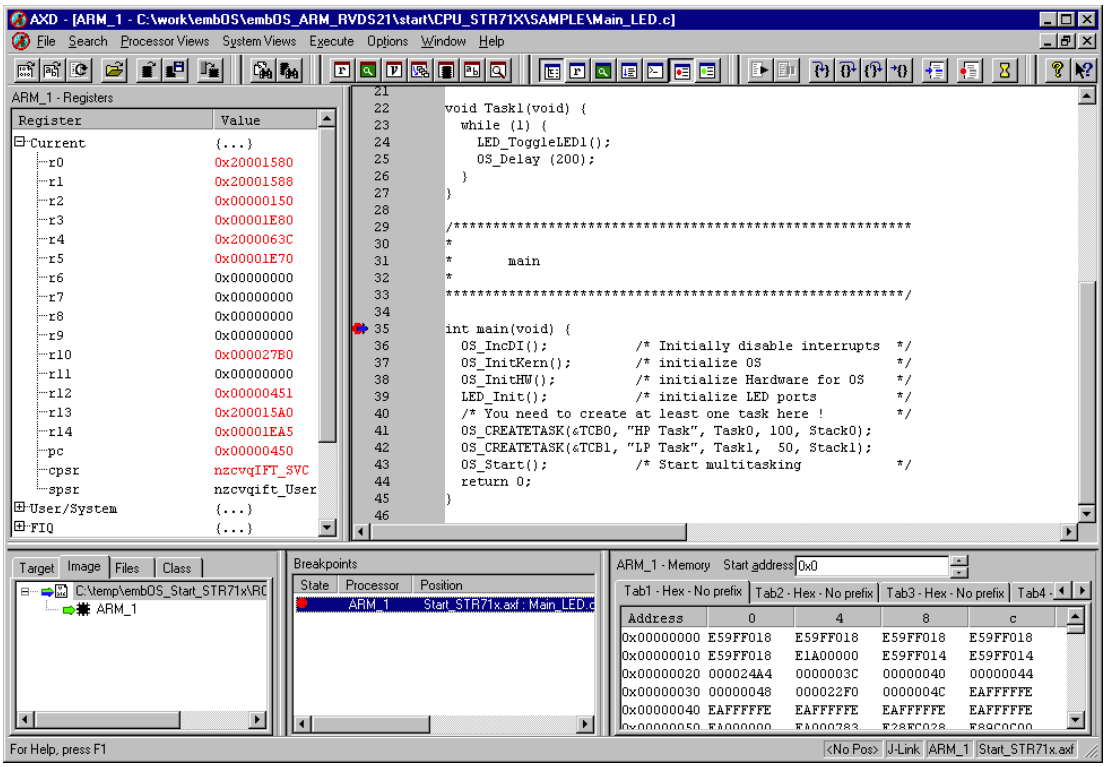
2. Press the **Add** Button to add the `JLinkRDI.dll`.



3. Now the J-Link RDI is in the **Target Environments** list.



4. Select J-Link and press **OK** to connect to the target via J-Link ARM. To configure J-Link RDI refer to the chapter *Configuration* on page 31. After downloading an image to the target board, the debugger window looks as follows:



2.2.3 Limitations

Their are no known limitations. All features including download into flash (add. license) and breakpoints in flash memory (add. license) can be used.

2.3 ARM's RVDS (RealView developer suite)

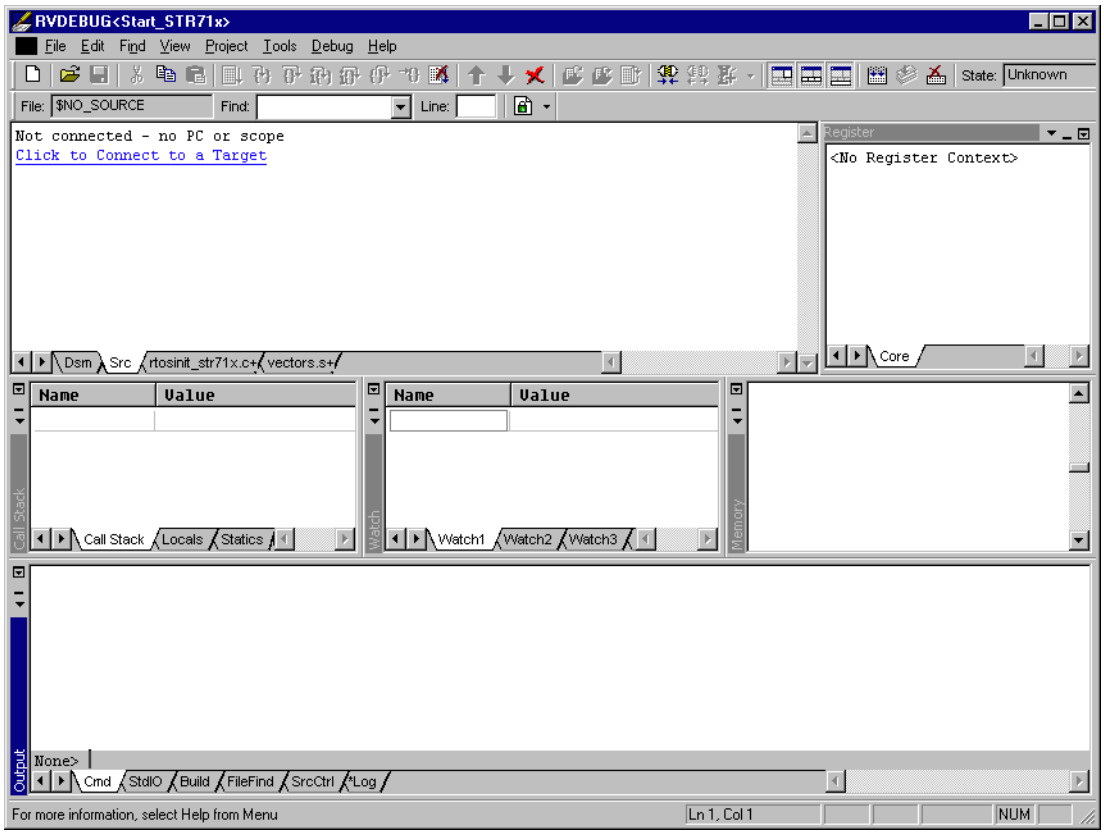
2.3.1 Software version

The `JLinkRDI.dll` has been tested with ARM's RVDS version 2.1 and 3.0. There should be no problems with earlier versions of RVDS (up to version v3.0.1). RVDS version 3.1 does not longer support RDI protocol to communicate with the debugger.

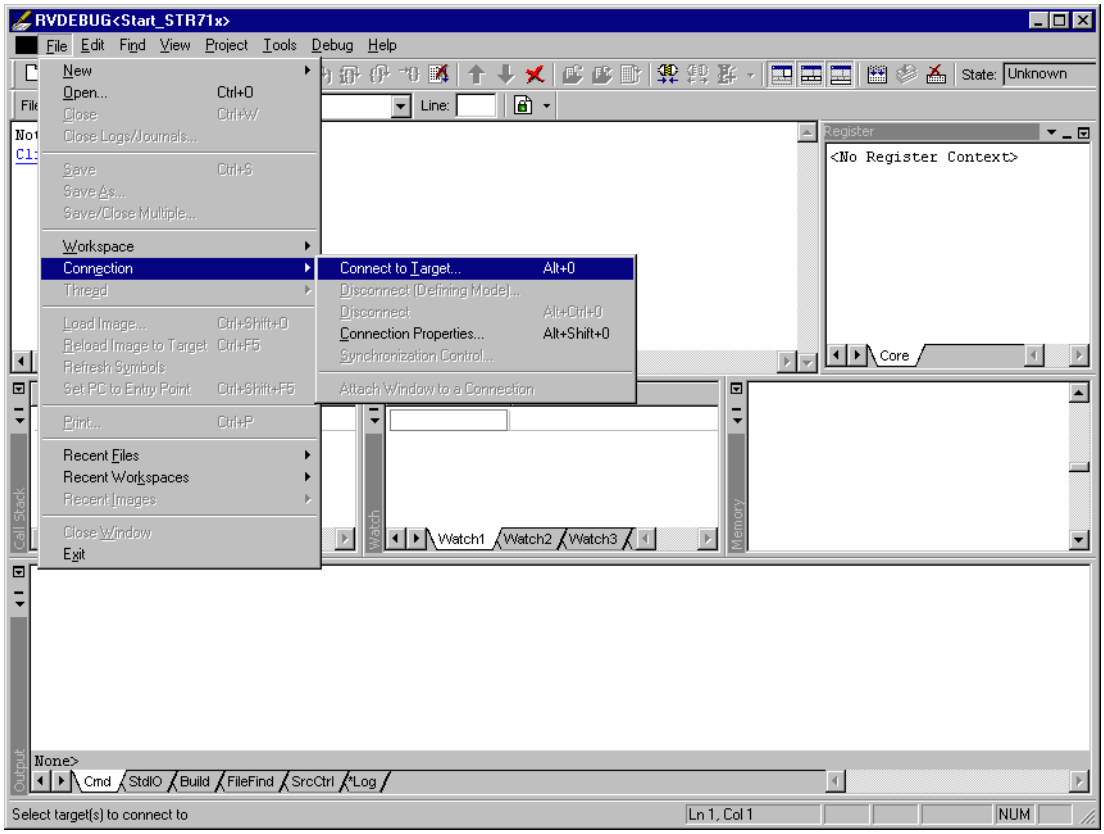
All screenshots are taken from ARM's RVDS version 2.1.

2.3.2 Configuring to use J-Link RDI

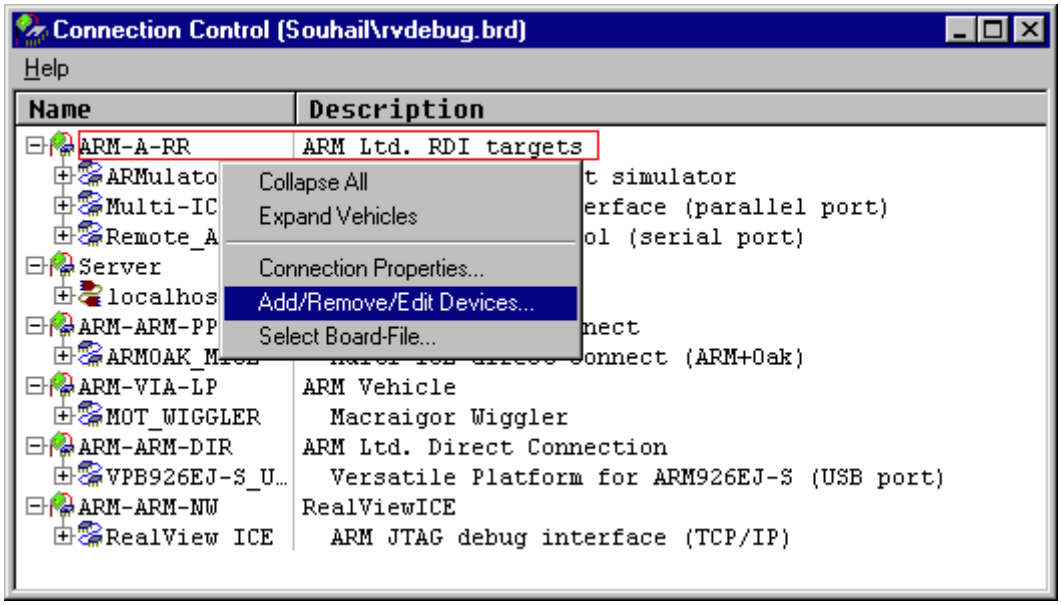
1. Start the Real View debugger:



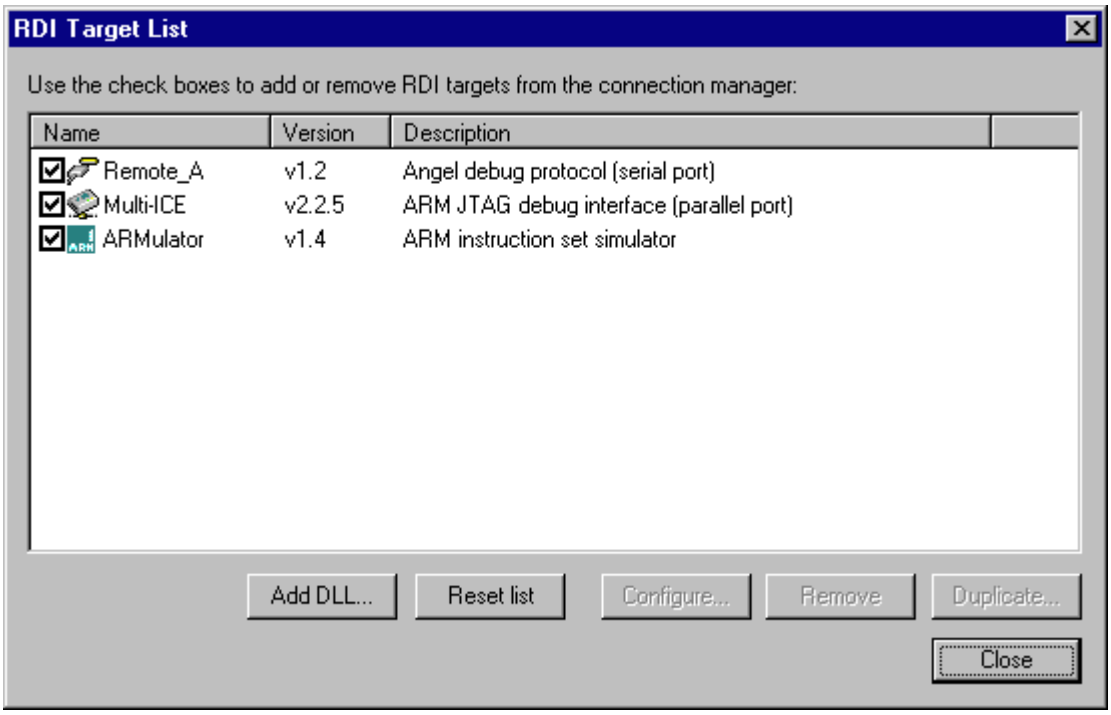
2. Select **File** | **Connection** | **Connect to Target**.



