# J-Link RDI

JTAG RDI Interface for J-Link ARM Emulator



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



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

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of the manufacturer. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2004 - 2007 SEGGER Microcontroller GmbH & Co. KG, Hilden / Germany

#### 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	Ву	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	<ul> <li>Chapter "Using J-Link RDI with different debuggers":</li> <li>* Section "KEIL µVision3 IDE" added.</li> <li>Chapter "Configuration":</li> <li>* Some changes in chapter structure.</li> <li>* Section "Location of Config file" renamed to "Config file"</li> <li>* Section "Config file" enhanced.</li> </ul>
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	ΤQ	Updated list of supported flash devices.
3	060703	00	Section "Reset strategy": Added listing of available reset types.
2	051111	ΤQ	Adding description of adaptive clocking. Minor corrections.
1	051028	00	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

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

This manual uses the following typographic conventions:

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 developmenttime 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

EMBEDDED SOFTWARE (Middleware)

# Sesser

#### 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.

embedded system with a USB client controller. Bulk communication and most standard device classes are sup-

#### emUSB USB device stack A USB stack designed to work on any

ported.

**United States Office:** 

http://www.segger-us.com

#### 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**

1	Introduction	on	9
	1.1 1.1.1 1.2 1.3 1.4 1.5	What is RDI? Features of J-Link RDI Requirements Basic principles Purchase Licensing	10 10 10 11 11
2	Using J-L	ink RDI with different debuggers	.13
	2.1 2.1.1 2.1.2 2.1.3 2.2 2.2.1 2.2.2 2.2.3 2.3 2.3 2.3 2.3 2.3 2.3 2.3	IAR Embedded Workbench IDE Software version Configuring to use J-Link RDI Limitations. ARM's AXD (ARM Developer Suite, ADS) Software version Configuring to use J-Link RDI Limitations. ARM's RVDS (RealView developer suite). Software version Configuring to use J-Link RDI Limitations. GHS MULTI Software version Configuring to use J-Link RDI Limitations. KEIL µVision IDE Software version Configuring to use J-Link RDI Limitations. KEIL µVision IDE Software version Configuring to use J-Link RDI Limitations.	14 14 15 16 16 17 18 18 23 24 24 24 24 27 27 27 30
3	Configura	tion	.31
	3.1 3.1.1 3.1.2 3.1.3 3.2 3.2.1 3.2.2 3.2.3 3.2.4 3.2.5 3.2.6 3.2.7 3.2.8 3.2.9	Overview Configuration file JLinkRDI.ini Using different configurations Using mutliple J-Links simulatenously Configuration dialog General Init Comands in the macro file Example of macro file JTAG Flash configuration Breakpoints CPU Log	32 32 32 33 33 35 36 36 37 38 39 40 43
4	Flash dov	vnload	.45
	4.1 4.2 4.3	Overview Why should I use RDI flash download? Enabling flash download	46 46 46

	4.4 4.5	Supported flash devices	47 ⊿0
_	T.J		
5	Breakpoir	its in flash memory	51
	5.1 5.2	How do breakpoints work? What is special about software breakpoints in	52
	БЭ	flash?	52
	5.5	What performance can I expect?	52
	55	How is this performance achieved?	52
	5.6	Licensing	53
	5.7	Setting up flash breakpoints	53
6	Semihost	ing	55
	6.1	Overview	56
	6.2	The SWI interface	56
	6.2.1	Changing the semihosting SWI numbers	57
	6.3	Implementation of semihosting in J-Link RDI	57
	6.3.1	DCC semihosting	57
	6.4	Semihosting with AXD	57
	6.4.1	Using SWIs in your application	57
	6.5	Unexpected / unhandled SWIs	58
7	Backgrou	nd information	59
	7.1	JTAG	60
	7.1.1	Test access port (TAP)	60
	7.1.2	Data registers	60
	7.1.3	Instruction register	60
	7.1.4	The TAP controller	61
	7.2	The ARM core	62
	7.2.1	Processor modes	63
	7.2.2	Registers of the CPU core	63
	7.2.3	ARM /Thumb instruction set	64
	7.3		64
	7.3.1	Breakpoints and watchpoints	64
	7.3.2		65
	7.4	FidSh programming via 1 Link APM work 2	00
	7.4.1	Available entions for flash programming	600
	7.4.2		00
8	FAQs		67
	8.1	FAQs	68
9	Support		69
	9.1	Troubleshooting	70
	9.1.1	General procedure	70
	9.1.2	Typical problem scenarios	70
	9.2	Contacting support	70
10	) Glossary	/	71