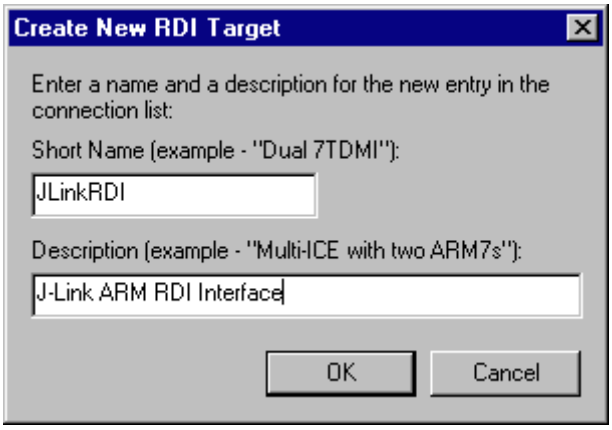
3. In the **Connection Control** dialog use the right mouse click on the first item and select **Add/Remove/Edit Devices**



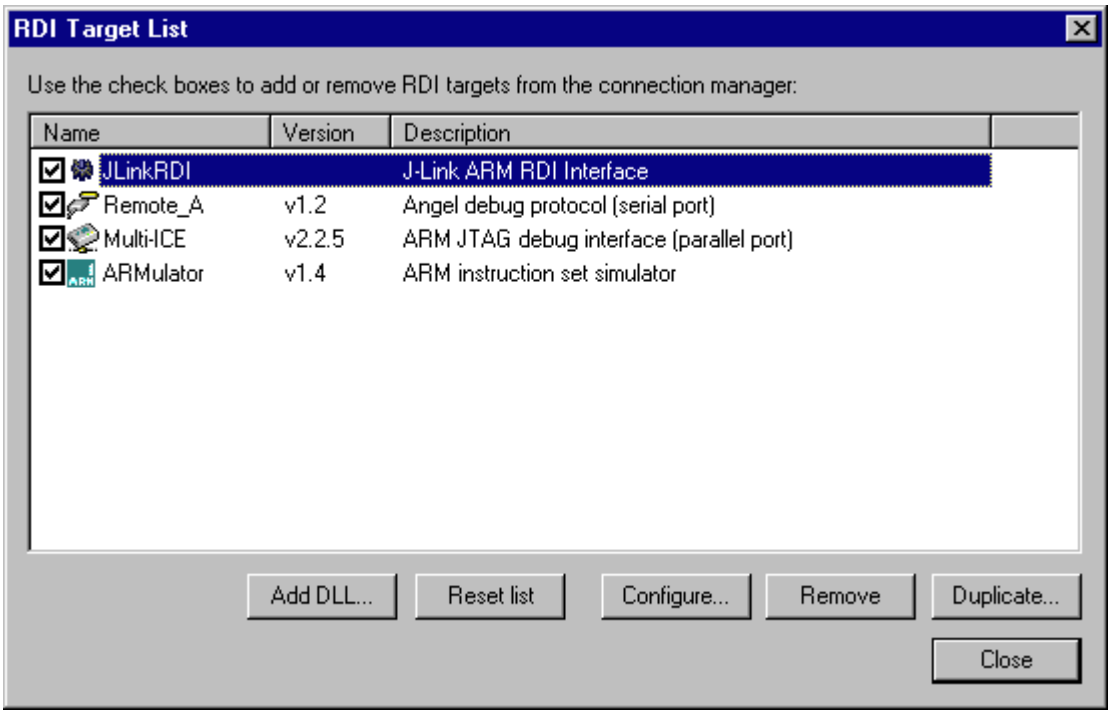
4. Now select **Add DLL** to add the `JLinkRDI.dll`. Select the installation path of the software, for example:
`C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll`



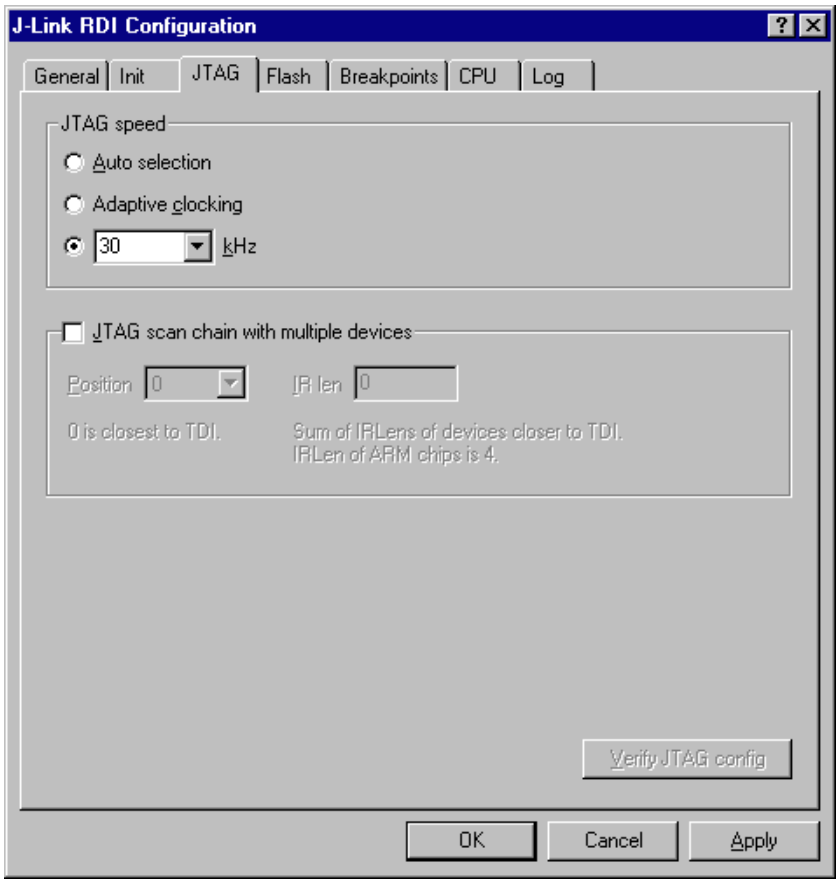
5. After adding the DLL, an additional Dialog opens and asks for description: (These values are voluntary, if you do not want change them, just click **OK**) Use the following values and click on **OK**, **Short Name:** `JLinkRDI` **Description:** `J-Link ARM RDI Interface`.



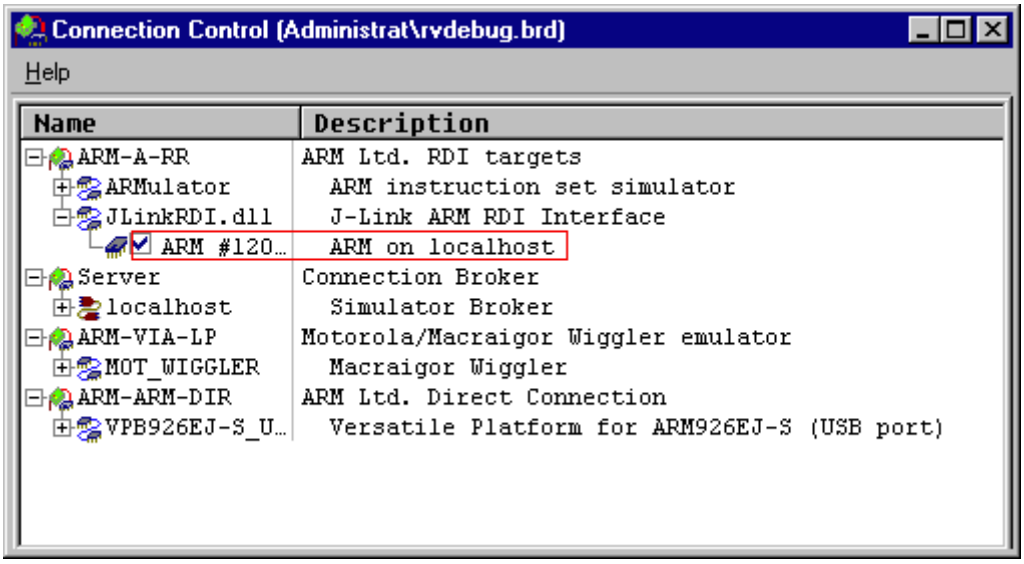
6. Back in the **RDI Target List** Dialog
Select **JLink-RDI** and click **Configure**. For configuration details refer to chapter *Configuration* on page 31.



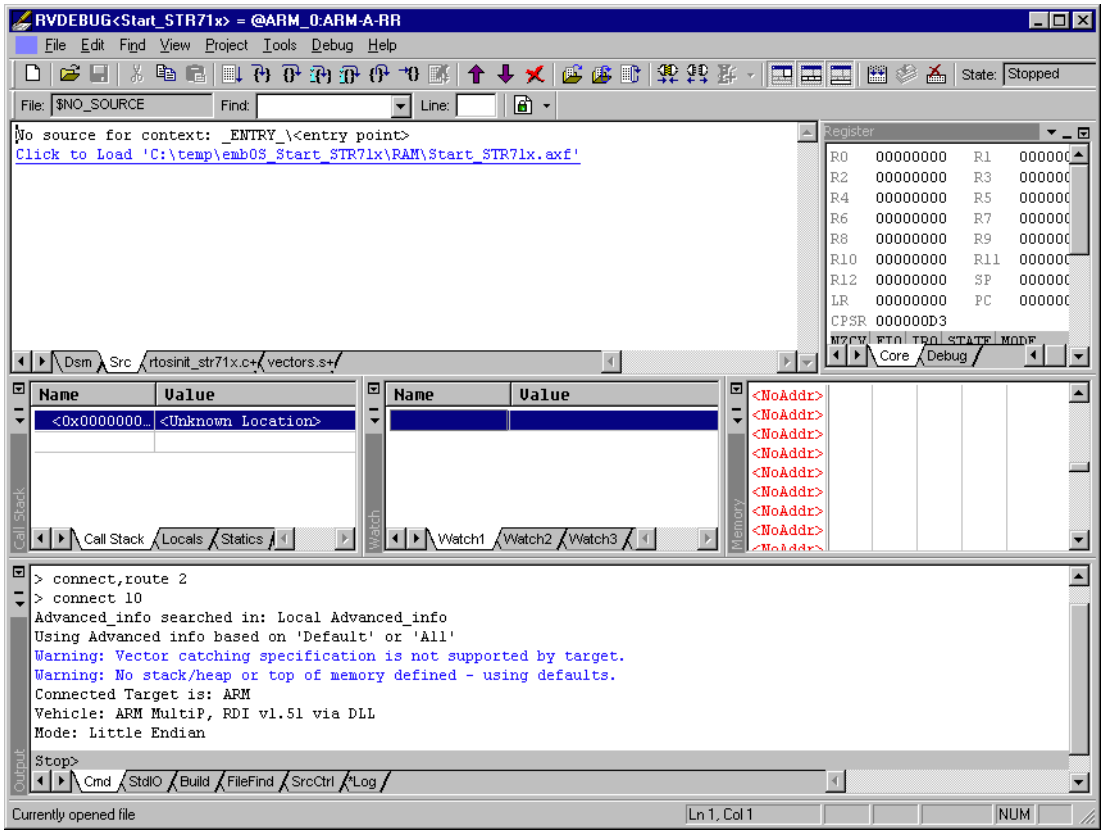
7. Click the **OK** button in the configuration dialog. Now close the **RDI Target List** dialog. Be sure your target hardware is already connected to J-Link.



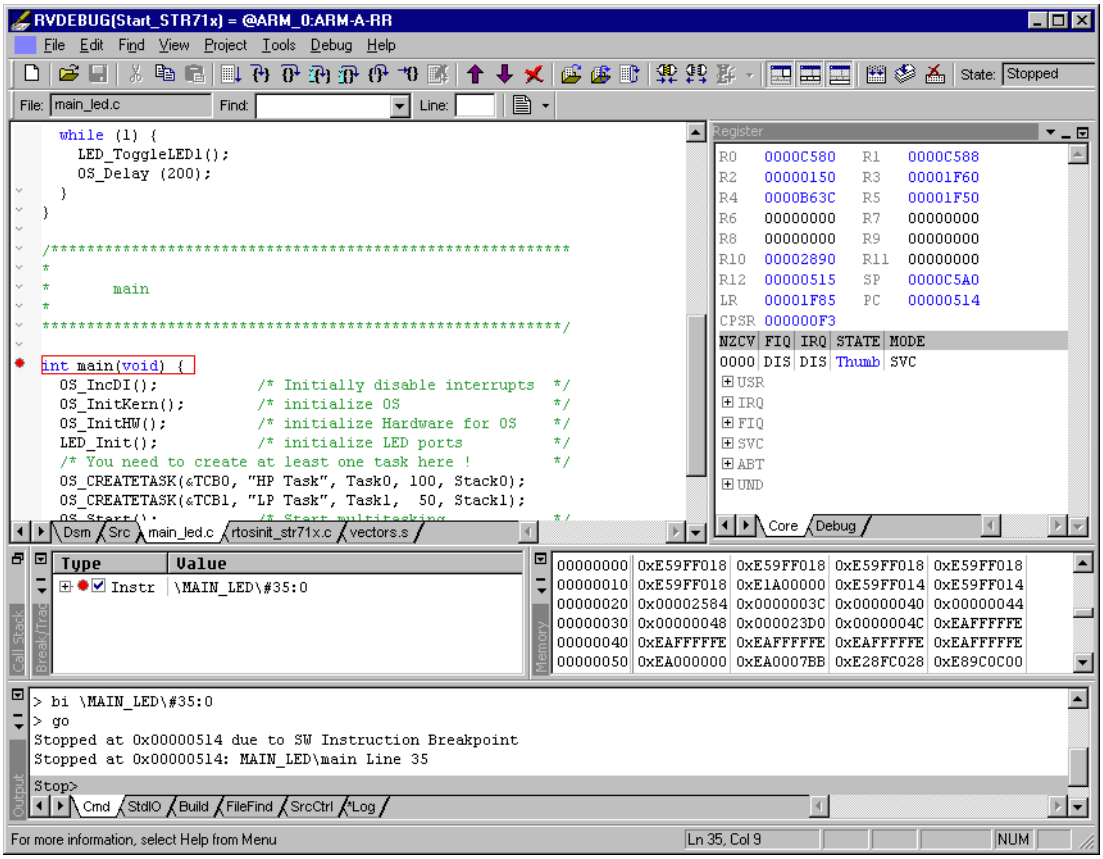
8. In the **Connection control** dialog, expand the **JLink ARM RDI Interface** and select the ARM_0 Processor. Close the **Connection Control** Window.



9. Now the RealView Debugger is connected to J-Link.



10. A project or an image is needed for debugging. After downloading, J-Link is used to debug the target.



2.3.3 Limitations

Their are no known limitations. All features including download into flash (add. license) and breakpoints in flash memory (add. license) can be used.

2.4 GHS MULTI

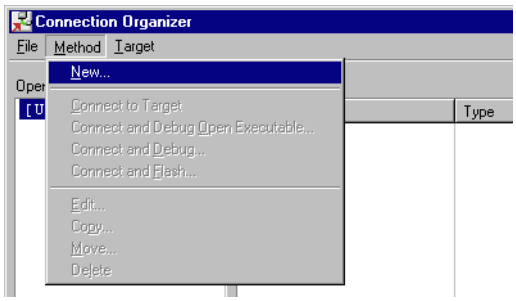
2.4.1 Software version

The `JLinkRDI.dll` has been tested with GHS MULTI version 4.07. There should be no problems with other versions of GHS MULTI.

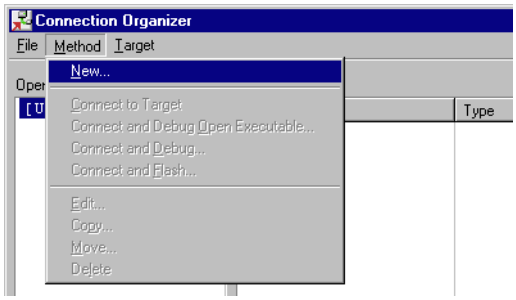
All screenshots are taken from GHS MULTI version 4.07.

2.4.2 Configuring to use J-Link RDI

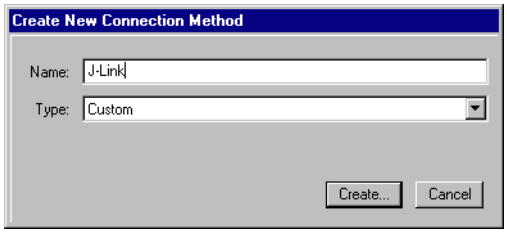
1. Start Green Hills Software MULTI integrated development environment. Click **Connect** | **Connection Organizer** to open the **Connection Organizer**.



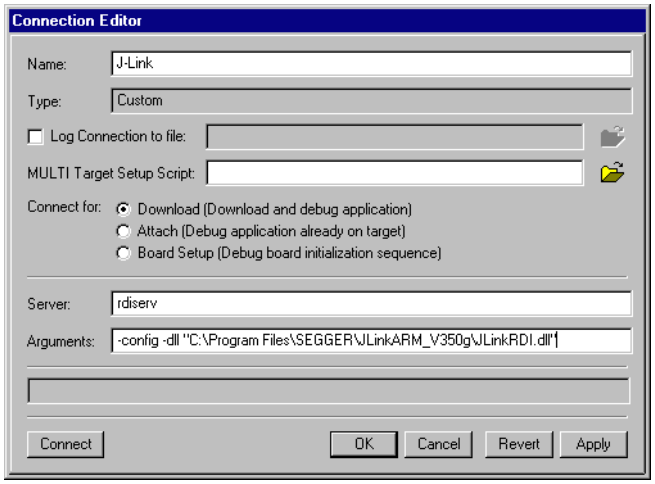
2. Click **Method** | **New** in the **Connection Organizer** dialog.



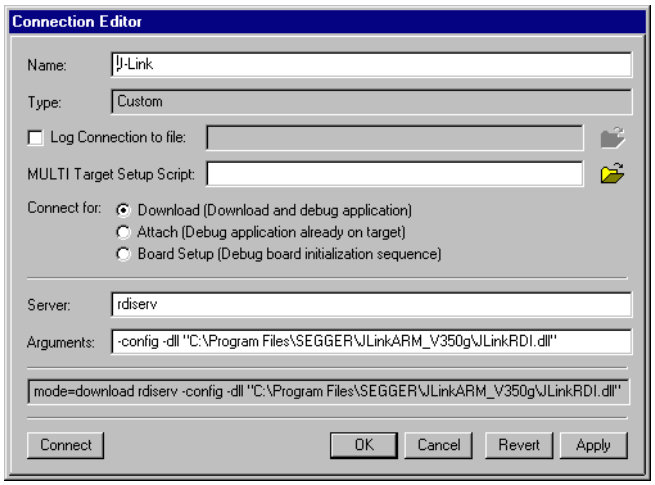
3. The **Create a new Connection Method** will be opened. Enter a name for your configuration in the **Name** field and select **Custom** in the **Type** list. Confirm your choice with the **Create...** button.



4. The **Connection Editor** dialog will be opened. Enter **rdiserv** in the **Server** field and enter the following values in the **Arguments** field:
   `-config -dll <FullPathToJLinkDLLs>`
   Note that `JLinkRDI.dll` and `JLinkARM.dll` must be stored in the same directory. If you have used the standard J-Link installation path or another path that includes spaces, enclose the path in quotation marks.
   Example:
   `-config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll"`
   Refer to GHS manual "MULTI: Configuring Connections for ARM Targets", chapter "ARM Remote Debug Interface (rdiserv) Connections" for the complete list of possible arguments.

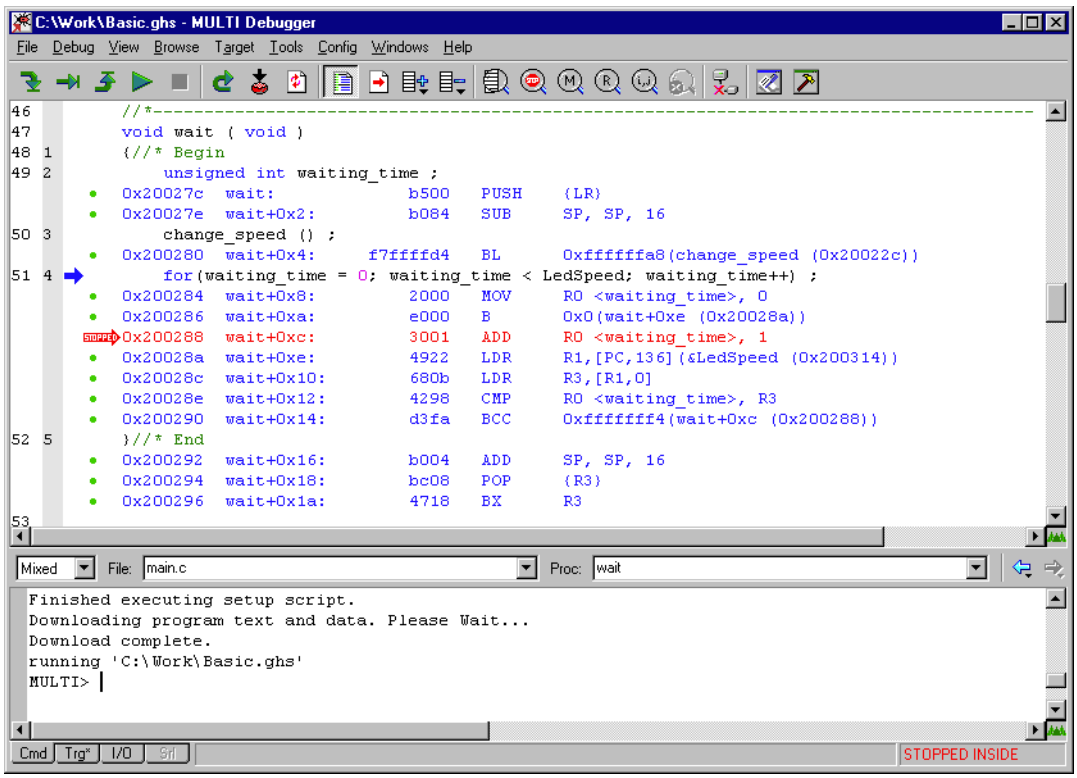


5. Confirm your choices by clicking the **Apply** button and click afterwards the **Connect** button.



6. The **J-Link RDI Configuration** dialog will be opened. J-Link RDI requires a valid license. If you do not have entered your J-Link RDI license, click **License** and add your license with the **J-Link RDI License management**. Refer to chapter *Configuration* on page 31 for further information about the options of the **J-Link RDI Configuration** dialog and the usage of the **License Manager**.

7. Click the **OK** button to connect to your target.
8. Build your project and start the debugger. Note that you have to perform at least one action (for example **step** or **run**) to initiate the download of your application to the target.



2.4.3 Limitations

Their are no known limitations. All features including download into flash (add. license) and breakpoints in flash memory (add. license) can be used.

2.5 KEIL µVision IDE

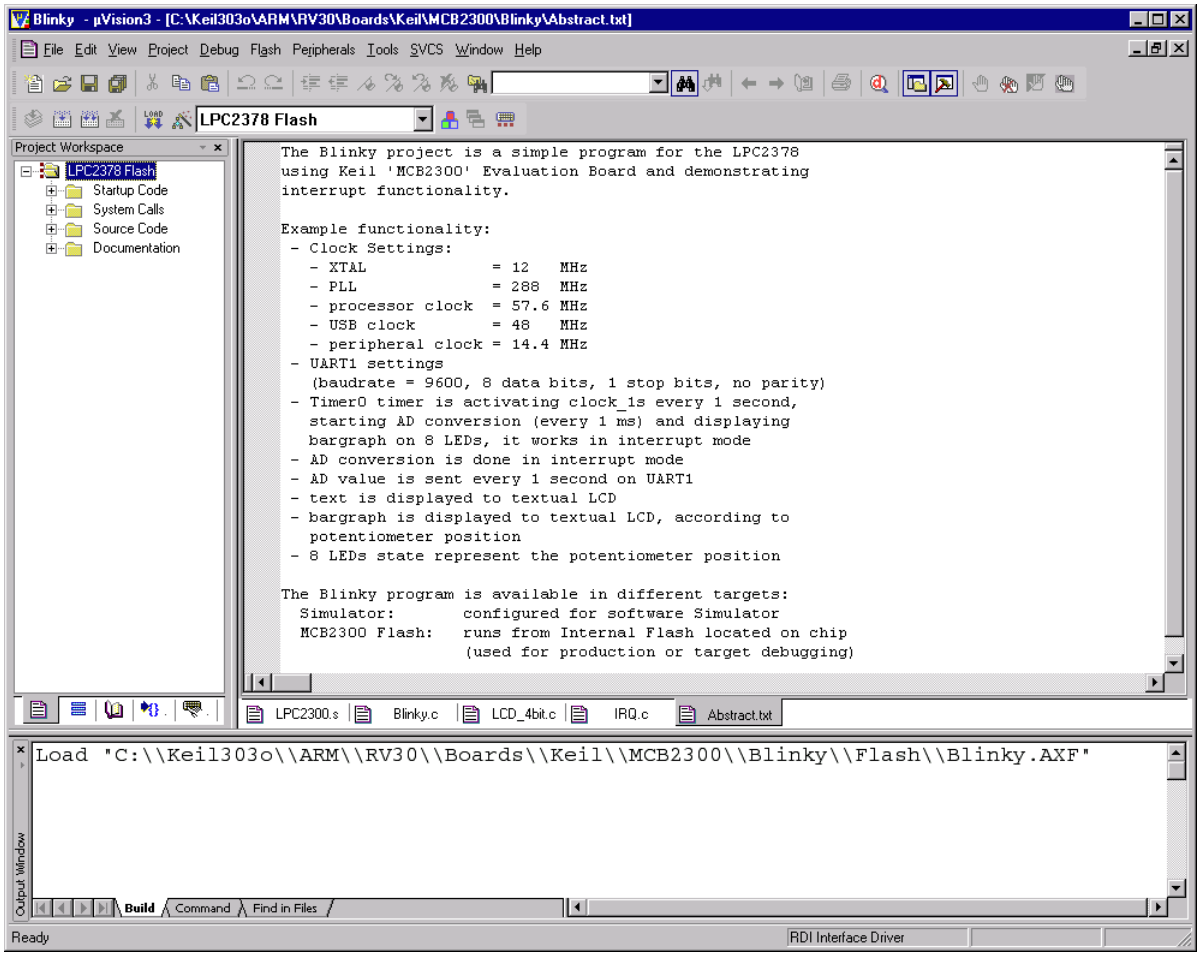
2.5.1 Software version

The `JLinkRDI.dll` has been tested with KEIL µVision3 IDE version 3.34. There should be no problems with other versions of KEIL µVision.

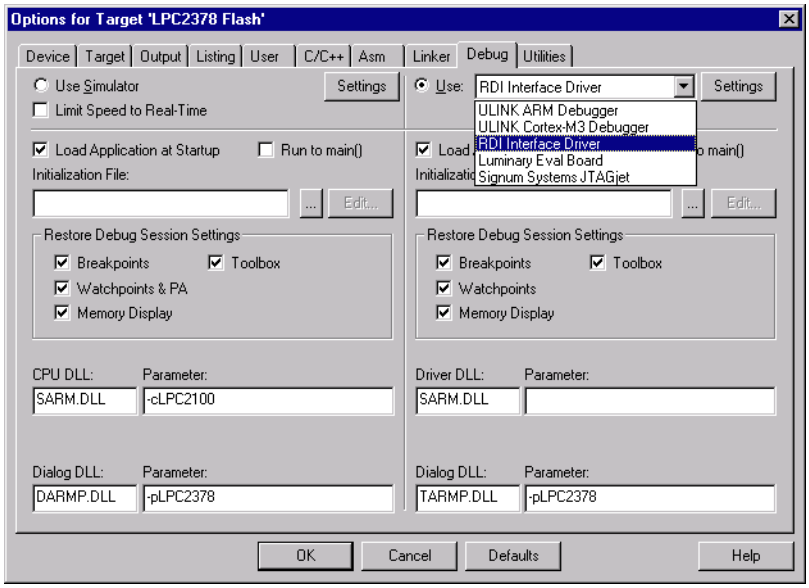
All screenshots are taken from KEIL µVision3 version 3.34.

2.5.2 Configuring to use J-Link RDI

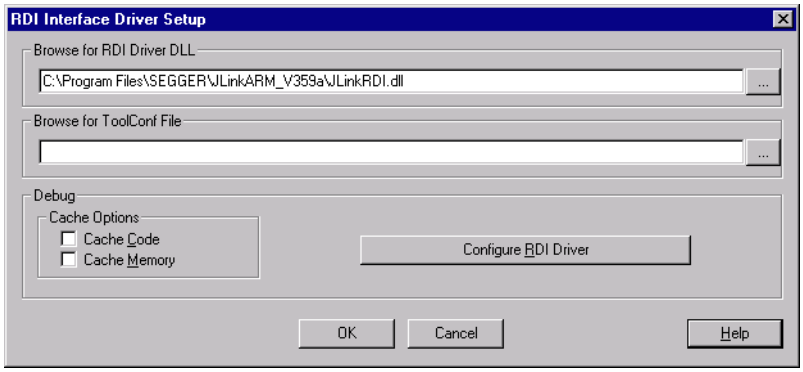
Start KEIL uVision and open your project.



Select **Project** | **Options for Target** '<NameOfYourTarget>' to open the project options dialog and select the **Debug** tab.