# 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 available 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.

Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker	Factory Settings       Setup     Download     Extra Options     Plugins       Driver     Imain     Imain
Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI Third-Party Driver	Setup macros

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

Category:		Factory Settings
General Options	BDL	
Assembler Custom Build Build Actions	Manufacturer RDI driver	
Linker Debugger Simulator Angel	Allow hardware reset	Note Use the RDI menu to specify additional driver settings. (This
IAR ROM-monitor J-Link Macraigor RDI	ETM trace	Catch exceptions       Reset
Third-Party Driver	Log RDI <u>communication</u> \$TOOLKIT_DIR\$\cspycomm.log	Undef Prefetch

 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.

14	KIAR Embedded Workbench IDE							
<u>F</u> ile	<u>E</u> dit	⊻iew	Project	<u>R</u> DI	<u>T</u> ools	<u>W</u> indow	Help	)
	<b>2</b>		🚭	<u>_</u> c	onfigure			•

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 <code>JLinkRDI.dll</code> 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:

Cł	oose Target					? ×
Г	Target Environme	ents —				
	Target	RDI	File		Version	Add
	ARM TPA	1.5.1	C:\Tool\C\\RVT.DLL C:\Tool\C\\armulate.dll		1.0.0.19	
						<u>H</u> emove
						Re <u>n</u> ame
						Save As
						200000
						<u>C</u> onfigure
	Use the ARM Deb execute ARM prog instructions in soft	bugger wit grams witł ware.	h the 'ARMulator' Instruction nout physical ARM hardware,	Set Simulator. Thi by simulating the	s allows you to ARM	
				ОК	Cancel	Help

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

Open				? ×
Look jn: 🔁	JLinkRDI	•	(† 🔁 🖻	* ⊞-
JLinkArm.c				
L				
File <u>n</u> ame:	JLinkRDI.dll			<u>O</u> pen
Files of <u>type</u> :	DLLs (*.dll)		•	Cancel

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

Cł	oose Target					? ×
Г	Target Environm	ients ——				
	Target	RDI	File		Version	Add
	ARM TPA	1.5.1	C:\Tool\C\\RVT.DLL		1.0.0.19	
	J-Link	1.5.1	C:VLinkRDIVLinkRDI.dll		2.40a	<u>H</u> emove
						Re <u>n</u> ame
						<u>S</u> ave As
						<u>C</u> onfigure
	Segger JLink ARI	M JTAG				
				ОК	Cancel	Help

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:

AVD . IADM 1 . Columnia		DVDC21\statt	
File Search Processor View	ws Sustem Views Fx	ecute Options Wi	indow Help
ARM_1 - Registers		22	void Taskl(void) (
Register	Value	23	while (1) {
Current	{}	24	LED_ToggleLED1();
r0	0x20001580	25	OS_Delay (200);
rl	0x20001588	26	}
r2	0x00000150	27	}
-r3	0x00001E80	20	/*******
	0x2000063C	30	*
-r5	0x00001E70	31	* main
r6	0x00000000	32	*
r7	0x00000000	33	***********
r8	0x00000000	34	
r9	0x00000000	35	int main(void) {
r10	0x000027B0	36	US_IncDI(); /* Initially disable interrupts */
r11	0x00000000	38	05_INICKEIN(); /" INICIALIZE 05 "/ 05_INICKEIN(); /* INICIALIZE 05 */
-r12	0x00000451	39	LED Init(): /* initialize LED ports */
-r13	0x200015A0	40	/* You need to create at least one task here ! */
-r14	0x00001EA5	41	OS_CREATETASK(&TCB0, "HP Task", Task0, 100, Stack0);
pc	0x00000450	42	OS_CREATETASK(&TCB1, "LP Task", Task1, 50, Stack1);
cpsr	nzcvqIFT_SVC	43	OS_Start(); /* Start multitasking */
spsr	nzcvqift_User	44	return 0;
⊞ User/System	{}	45	}
⊞·FIQ	{}		
Target Image Files Class	Break	points	ARM_1 - Memory Start address 0x0
B ➡₩ C:\temp\embOS_Start	STR71x\RC State	e Processor Po:	isition Tab1 · Hex · No prefix Tab2 · Hex · No prefix Tab3 · Hex · No prefix Tab4 · 🗸 🕨
		ARM_1 St	Address 0 4 8 c
			0x00000000 E59FF018 E59FF018 E59FF018 E59FF018
			0x00000010 E59FF018 E1A00000 E59FF014 E59FF014
			0x00000020 000024A4 0000003C 00000040 00000044
			0x00000030 0000048 000022F0 0000004C EAFFFFE
			0x00000040 EAFFFFFE EAFFFFFE EAFFFFFE EAFFFFFE
For Help, press F1			<no pos=""> J-Link ARM_1 Start_STR71x.axf //</no>