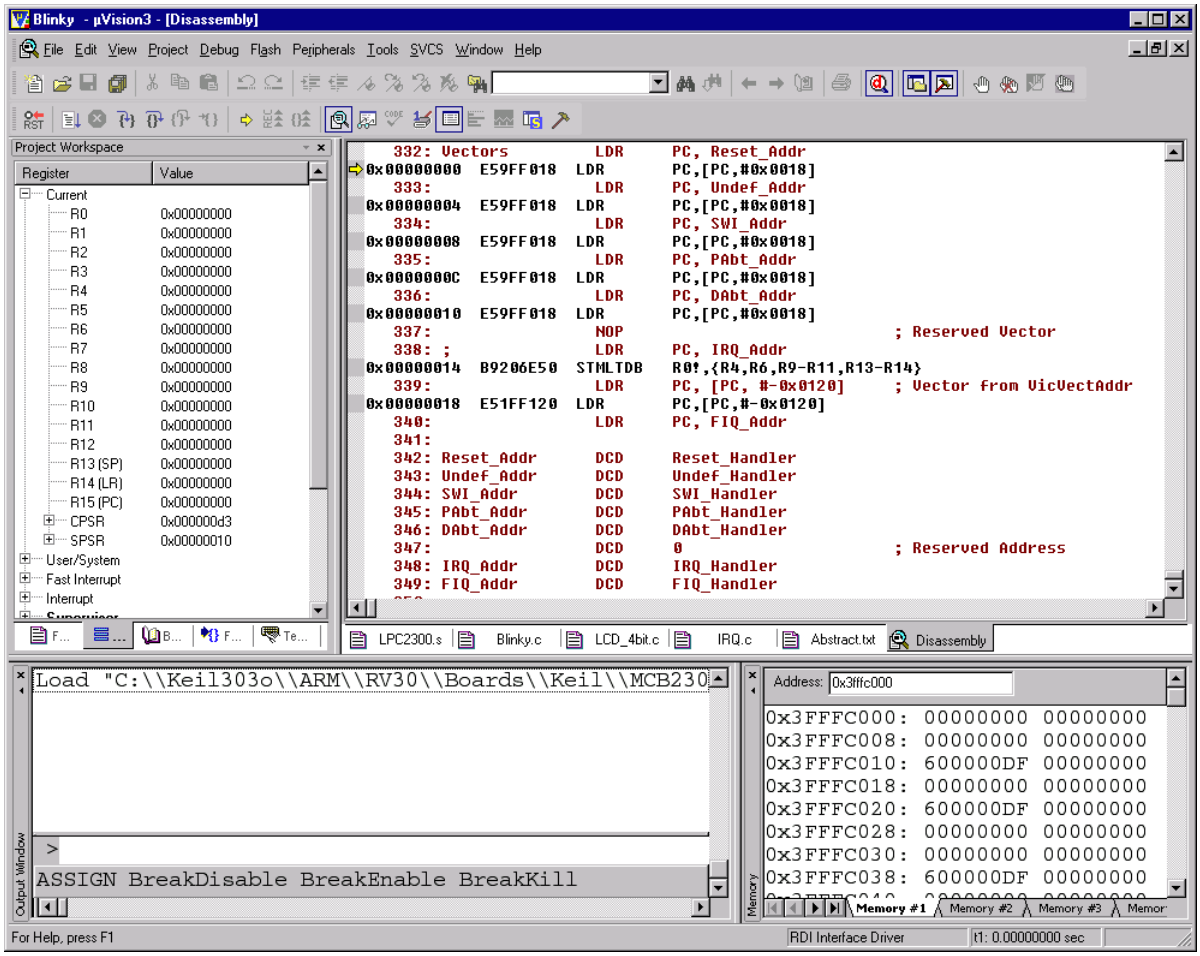


Choose **RDI Interface Driver** from the list as shown above and click the **Settings** button. Select the location of `JLinkRDI.dll` in **Browse for RDI Driver DLL** field. and click the **Configure RDI Driver** button.



The J-Link RDI Configuration dialog will be opened. For detailed information about the confiiguration of J-Link RDI, refer to chapter *Configuration* on page 31.

After finishing configuration, you can build your project (**Project** | **Build Target**) and start the debugger (**Debug** | **Start/Stop debug session**).

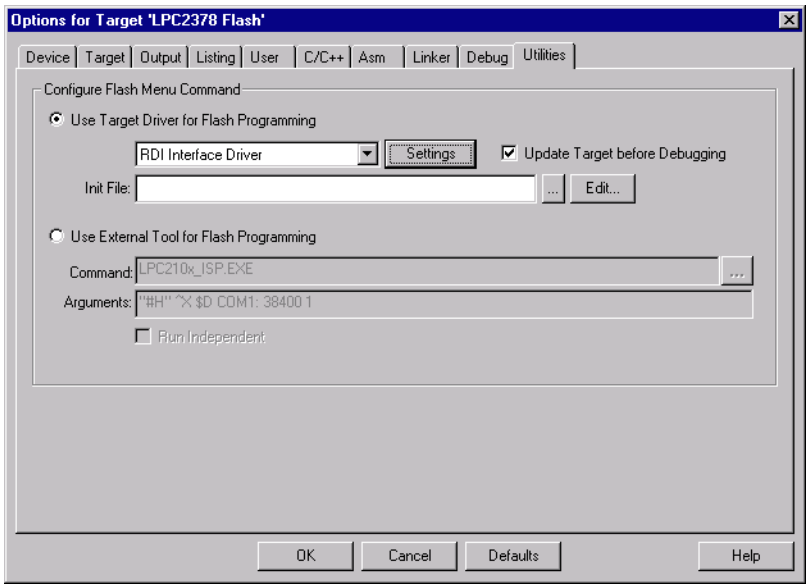


Configuring flash download via J-Link RDI

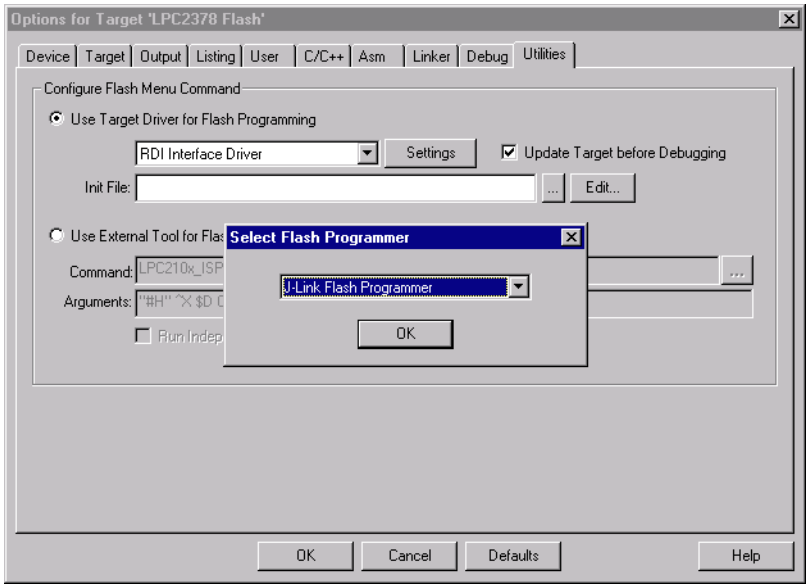
You can use the J-Link RDI flash download feature instead of the standard µVision-flash loader.

Note: This feature requires an additional licence. A free trial license is avaliable upon request from *www.segger.com*.

If you have the required licence and want to use J-Link RDI also for flash download, select **Flash** | **Configure Flash Tools** and choose **RDI Interface Driver** from the list in the **Configure Flash Menu Command** as shown below.



Click the **Settings** button and select **J-Link Flash Programmer** in the **Select Flash Programmer** dialog. Confirm your choice with a click on the **OK** button.



Refer to subchapter *Flash configuration* on page 38 for detailed information about the configuration of the flash programming feature.

2.5.3 Limitations

Their are no known limitations. All features including download into flash (add. license) and breakpoints in flash memory (add. license) can be used.

Chapter 3

Configuration

This chapter describes how to confgure J-Link RDI.

3.1 Overview

This chapter provides a short overview about the configuration abilities of J-Link RDI. Normally, the default settings can be used.

3.1.1 Configuration file JLinkRDI.ini

All settings are stored in the file `JLinkRDI.ini`. This file is located in the same directory as `JLinkRDI.dll`.

3.1.2 Using different configurations

It can be desirable to use different configurations for different targets. If you intent to do this, you should create a new folder and copy the `JLinkARM.dll` and the `JLinkRDI.dll` into it. You can now have project A which uses the DLLs in the original folder and project B which uses the DLLs in the newly created directory. Both projects will use separate configuration files, stored in the same directory as the DLLs they are using.

If your debugger allows using a project-relative path (such as IAR's EWARM: Use for example `$PROJ_DIR$\RDI\`), it can make sense to create the directory for the DLLs and configuration file in a subdirectory of the project.

3.1.3 Using mutliple J-Links simulatenously

This procedure can also be used to operate 2 J-Links with different settings on the same host at the same time.

3.2 Configuration dialog

The configuration dialog consists of several tabs making the configuration of J-Link RDI an easy step.

3.2.1 General

3.2.1.1 Connection to J-Link

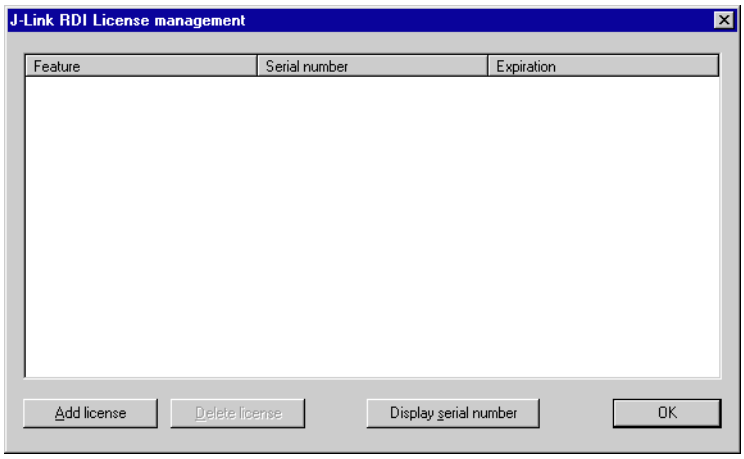
This setting allows to configure if J-Link ARM is connected locally via USB or is connected on a remote system and should be accessed by a given network address.

3.2.1.2 About

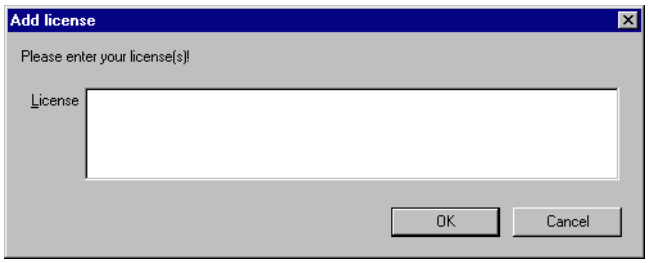
Opens the "About" window.

3.2.1.3 License (J-Link RDI License managment)

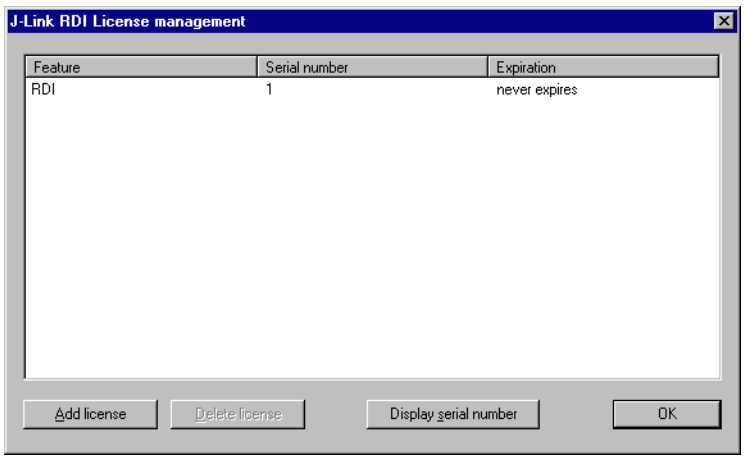
1. The **License** button opens the **J-Link RDI License management** dialog. J-Link RDI requires a valid license.



2. Click the **Add license** button and enter your license. Confirm your input by clicking the **OK** button.



3. The J-Link RDI license is now added.



3.2.2 Init

3.2.2.1 Macro file

A macro file can be specified to load custom settings to configure J-Link RDI with advanced commands for special chips or operations. For example a macro file can be used to initialize a target system in just about any way required.

3.2.3 Comands in the macro file

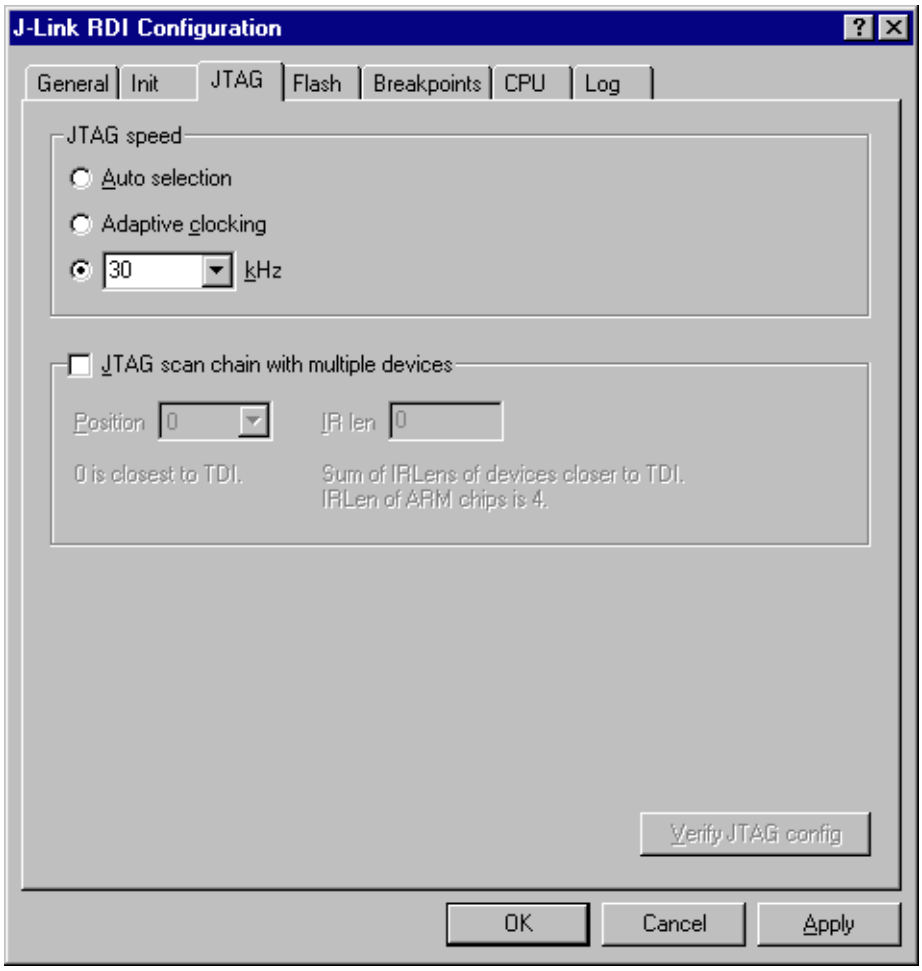
Command	Description
SetJTAGSpeed(x);	Sets the JTAG speed, x = speed in kHz (0=Auto)
Delay(x);	Waits a given time, x = delay in milliseconds
Reset(x);	Resets the target, x = delay in milliseconds
Go();	Starts the ARM core
Halt();	Halts the ARM core
Read8(Addr);	Reads a 8/16/32 bit value, Addr = address to read (as hex value)
Read16(Addr);	
Read32(Addr);	
Verify8(Addr, Data);	Verifies a 8/16/32 bit value, Addr = address to verify (as hex value) Data = data to verify (as hex value)
Verify16(Addr, Data);	
Verify32(Addr, Data);	
Write8(Addr, Data);	Writes a 8/16/32 bit value, Addr = address to write (as hex value) Data = data to write (as hex value)
Write16(Addr, Data);	
Write32(Addr, Data);	
WriteVerify8(Addr, Data);	Writes and verifies a 8/16/32 bit value, Addr = address to write (as hex value) Data = data to write (as hex value)
WriteVerify16(Addr, Data);	
WriteVerify32(Addr, Data);	
WriteRegister(Reg, Data);	Writes a register
WriteJTAG_IR(Cmd);	Writes the JTAG instruction register
WriteJTAG_DR(nBits, Data);	Writes the JTAG data register

Table 3.1: Macro file commands

3.2.4 Example of macro file

```
/*********************************************************************
*
*    Macro file for J-LINK RDI
*
*********************************************************************
* File:    LPC2294.setup
* Purpose: Setup for Philips LPC2294 chip
*********************************************************************
*/
SetJTAGSpeed(1000);
Reset(0);
Write32(0xE01FC040, 0x00000001); // Map User Flash into Vector area at (0-3f)
Write32(0xFFE00000, 0x20003CE3); // Setup CS0
Write32(0xE002C014, 0x0E6001E4); // Setup  PINSEL2 Register
SetJTAGSpeed(2000);
```


3.2.5 JTAG



3.2.5.1 JTAG speed

This allows the selection of the JTAG speed. There are basically three types of speed settings (which are explained below):

- Fixed JTAG speed
- Automatic JTAG speed
- Adaptive clocking

Fixed JTAG speed

The target is clocked at a fixed clock speed. The maximum JTAG speed the target can handle depends on the target itself. In general ARM cores without JTAG synchronization logic (such as ARM7-TDMI) can handle JTAG speeds up to the CPU speed, ARM cores with JTAG synchronization logic (such as ARM7-TDMI-S, ARM946E-S, ARM966EJ-S) can handle JTAG speeds up to 1/6 of the CPU speed. JTAG speeds of more than 10 MHz are not recommended.

Automatic JTAG speed

Selects automatically the maximum JTAG speed handled by the TAP controller.

Note: On ARM cores without synchronization logic, this may not work reliably, since the CPU core may be clocked slower than the maximum JTAG speed.

Adaptive clocking

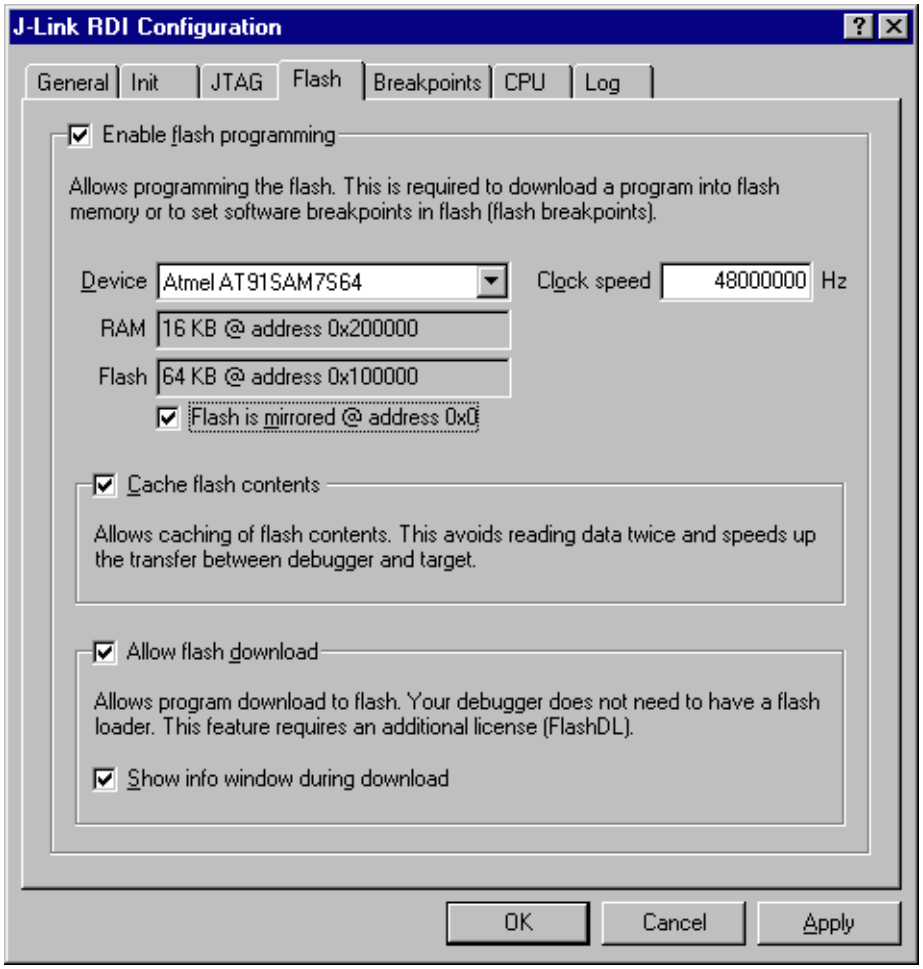
If the target provides the RTCK signal, select the adaptive clocking function to synchronize the clock to the processor clock outside the core. This ensures there are no synchronization problems over the JTAG interface.

Note: If you use the adaptive clocking feature, transmission delays, gate delays, and synchronization requirements result in a lower maximum clock frequency than with non-adaptive clocking. Do not use adaptive clocking unless it is required by the hardware design.

3.2.5.2 JTAG scan chain with multiple devices

The JTAG scan chain allows to specify the instruction register organization of the target system. This may be needed if there are more devices located on the target system than the ARM chip you want to access or if more than one target system is connected to one J-Link ARM at once.

3.2.6 Flash configuration



3.2.6.1 Enable flash programming

This checkbox enables flash programming. Flash programming is needed to use either flash download or to use flash breakpoints.

If flash programming is enabled you must select the correct flash memory and flash base address. Furthermore it is necessary for some chips to enter the correct CPU clock frequence.

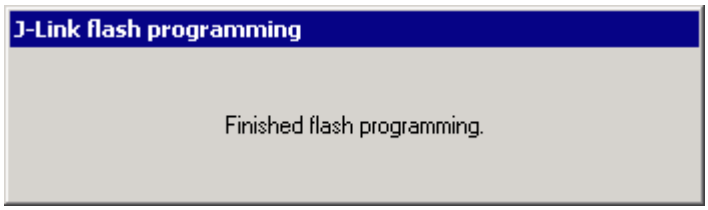
3.2.6.2 Cache flash contents

If enabled, the flash contents is cached by the J-Link RDI software to avoid reading data twice and to speed up the transfer between debugger and target.

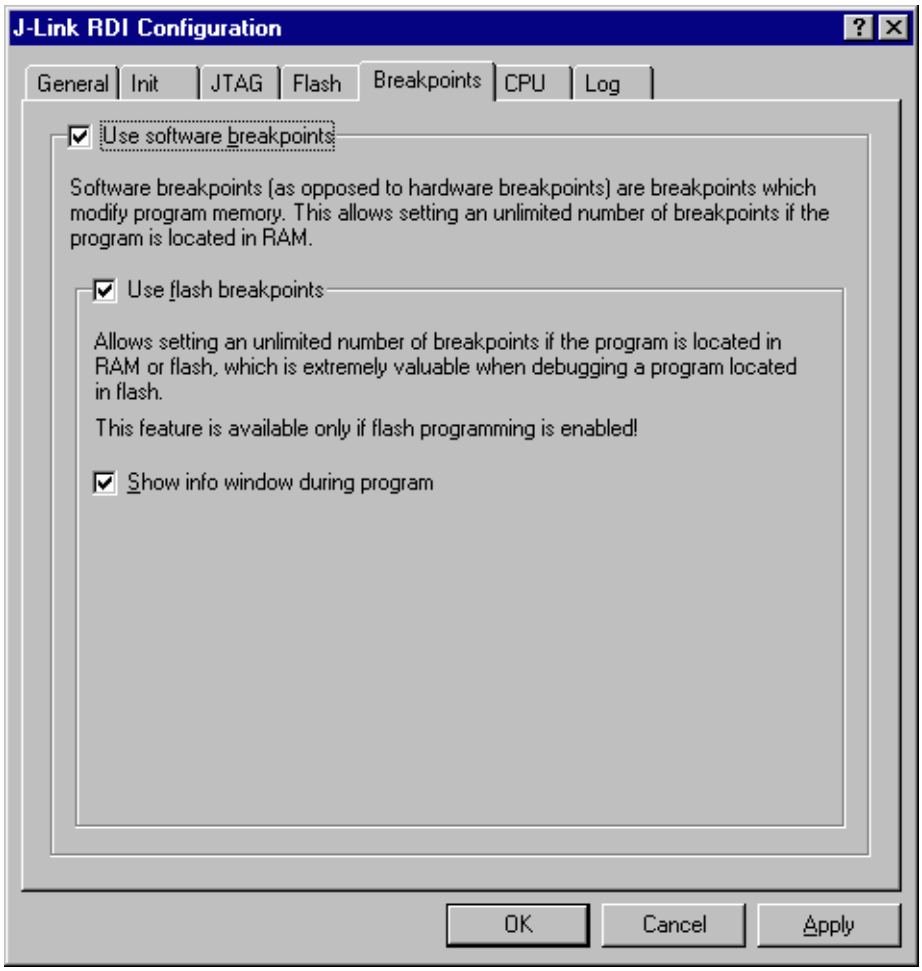
3.2.6.3 Allow flash download

This allows the J-Link RDI software to download program into flash. A small piece of code will be downloaded and executed in the target RAM which then programs the flash memory. This provides flash loading abilities even for debuggers without a build-in flash loader.

An info window can be shown during download displaying the current operation. Depending on your JTAG speed you may see the info window only very short.



3.2.7 Breakpoints



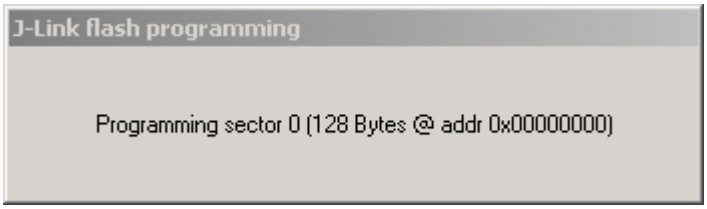
3.2.7.1 Use software breakpoints

This allows to set an unlimited number of breakpoints if the program is located in RAM by setting and resetting breakpoints according to program code.

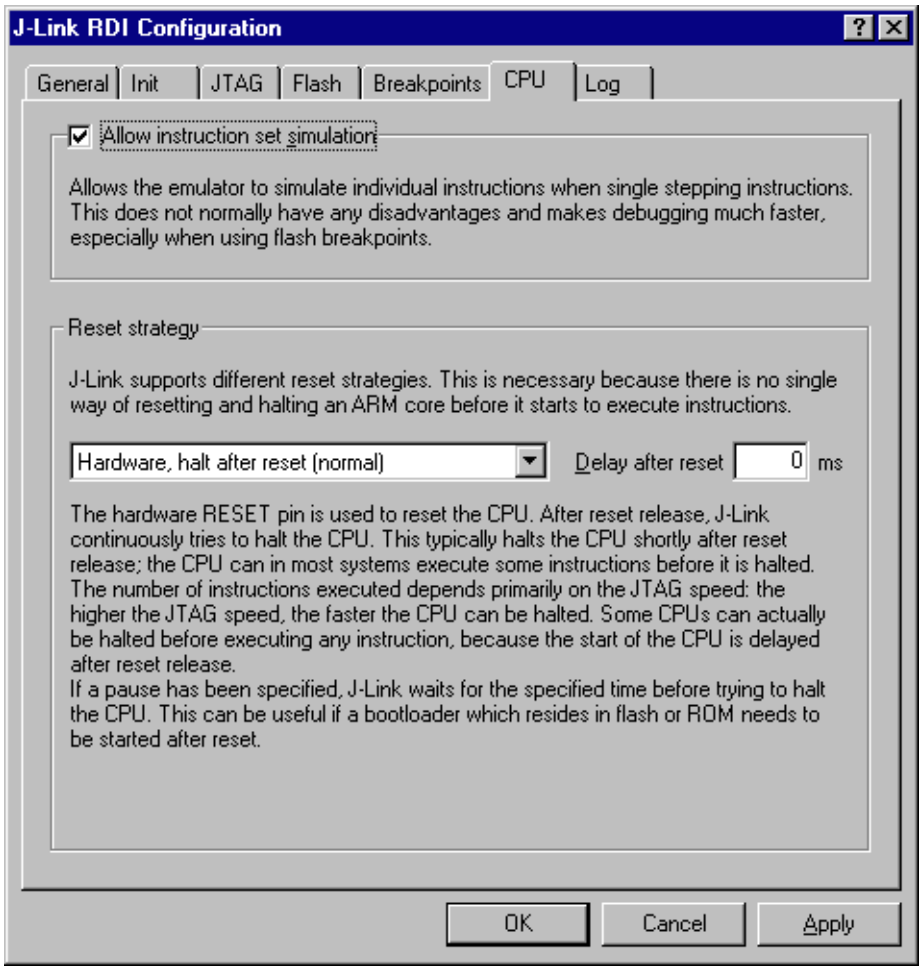
3.2.7.2 Use flash breakpoints

This allows to set an unlimited number of breakpoints if the program is located either in RAM or in flash by setting and resetting breakpoints according to program code.

An info window can be displayed while flash breakpoints are used showing the current operation. Depending on your JTAG speed the info window may only hardly to be seen.



3.2.8 CPU



3.2.8.1 Instruction set simulation

This enables instruction set simulation which speeds up single stepping instructions especially when using flash breakpoints.

3.2.8.2 Reset strategy

This defines the behavior how J-Link RDI should handle resets called by software.

J-Link supports different reset strategies. This is necessary because there is no single way of resetting and halting an ARM core before it starts to execute instructions.

What is the problem if the core executes some instructions after RESET?

The instructions executed can cause various problems. Some cores can be completely "confused", which means they can not be switched into debug mode (CPU can not be halted). In other cases, the CPU may already have initialized some hardware compo-

nents, causing unexpected interrupts or worse, the hardware may have been initialized with illegal values. In some of these cases, such as illegal PLL settings, the CPU may be operated beyond specification, possibly locking the CPU.

Available reset strategies

The following reset strategies, described in detail below, are available:

- Hardware, halt after reset (normal)
- Hardware, halt after reset using WP
- Hardware, halt after reset using DBGRQ
- Hardware, halt with BP@
- Software, for Analog Devices ADuC7xxx MCUs
- No reset

Hardware, halt after reset (normal)

The hardware reset pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU. This typically halts the CPU shortly after reset release; the CPU can in most systems execute some instructions before it is halted. The number of instructions executed depends primarily on the JTAG speed: the higher the JTAG speed, the faster the CPU can be halted.

Some CPUs can actually be halted before executing any instruction, because the start of the CPU is delayed after reset release. If a pause has been specified, J-Link waits for the specified time before trying to halt the CPU. This can be useful if a bootloader which resides in flash or ROM needs to be started after reset.

Hardware, halt after reset using WP

The hardware RESET pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU. This typically halts the CPU shortly after reset release; the CPU can in most systems execute some instructions before it is halted.

The number of instructions executed depends primarily on the JTAG speed: the higher the JTAG speed, the faster the CPU can be halted. Some CPUs can actually be halted before executing any instruction, because the start of the CPU is delayed after reset release.

Hardware, halt after reset using DBGRQ

The hardware RESET pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU. This typically halts the CPU shortly after reset release; the CPU can in most systems execute some instructions before it is halted.

The number of instructions executed depends primarily on the JTAG speed: the higher the JTAG speed, the faster the CPU can be halted. Some CPUs can actually be halted before executing any instruction, because the start of the CPU is delayed after reset release.

Hardware, halt with BP@0

The hardware reset pin is used to reset the CPU. Before doing so, the ICE breaker is programmed to halt program execution at address 0; effectively a breakpoint is set at address 0. If this strategy works, the CPU is actually halted before executing a single instruction.

This reset strategy does not work on all systems for two reasons:

- If nRESET and nTRST are coupled, either on the board or the CPU itself, reset clears the breakpoint, which means the CPU is not stopped after reset.
- Some MCUs contain a bootloader program (sometimes called kernel), which needs to be executed to enable JTAG access.

Software, for Analog Devices ADuC7xxx MCUs

The following sequence is executed:

- The CPU is halted
- A software reset sequence is downloaded to RAM

- A breakpoint at address 0 is set
- The software reset sequence is executed

This sequence performs a reset of CPU and peripherals and halts the CPU before executing instructions of the user program. It is recommended reset sequence for Analog Devices ADuC7xxx MCUs and works with these chips only.

No reset

No reset is performed.

3.2.9 Log

A log file can be generated for J-Link ARM and J-Link RDI. This log files may be useful for debugging and evaluating. They may help you to solve a problem yourself but is also needed by the support to help you with it.

Default path of the J-Link ARM log file: `c:\JLinkARM.log`
Default path of the J-Link RDI log file: `c:\JLinkRDI.log`

Example of logfile content:

```
060:028 (0000) Logging started @ 2005-10-28 07:36
060:028 (0000) DLL Compiled: Oct  4 2005 09:14:54
060:031 (0026) ARM_SetMaxSpeed - Testing speed 3F0F0F0F 3F0F0F0F 3F0F0F0F 3F0F0F0F
3F0F0F0F 3F0F0F0F 3F0F0F0F 3F0F0F0F 3F0F0F0F 3F0F0F0F 3F0F0F0FAuto JTAG
speed: 4000 kHz
060:059 (0000) ARM_SetEndian(ARM_ENDIAN_LITTLE)
060:060 (0000) ARM_SetEndian(ARM_ENDIAN_LITTLE)
060:060 (0000) ARM_ResetPullsRESET(ON)
060:060 (0116) ARM_Reset(): SpeedIsFixed == 0 -> JTAGSpeed = 30kHz >48>  >2EF>
060:176 (0000) ARM_WriteIceReg(0x02,00000000)
060:177 (0016) ARM_WriteMem(FFFFFC20,0004) -- Data:  01 06 00 00 - Writing 0x4 bytes
@ 0xFFFFFC20 >1D7>
060:194 (0014) ARM_WriteMem(FFFFFC2C,0004) -- Data:  05 1C 19 00 - Writing 0x4 bytes
@ 0xFFFFFC2C >195>
060:208 (0015) ARM_WriteMem(FFFFFC30,0004) -- Data:  07 00 00 00 - Writing 0x4 bytes
@ 0xFFFFFC30 >195>
060:223 (0002) ARM_ReadMem (00000000,0004)JTAG speed: 4000 kHz -- Data:  0C 00 00 EA
060:225 (0001) ARM_WriteMem(00000000,0004) -- Data:  0D 00 00 EA - Writing 0x4 bytes
@ 0x00000000 >195>
060:226 (0001) ARM_ReadMem (00000000,0004) -- Data:  0C 00 00 EA
060:227 (0001) ARM_WriteMem(FFFFFF00,0004) -- Data:  01 00 00 00 - Writing 0x4 bytes
@ 0xFFFFFF00 >195>
060:228 (0001) ARM_ReadMem (FFFFF240,0004) -- Data:  40 05 09 27
060:229 (0001) ARM_ReadMem (FFFFF244,0004) -- Data:  00 00 00 00
060:230 (0001) ARM_ReadMem (FFFFFF6C,0004) -- Data:  10 01 00 00
060:232 (0000) ARM_WriteMem(FFFFF124,0004) -- Data:  FF FF FF FF - Writing 0x4 bytes
@ 0xFFFFF124 >195>
060:232 (0001) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:233 (0001) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:234 (0001) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:236 (0000) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:237 (0000) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:238 (0001) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:239 (0001) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:240 (0001) ARM_ReadMem (FFFFF130,0004) -- Data:  00 00 00 00
060:241 (0001) ARM_WriteMem(FFFFFD44,0004) -- Data:  00 80 00 00 - Writing 0x4 bytes
@ 0xFFFFFD44 >195>
060:277 (0000) ARM_WriteMem(00000000,0178) -- Data:  0F 00 00 EA FE FF FF EA ...
060:277 (0000) ARM_WriteMem(000003C4,0020) -- Data:  01 00 00 00 02 00 00 00 ... -
Writing 0x178 bytes @ 0x00000000
060:277 (0000) ARM_WriteMem(000001CC,00F4) -- Data:  30 B5 15 48 01 68 82 68 ... -
Writing 0x20 bytes @ 0x000003C4
060:277 (0000) ARM_WriteMem(000002C0,0002) -- Data:  00 47
060:278 (0000) ARM_WriteMem(000002C4,0068) -- Data:  F0 B5 00 27 24 4C 34 4D ... -
Writing 0xF6 bytes @ 0x000001CC
060:278 (0000) ARM_WriteMem(0000032C,0002) -- Data:  00 47
060:278 (0000) ARM_WriteMem(00000330,0074) -- Data:  30 B5 00 24 A0 00 08 49 ... -
Writing 0x6A bytes @ 0x000002C4
060:278 (0000) ARM_WriteMem(000003B0,0014) -- Data:  00 00 00 00 0A 00 00 00 ... -
Writing 0x74 bytes @ 0x00000330
060:278 (0000) ARM_WriteMem(000003A4,000C) -- Data:  14 00 00 00 E4 03 00 00 ... -
Writing 0x14 bytes @ 0x000003B0
060:278 (0000) ARM_WriteMem(00000178,0054) -- Data:  12 4A 13 48 70 B4 81 B0 ... -
Writing 0xC bytes @ 0x000003A4
060:278 (0000) ARM_SetEndian(ARM_ENDIAN_LITTLE)
060:278 (0000) ARM_SetEndian(ARM_ENDIAN_LITTLE)
060:278 (0000) ARM_ResetPullsRESET(OFF)
060:278 (0009) ARM_Reset():  - Writing 0x54 bytes @ 0x00000178 >3E68>
060:287 (0001) ARM_Halt():  **** Warning: Chip has already been halted.
...
```


Chapter 4

Flash download

This chapter describes how to use flash download with J-Link RDI. It basically allows a debugger to download program into flash even if the debugger does not have a flash loader. This feature requires a separate license from SEGGER.

4.1 Overview

J-Link RDI flash download allows a debugger to download program into flash even if the debugger does not have a flash loader. This way any RDI compliant debugger can be used to download into any supported flash memory. From a debuggers perspective, the flash download works just like download to RAM; the flash programming is handled completely by the J-Link RDI software.

Flash download is a feature of the J-Link RDI software, which requires a separate license from SEGGER.

4.2 Why should I use RDI flash download?

Being able to download code directly into flash from the debugger or integrated IDE significantly shortens the turn-around times when testing software. The flash loader integrated into J-Link RDI is very efficient and allows fast flash programming.

Once the setup of flash download is completed, flash breakpoints can be used without additional configuration (if a license for this feature is present).

4.3 Enabling flash download

Before you can use flash download, some parameters need to be defined correctly and the checkbox **Enable flash programming** needs to be checked. For a detailed description please refer to the chapter *Flash configuration* on page 38.

4.4 Supported flash devices

Manufacturer	Name
Analog Devices	ADuC7020x62 (to E)
Analog Devices	ADuC7020x62 (G on)
Analog Devices	ADuC7021x32 (to E)
Analog Devices	ADuC7021x32 (G on)
Analog Devices	ADuC7021x62 (to E)
Analog Devices	ADuC7021x62 (G on)
Analog Devices	ADuC7022x32 (to E)
Analog Devices	ADuC7022x32 (G on)
Analog Devices	ADuC7022x62 (to E)
Analog Devices	ADuC7022x62 (G on)
Analog Devices	ADuC7024x62 (to E)
Analog Devices	ADuC7024x62 (G on)
Analog Devices	ADuC7025x62 (to E)
Analog Devices	ADuC7025x62 (G on)
Analog Devices	ADuC7025x32 (to E)
Analog Devices	ADuC7025x32 (G on)
Analog Devices	ADuC7026x62 (to E)
Analog Devices	ADuC7026x62 (G on)
Analog Devices	ADuC7027x62 (to E)
Analog Devices	ADuC7027x62 (G on)
Analog Devices	ADuC7030
Analog Devices	ADuC7031
Analog Devices	ADuC7032
Analog Devices	ADuC7033

Table 4.1: Supported flash devices

Analog Devices	ADuC7128
Analog Devices	ADuC7129
Analog Devices	ADuC7229x126
Atmel	AT91SAM7A3
Atmel	AT91SAM7S32
Atmel	AT91SAM7S321
Atmel	AT91SAM7S64
Atmel	AT91SAM7S128
Atmel	AT91SAM7S256
Atmel	AT91SAM7X128
Atmel	AT91SAM7X256
OKI	ML67Q4050
OKI	ML67Q4051
OKI	ML67Q4060
OKI	ML67Q4061
Philips	LPC2101
Philips	LPC2102
Philips	LPC2103
Philips	LPC2104
Philips	LPC2105
Philips	LPC2106
Philips	LPC2114
Philips	LPC2119
Philips	LPC2124

Manufacturer	Name
Philips	LPC2129
Philips	LPC2131
Philips	LPC2132
Philips	LPC2134
Philips	LPC2136
Philips	LPC2138
Philips	LPC2141
Philips	LPC2142
Philips	LPC2144
Philips	LPC2146
Philips	LPC2148
Philips	LPC2194
Philips	LPC2212
Philips	LPC2214
Philips	LPC2292
Philips	LPC2294
ST	STR710FZ1
ST	STR710FZ2
ST	STR711FR0
ST	STR711FR1
ST	STR711FR2
ST	STR712FR0
ST	STR712FR1
ST	STR712FR2
ST	STR715FR0
ST	STR730FZ1
ST	STR730FZ2
ST	STR731FV0
ST	STR731FV1
ST	STR731FV2
ST	STR911FM32
ST	STR911FM44
ST	STR912FM32
ST	STR912FM44
TI	TMS470R1A64
TI	TMS470R1A128
TI	TMS470R1A256
TI	TMS470R1A288
TI	TMS470R1A384
TI	TMS470R1B512
TI	TMS470R1B768
TI	TMS470R1B1M
TI	TMS470R1VF288
TI	TMS470R1VF688
TI	TMS470R1VF689

Table 4.1: Supported flash devices

4.5 Licensing

The software is licensed on a per J-Link basis. The following items are required to use the J-Link RDI software:

1. J-Link ARM
2. J-Link RDI license
3. Flash download license

Chapter 5

Breakpoints in flash memory

This chapter describes how to configure and use breakpoints in flash memory.

The J-Link RDI software contains an additional feature, called flash breakpoints (short flash BPs). Flash breakpoints allow the user to set an unlimited number of software breakpoints when debugging in flash memory, rather than just the 2 hardware breakpoints. This feature requires an additional license from SEGGER.

5.1 How do breakpoints work?

There are basically 2 types of breakpoints in a computer system: Hard ones and soft ones. Hardware breakpoints require a dedicate hardware unit for every breakpoint. In other words, the hardware dictates how many hardware breakpoints can be set simultaneously. ARM7 and ARM 9 cores have 2 breakpoint units (called "watchpoint units" in ARM's documentation), allowing 2 hardware breakpoints to be set. Hardware breakpoints do not require modification of the program code. Software breakpoints are different: The debugger modifies the program and replaces the breakpointed instruction with a special value. Additional software breakpoints do not require additional hardware units in the processor, since simply more instructions are replaced. This is a standard procedure that most debuggers are capable of, however, it requires the program to be located in RAM.

5.2 What is special about software breakpoints in flash?

It allows you to set an unlimited number of breakpoints even if your application program is not located in RAM, but in flash memory. This is a scenario which was very rare before ARM-microcontrollers hit the market. This new technology makes very powerful, yet inexpensive ARM microcontrollers available for systems, which required external RAM before. The downside of this new technology is that it is not possible to debug larger programs on these micros in RAM, since the RAM is not big enough to hold program and data (typically, these chips contain about 4 times as much flash as RAM), and therefore with standard debuggers, only 2 breakpoints can be set. The 2 breakpoint limit makes debugging very tough; a lot of times the debugger requires 2 breakpoints to simply step over a line of code. With software breakpoints in flash, this limitation is gone.

5.3 How does this work?

Basically very simple:

The J-Link RDI software reprograms a sector of the flash to set or clear a breakpoint.

5.4 What performance can I expect?

A RAMCode, specially designed for this purpose, sets and clears flash breakpoints extremely fast; on micros with fast flash the difference between breakpoints in RAM and flash is hardly noticeable.

5.5 How is this performance achieved?

We have put a lot of effort in making flash breakpoints really usable and convenient. Flash sectors are programmed only when necessary; this is usually the moment execution of the target program is started. A lot of times, more then one breakpoint is located in the same flash sector, which allows programming multiple breakpoints by programming just a single sector. The contents of program memory are cached, avoiding time consuming reading of the flash sectors. A smart combination of software and hardware breakpoints allows us to use hardware breakpoints a lot of times, especially when the debugger is source level-stepping, avoiding reprogramming flash in these situations. A built-in instruction set simulator further reduces the number of flash operations which need to be performed. This minimizes delays for the user, maximizing the life time of the flash. All resources of the ARM micro are available to the application program, no memory is lost for debugging. All of the optimizations described above can be disabled.

5.6 Licensing

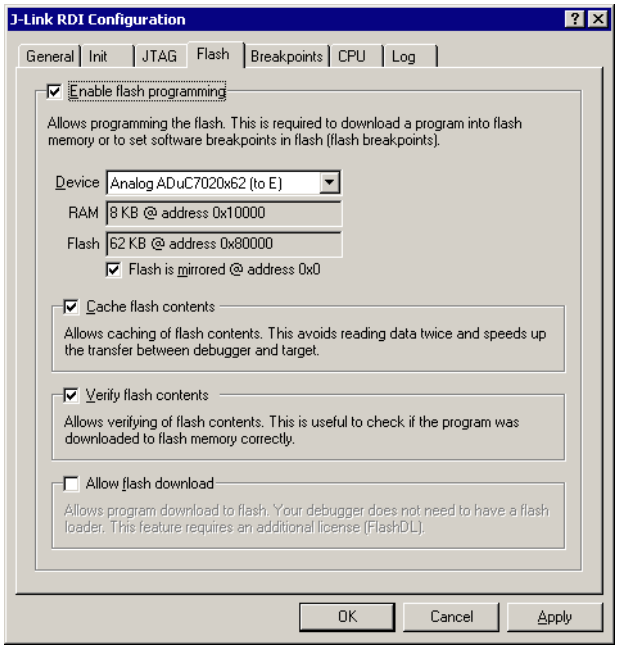
The software is licensed on a per J-Link basis. The following items are required to use the software:

1. J-Link ARM
2. J-Link RDI license
3. Flash breakpoints license

5.7 Setting up flash breakpoints

1. Open the RDI configuration dialog box and click on the **Flash** tab.

2. Set the **Enable flash programming** checkbox and select your processor in the **Device** list. Afterwards select the **Breakpoint** tab.



3. Select **Use software breakpoints** as well as **Use flash breakpoints**. Then click the **OK** button to close the **J-Link RDI configuration** dialog.

4. You can now use flash breakpoints with the debugger of your choice.

Chapter 6

Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger.

It effectively allows the target to do disk operations and console I/O and is used primarily for flash loaders with ARM debuggers such as AXD.

6.1 Overview

Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism is used, to allow functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

This is useful because development hardware often does not have all the input and output facilities of the final system. Semihosting allows the host computer to provide these facilities.

Semihosting is also used for Disk I/O and flash programming; a flash loader uses semihosting to load the target program from disk.

Semihosting is implemented by a set of defined software interrupt (SWI) operations. The application invokes the appropriate SWI and the debug agent then handles the SWI exception. The debug agent provides the required communication with the host. In many cases, the semihosting SWI will be invoked by code within library functions.

Usage of semihosting

The application can also invoke the semihosting SWI directly. Refer to the C library descriptions in the ADS Compilers and Libraries Guide for more information on support for semihosting in the ARM C library.

Semihosting is not used by all tool chains; most modern tool chains (such as IAR) use different mechanisms to achive the same goal.

Semihosting is used primarily by ARM’s tool chain and debuggers, such as AXD.

Since semihosting has been used primarily by ARM, documents published by ARM are the best source of add. information.

For further information on semihosting and the C libraries, see the "C and C++ Libraries" chapter in ADS Compilers and Libraries Guide. Please see also the "Writing Code for ROM" chapter in ADS Developer Guide.

6.2 The SWI interface

The ARM and Thumb SWI instructions contain a field that encodes the SWI number used by the application code. This number can be decoded by the SWI handler in the system. See the chapter on exception handling in ADS Developer Guide for more information on SWI handlers.

Semihosting operations are requested using a single SWI number. This leaves the other SWI numbers available for use by the application or operating system. The SWI used for semihosting is:

0x123456 in ARM state
0xAB in Thumb state

The SWI number indicates to the debug agent that the SWI is a semihosting request. In order to distinguish between operations, the operation type is passed in r0. All other parameters are passed in a block that is pointed to by r1. The result is returned in r0, either as an explicit return value or as a pointer to a data block. Even if no result is returned, assume that r0 is corrupted.

The available semihosting operation numbers passed in r0 are allocated as follows:

0x00 to 0x31 These are used by ARM.
0x32 to 0xFF These are reserved for future use by ARM.
0x100 to 0x1FF Reserved for applications.

6.2.1 Changing the semihosting SWI numbers

It is strongly recommended that you do not change the semihosting SWI numbers 0x123456 (ARM) or 0xAB (Thumb). If you do so you must:

- change all the code in your system, including library code, to use the new SWI number
- reconfigure your debugger to use the new SWI number.

6.3 Implementation of semihosting in J-Link RDI

When using J-Link RDI in default configuration, semihosting is implemented as follows:

- A breakpoint / vector catch is set on the SWI vector.
- When this breakpoint is hit, J-Link RDI examines the SWI number.
- If the SWI is recognized as a semihosting SWI, J-Link RDI emulates it and transparently restarts execution of the application.
- If the SWI is not recognized as a semihosting SWI, J-Link RDI halts the processor and reports an error. (See *Unexpected / unhandled SWIs* on page 58)

6.3.1 DCC semihosting

J-Link RDI does not support using the debug communications channel for semihosting.

6.4 Semihosting with AXD

This semihosting mechanism can be disabled or changed by the following debugger internal variables:

$semihosting_enabled

Set this variable to 0 to disable semihosting. If you are debugging an application running from ROM, this allows you to use an additional watchpoint unit.
Set this variable to 1 to enable semihosting. This is the default.
Set this variable to 2 to enable Debug Communications Channel (DCC) semihosting.
The S bit in $vector_catch has no effect unless semihosting is disabled.

$semihosting_vector

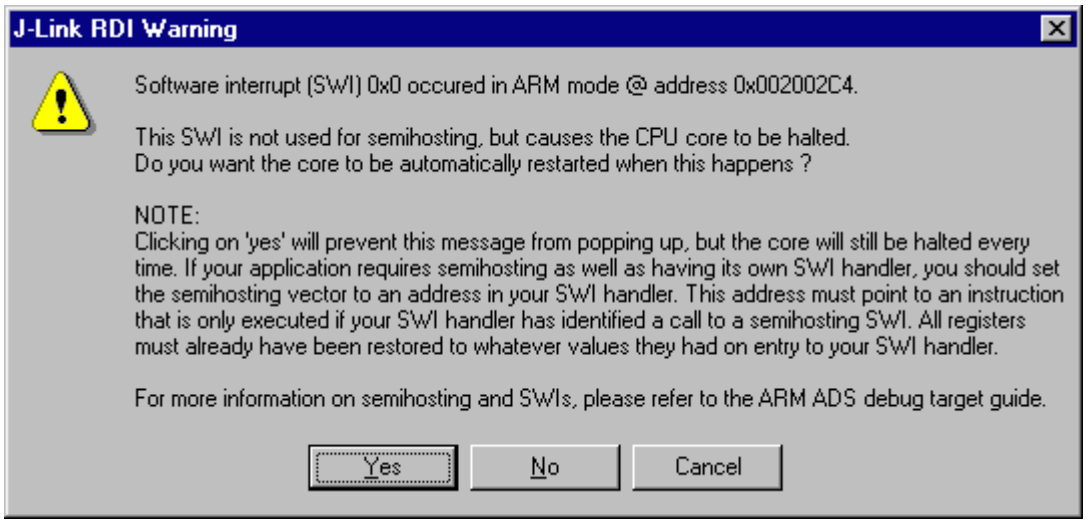
This variable controls the location of the breakpoint set by J-Link RDI to detect a semihosted SWI. It is set to the SWI entry in the exception vector table () by default.

6.4.1 Using SWIs in your application

If your application requires semihosting as well as having its own SWI handler, set $semihosting_vector to an address in your SWI handler. This address must point to an instruction that is only executed if your SWI handler has identified a call to a semihosting SWI. All registers must already have been restored to whatever values they had on entry to your SWI handler.

6.5 Unexpected / unhandled SWIs

When an unhandled SWI is detected by J-Link RDI, the message box below is shown.



This typically indicates that your application is using SWIs not only for semihosting, but also for other purposes, but J-Link RDI stops on every SWI, which is inefficient and affects the real-time behaviour of your application program. This is discouraged; you should follow the instruction in the message box.

Chapter 7

Background information

This chapter provides background information about JTAG and ARM. The ARM7 and ARM9 architecture is based on Reduced Instruction Set Computer (RISC) principles. The instruction set and related decode mechanism are greatly simplified compared with microprogrammed Complex Instruction Set Computer (CISC).

7.1 JTAG

JTAG is the acronym for Joint Test Action Group. In the scope of this document, "the JTAG standard" means compliance with IEEE Standard 1149.1-2001.

7.1.1 Test access port (TAP)

JTAG defines a TAP (Test access port). The TAP is a general-purpose port that can provide access to many test support functions built into a component. It is composed as a minimum of the three input connections (TDI, TCK, TMS) and one output connection (TDO). An optional fourth input connection (nTRST) provides for asynchronous initialization of the test logic.

PIN	Type	Explanation
TCK	Input	The test clock input (TCK) provides the clock for the test logic.
TDI	Input	Serial test instructions and data are received by the test logic at test data input (TDI).
TMS	Input	The signal received at test mode select (TMS) is decoded by the TAP controller to control test operations.
TDO	Output	Test data output (TDO) is the serial output for test instructions and data from the test logic.
TRST	Input (optional)	The optional test reset (TRST) input provides for asynchronous initialization of the TAP controller.

7.1.2 Data registers

JTAG requires at least two data registers to be present: the bypass and the boundary-scan register. Other registers are allowed but are not obligatory.

Bypass data register

A single-bit register that passes information from TDI to TDO.

Boundary-scan data register

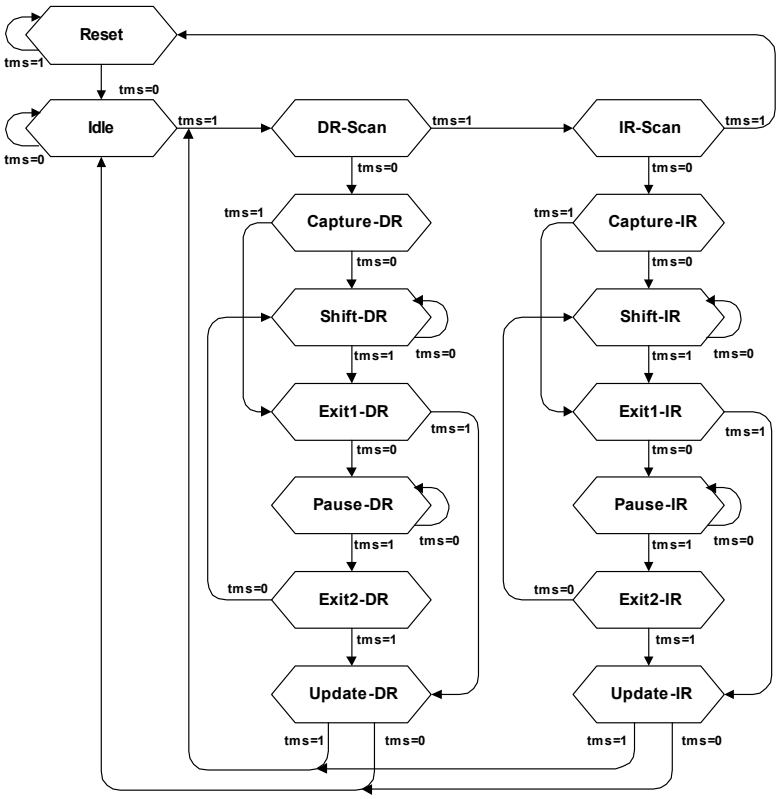
A test data register which allows the testing of board interconnections, access to input and output of components when testing their system logic and so on.

7.1.3 Instruction register

The instruction register holds the current instruction and its content is used by the TAP controller to decide which test to perform or which data register to access. It consist of at least two shift-register cells.

7.1.4 The TAP controller

The TAP controller is a synchronous finite state machine that responds to changes at the TMS and TCK signals of the TAP and controls the sequence of operations of the circuitry.



7.1.4.1 State descriptions

Reset

The test logic is disabled so that normal operation of the chip logic can continue unhindered. No matter in which state the TAP controller currently is, it can change into Reset state if TMS is high for at least 5 clock cycles. As long as TMS is high, the TAP controller remains in Reset state.

Idle

Idle is a TAP controller state between scan (DR or IR) operations. Once entered, this state remains active as long as TMS is low.

DR-Scan

Temporary controller state. If TMS remains low, a scan sequence for the selected data registers is initiated.

IR-Scan

Temporary controller state. If TMS remains low, a scan sequence for the instruction register is initiated.

Capture-DR

Data may be loaded in parallel to the selected test data registers.

Shift-DR

The test data register connected between TDI and TDO shifts data one stage towards the serial output with each clock.

Exit1-DR

Temporary controller state.

Pause-DR

The shifting of the test data register between TDI and TDO is temporarily halted.

Exit2-DR

Temporary controller state. Allows to either go back into Shift-DR state or go on to Update-DR.

Update-DR

Data contained in the currently selected data register is loaded into a latched parallel output (for registers that have such a latch). The parallel latch prevents changes at the parallel output of these registers from occurring during the shifting process.

Capture-IR

Instructions may be loaded in parallel into the instruction register.

Shift-IR

The instruction register shifts the values in the instruction register towards TDO with each clock.

Exit1-IR

Temporary controller state.

Pause-IR

Wait state that temporarily halts the instruction shifting.

Exit2-IR

Temporary controller state. Allows to either go back into Shift-IR state or go on to Update-IR.

Update-IR

The values contained in the instruction register are loaded into a latched parallel output from the shift-register path. Once latched, this new instruction becomes the current one. The parallel latch prevents changes at the parallel output of the instruction register from occurring during the shifting process.

7.2 The ARM core

The ARM7 family is a range of low-power 32-bit RISC microprocessor cores. Offering up to 130MIPs (Dhrystone2.1), the ARM7 family incorporates the 16-bit Thumb instruction set. The family consists of the ARM7TDMI, ARM7TDMI-S and ARM7EJ-S processor cores and the ARM720T cached processor macrocell.

The ARM9 family is built around the ARM9TDMI processor core and incorporates the 16-bit Thumb instruction set. The ARM9 Thumb family includes the ARM920T and ARM922T cached processor macrocells.

7.2.1 Processor modes

The ARM architecture supports seven processor modes.

Processor mode		Description
User	usr	Normal program execution mode
System	sys	Runs privileged operating system tasks
Supervisor	svc	A protected mode for the operating system
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
Interrupt	irq	Used for general-purpose interrupt handling
Fast inter-rupt	fiq	Supports a high-speed data transfer or channel process

Table 7.1: ARM processor modes

7.2.2 Registers of the CPU core

The CPU core has the following registers:

User/ System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0					
R1					
R2					
R3					
R4					
R5					
R6					
R7					
R8					R8_fiq
R9					R9_fiq
R10					R10_fiq
R11					R11_fiq
R12					R12_fiq
R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC					
CPSR					
	SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Table 7.2: ARM CPU registers

= indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode.

The ARM core has a total of 37 registers:

- 31 general-purpose registers, including a program counter. These registers are 32 bits wide.
- 6 status registers. These are also 32-bits wide, but only 12-bits are allocated or need to be implemented.

Registers are arranged in partially overlapping banks, with a different register bank for each processor mode. At any time, 15 general-purpose registers (R0 to R14), one or two status registers and the program counter are visible.

7.2.3 ARM /Thumb instruction set

An ARM core starts execution in ARM mode after reset or any type of exception. Most (but not all) ARM cores come with a secondary instruction set, called the Thumb instruction set. The core is said to be in `Thumb mode` if it is using the thumb instruction set. The thumb instruction set consists of 16-bit instructions, where the ARM instruction set consists of 32-bit instructions. Thumb mode improves code density by approximately 35%, but reduces execution speed on systems with high memory bandwidth (because more instructions are required). On systems with low memory bandwidth, Thumb mode can actually be as fast or faster than ARM mode. Mixing ARM and Thumb code (interworking) is possible.

J-Link ARM fully supports debugging of both modes without limitation.

7.3 EmbeddedICE

EmbeddedICE is a set of registers and comparators used to generate debug exceptions (such as breakpoints).

EmbeddedICE is programmed in a serial fashion using the ARM core controller. It consists of two real-time watchpoint units, together with a control and status register. You can program one or both watchpoint units to halt the execution of instructions by ARM core. Two independent registers, debug control and debug status, provide overall control of EmbeddedICE operation.

Execution is halted when a match occurs between the values programmed into EmbeddedICE and the values currently appearing on the address bus, data bus, and various control signals. Any bit can be masked so that its value does not affect the comparison.

Either of the two real-time watchpoint units can be configured to be a watchpoint (monitoring data accesses) or a breakpoint (monitoring instruction fetches). You can make watchpoints and breakpoints data-dependent.

EmbeddedICE is an additional debug hardware within the core, therefore the EmbeddedICE debug architecture requires almost no target resources (for example, memory, access to exception vectors, and time).

7.3.1 Breakpoints and watchpoints

Breakpoints

A "breakpoint" stops the core when a selected instruction is executed. It is then possible to examine the contents of both memory (and variables).

Watchpoints

A "watchpoint" stops the core if a selected memory location is accessed. For a watchpoint (WP), the following properties can be specified:

- Address (including address mask)
- Type of access (R, R/W, W)
- Data (including data mask)

Software / hardware breakpoints

Hardware breakpoints are "real" breakpoints, using one of the 2 available watchpoint units to breakpoint the instruction at any given address. Hardware breakpoints can be set in any type of memory (RAM, ROM, Flash) and also work with self-modifying code. Unfortunately, there is only a limited number of these available (2 in the EmbeddedICE). When debugging a program located in RAM, another option is to use software breakpoints. With software breakpoints, the instruction in memory is modified. This does not work when debugging programs located in ROM or Flash, but has one huge advantage: The number of software breakpoints is not limited.

7.3.2 The ICE registers

The two watchpoint units are known as watchpoint 0 and watchpoint 1. Each contains three pairs of registers:

- address value and address mask
- data value and data mask
- control value and control mask

The following table shows the function and mapping of EmbeddedICE registers.

Register	Width	Function
0x00	3	Debug control
0x01	5	Debug status
0x04	6	Debug comms control register
0x05	32	Debug comms data register
0x08	32	Watchpoint 0 address value
0x09	32	Watchpoint 0 address mask
0x0A	32	Watchpoint 0 data value
0x0B	32	Watchpoint 0 data mask
0x0C	9	Watchpoint 0 control value
0x0D	8	Watchpoint 0 control mask
0x10	32	Watchpoint 1 address value
0x11	32	Watchpoint 1 address mask
0x12	32	Watchpoint 1 data value
0x13	32	Watchpoint 1 data mask
0x14	9	Watchpoint 1 control value
0x15	8	Watchpoint 1 control mask

Table 7.3: Function and mapping of EmbeddedICE registers

For more information about EmbeddedICE please see the technical reference manual of your ARM CPU. (www.arm.com)

7.4 Flash programming

J-Link ARM comes with a DLL, which allows - amongst other functionalities - reading and writing RAM, CPU registers, starting and stopping the CPU and setting breakpoints. The standard DLL does not have API functions for flash programming. However, the functionality offered can be used to program the flash. In that case a flashloader is required.

7.4.1 How does flash programming via J-Link ARM work ?

This requires extra code. This extra code typically downloads a program into the RAM of the target system, which is able to erase and program the flash memory. This program is called RAMCode and "knows" how to program the flash; it contains an implementation of the flash programming algorithm for the particular flash. Different flash devices have different programming algorithms; the programming algorithm also depends on other things such as endianess of the target system and organization of the flash memory (e.g. 1*8 bits, 1*16 bits, 2*16 bits or 32 bits). The RAMCode also requires the data to be programmed into the flash memory. There are two ways of supplying this data:

- Data download to RAM
- Data download via DCC.

7.4.1.1 Data download to RAM

The data (or part of it) is downloaded to an other part of the RAM of the target system. The instruction pointer (R15) of the CPU is then set to the start address of the RAMCode, the CPU is started, executing the RAMCode. The RAMCode, which contains the programming algorithm for the flash chip, copies the data into the flash chip. The CPU is stopped after this. This process may have to be repeated until the entire data is programmed into the flash.

7.4.1.2 Data download via DCC

In this case, the RAMCode is started as described above before downloading any data. The RAMCode then communicates with the PC (via DCC, JTAG and J-Link ARM), transferring data to the target. The RAMCode then programs the data into flash and waits for new data from the host. The write memory functions of J-Link ARM are used to transfer the RAMCode only, but not to transfer the data. The CPU is started and stopped only once. Using DCC for communication is typically faster than using write memory functions for RAM download since the overhead is lower.

7.4.2 Available options for flash programming

There are different solutions available to program internal or external flash memory connected to ARM cores using J-Link ARM.

7.4.2.1 J-Flash ARM - Complete flash programming solution.

J-Flash ARM is a stand-alone Windows application, which can read / write data files and program the flash in almost any ARM core supported by J-Link ARM. J-Flash ARM requires an extra license from SEGGER.

7.4.2.2 JLinkARMFlash.dll - A DLL with flash programming capabilities.

An enhanced version of the `JLinkARM.dll` with additional API functions, which allow loading and programming of data files. This DLL comes with a sample executable, as well as the source code of this executable and a project file. This can be an interesting option if you want to write your own programs for production purposes.

This DLL also requires an extra license from SEGGER; please contact us for more information.

7.4.2.3 J-Link RDI Flash download - Allows flash download from any RDI-compliant tool chain.

RDI (Remote Debug Interface) is a standard for "debug transfer agents" such as J-Link ARM. The J-Link RDI software allows using J-Link ARM from any RDI compliant debugger. You can use the flash download option integrated in the J-Link RDI software to download your application program into flash memory.

The J-Link RDI software as well as the flash download option require licenses from SEGGER.

7.4.2.4 Flash loader of compiler / debugger vendor such as IAR.

A lot of debuggers (some of them integrated into a workbench / IDE) come with their own flash loaders. The flash loaders can of course be used if they match to your flash configuration, which is something that needs to be checked with the debugger vendor.

7.4.2.5 Write your own flash loader

Implement your own flash loader using the functionality of the `JLinkARM.dll` as described above. This can be a time consuming process and requires in-depth knowledge of the flash programming algorithm used as well as the target system.

Chapter 8

FAQs

You can find in this chapter a collection of frequently asked questions (FAQs) together with answers.

8.1 FAQs

Q: Which CPUs are supported?
A: J-Link RDI is based on J-Link ARM and should work with any ARM7 / ARM9 core. For a list of supported cores see section “Supported ARM Cores” in the J-Link ARM manual.

Q: Which CPUs are flash breakpoint supported?
A: For a list of supported cores see section *Supported flash devices* on page 47.

Q: What is the advantage of flash download versus the flash loader that comes with my IDE?
A: In a lot of cases, the J-LINK RDI flash download is significantly faster than that provided by the IDE. Another advantage is that it uses the same flash programming code being used for flash breakpoints, so it is very easy to set up flash breakpoints if you are already using J-Link RDI flash download.

Chapter 9

Support

This chapter contains troubleshooting tips together with solutions for common problems which might occur when using J-Link RDI. There are several steps you can take before contacting support. Performing these steps can solve many problems and often eliminates the need for assistance.

9.1 Troubleshooting

9.1.1 General procedure

If you experience problems with J-Link RDI, you should follow the steps below to solve these problems:

1. Close all running applications on your host system.
2. Disconnect the J-Link ARM device from USB.
3. Power-off target.
4. Re-connect J-Link ARM with host system (attach USB cable).
5. Power-on target.
6. Try your target application again. If the problem vanished, you are done; otherwise continue.
7. Close all running applications on your host system again.
8. Disconnect the J-Link ARM device from USB.
9. Power-off target.
10. Re-connect J-Link ARM with host system (attach USB cable).
11. Power-on target.
12. Start `JLink.exe`.
13. If `JLink.exe` reports the J-Link ARM serial number and the target processor's core ID, your J-Link ARM is working properly and cannot be the cause of the problem.
14. If `JLink.exe` is unable to read the target processor's core ID you should analyze the communication between your target and J-Link ARM with a logic analyzer or oscilloscope. Follow the instructions in chapter "Support|Signal analysis" in the J-Link ARM users manual.
15. If your problem persists and you own an original Segger J-Link ARM (not an OEM version), see section *Contacting support* on page 70.

9.1.2 Typical problem scenarios

J-Link RDI doesn't seem to do anything

Most likely reason:

The J-Link RDI DLL may not initialized by the debugger.

Remedy:

Please restart your debugger.

9.2 Contacting support

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section *General procedure* on page 70. You may also try your J-Link ARM with another PC and if possible with another target system to see if it works there. If the device functions correctly, the USB setup on the original machine or your target hardware is the source of the problem, not J-Link ARM.

If you need to contact support, please send the following information to support@segger.com:

- A detailed description of the problem.
- J-Link ARM serial number.
- Output of JLink.exe if available.
- Your findings of the signal analysis.
- Information about your target hardware (processor, board etc.).

J-Link ARM is sold directly by SEGGER or as OEM-product by other vendors. We can support only official SEGGER products.

Chapter 10

Glossary

This chapter explains important terms used throughout this manual.

Application Program Interface

A specification of a set of procedures, functions, data structures, and constants that are used to interface two or more software components together.

Big-endian

Memory organization where the least significant byte of a word is at a higher address than the most significant byte. See Little-endian.

Cache cleaning

The process of writing dirty data in a cache to main memory.

Coprocessor

An additional processor that is used for certain operations, for example, for floating-point math calculations, signal processing, or memory management.

Dirty data

When referring to a processor data cache, data that has been written to the cache but has not been written to main memory. Only write-back caches can have dirty data, because a write-through cache writes data to the cache and to main memory simultaneously. The process of writing dirty data to main memory is called cache cleaning.

Dynamic Linked Library (DLL)

A collection of programs, any of which can be called when needed by an executing program. A small program that helps a larger program communicate with a device such as a printer or keyboard is often packaged as a DLL.

EmbeddedICE

The additional hardware provided by debuggable ARM processors to aid debugging.

Host

A computer which provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

ICache

Instruction cache.

ICE Extension Unit

A hardware extension to the EmbeddedICE logic that provides more breakpoint units.

ID

Identifier.

IEEE 1149.1

The IEEE Standard which defines TAP. Commonly (but incorrectly) referred to as JTAG.

Image

An executable file that has been loaded onto a processor for execution.

In-Circuit Emulator (ICE)

A device enabling access to and modification of the signals of a circuit while that circuit is operating.

Instruction Register

When referring to a TAP controller, a register that controls the operation of the TAP.

IR

See Instruction Register.

Joint Test Action Group (JTAG)

The name of the standards group which created the IEEE 1149.1 specification.

Little-endian

Memory organization where the least significant byte of a word is at a lower address than the most significant byte. See also Big-endian.

Memory coherency

A memory is coherent if the value read by a data read or instruction fetch is the value that was most recently written to that location. Memory coherency is made difficult when there are multiple possible physical locations that are involved, such as a system that has main memory, a write buffer and a cache.

Memory management unit (MMU)

Hardware that controls caches and access permissions to blocks of memory, and translates virtual to physical addresses.

Memory Protection Unit (MPU)

Hardware that controls access permissions to blocks of memory. Unlike an MMU, an MPU does not translate virtual addresses to physical addresses.

Multi-ICE

Multi-processor EmbeddedICE interface. ARM registered trademark.

nSRST

Abbreviation of System Reset. The electronic signal which causes the target system other than the TAP controller to be reset. This signal is known as nSYSRST in some other manuals. See also nTRST.

nTRST

Abbreviation of TAP Reset. The electronic signal that causes the target system TAP controller to be reset. This signal is known as nICERST in some other manuals. See also nSRST.

Open collector

A signal that may be actively driven LOW by one or more drivers, and is otherwise passively pulled HIGH. Also known as a "wired AND" signal.

Processor Core

The part of a microprocessor that reads instructions from memory and executes them, including the instruction fetch unit, arithmetic and logic unit and the register bank. It excludes optional coprocessors, caches, and the memory management unit.

Program Status Register (PSR)

Contains some information about the current program and some information about the current processor. Often, therefore, also referred to as Processor Status Register.

Is also referred to as Current PSR (CPSR), to emphasize the distinction between it and the Saved PSR (SPSR). The SPSR holds the value the PSR had when the current function was called, and which will be restored when control is returned.

Remapping

Changing the address of physical memory or devices after the application has started executing. This is typically done to allow RAM to replace ROM once the initialization has been done.

Remote Debug Interface (RDI)

RDI is an open ARM standard procedural interface between a debugger and the debug agent. The widest possible adoption of this standard is encouraged.

Scan Chain

A group of one or more registers from one or more TAP controllers connected between TDI and TDO, through which test data is shifted.

Semihosting

A mechanism whereby the target communicates I/O requests made in the application code to the host system, rather than attempting to support the I/O itself.

SWI

Software Interrupt. An instruction that causes the processor to call a programer-specified subroutine. Used by ARM to handle semihosting.

TAP Controller

Logic on a device which allows access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1.

Target

The actual processor (real silicon or simulated) on which the application program isrunning.

TCK

The electronic clock signal which times data on the TAP data lines TMS, TDI, and TDO.

TDI

The electronic signal input to a TAP controller from the data source (upstream). Usually this is seen connecting the Multi-ICE Interface Unit to the first TAP controller.

TDO

The electronic signal output from a TAP controller to the data sink (downstream). Usually this is seen connecting the last TAP controller to the Multi-ICE Interface Unit.

Test Access Port (TAP)

The port used to access a device's TAP Controller. Comprises TCK, TMS, TDI, TDO and nTRST (optional).

Transistor-transistor logic (TTL)

A type of logic design in which two bipolar transistors drive the logic output to one or zero. LSI and VLSI logic often used TTL with HIGH logic level approaching +5V and LOW approaching 0V.

Watchpoint

A location within the image that will be monitored and that will cause execution to stop when it changes.

Word

A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Index

W