## 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 <code>JLinkRDI.dll</code> 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:

File Edit Find View Project Tools Debug Help	
	2 🟥 📰 - 🛄 🧮 📰 😢 👗 State: Unknown
File: \$NO_SOURCE Find:	
Not connected - no PC or scope	🔺 Register 🛛 👻 🗖 🗖
Click to Connect to a Target	<no context="" register=""></no>
Src {rtosinit_str71x.c+{ vectors.s+/	Core F
Name Ualue  Name Ualue Name Ualue Name Ualue Name Ualue Name Ualue Name Ualue Name Ualue Name Ualue Name Ualue Name Ualue Name Name Ualue Name Name Name Name Name Name Name Nam	Mamory A A
None>	
Crnd / StallO / Build / FileFind / SrcCtrl / Log /	T
For more information, select Help from Menu	Ln 1, Col 1 NUM ///

- **₩DEBUG<Start\_STR71x>** \_ 🗆 × \_\_\_\_\_<u>File</u> <u>E</u>dit Fi<u>n</u>d <u>V</u>iew <u>P</u>roject <u>T</u>ools <u>D</u>ebug <u>H</u>elp New Ľ ▶ 内田 丹 19 酥 🛧 🔸 🗶 🗈 彫 卧 🛱 禁 🗉 🎞 🎞 🖽 参 👗 State: Unknown <u>0</u>pen... Ctrl+O 💌 Line: 🔡 👘 💌 File No1 ▼\_ . C1: <No Register Context> <u>W</u>orkspace Alt+0 Connection • Connect to Target. Connection Properties.. Alt+Shift+0 De Core 4 ► ⊽ Ţ Ţ Recent <u>F</u>iles  ${\sf Recent\,Wor\underline{k}spaces}$ • Vatch1 (Watch2 (Watch3 ( 1 ► • E<u>x</u>it Ţ Crnd StdlO Build FileFind SrcCtrl ALog  $|\mathbf{F}|$ Ln 1, Col 1 NUM Select target(s) to connect to
- 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** 

🊱 Connection Control (	Souhail\rvdebug.brd)
<u>H</u> elp	
Name	Description
🖃 🚱 arm-a-rr	ARM Ltd. RDI targets
🗄 🚰 ARMulato 🛛 Coll	apse All t simulator
🕀 👺 Multi-IC 🛛 🛛 🗛	and Vehicles erface (parallel port)
🗄 🕾 Remote_A	ol (serial port)
🖃 🖓 Server 🛛 🔹 Cor	nection Properties
🗄 🔁 localhos 🛛 🗛 do	d/Remove/Edit Devices
ER ARM-ARM-PP Sele	ect Board-File nect
🕀 🚟 ARMOAK_M	- maror for arress Jonnect (ARM+Oak)
E 🖗 ARM-VIA-LP	ARM Vehicle
🗄 🚟 MOT_WIGGLER	Macraigor Wiggler
E 🖗 ARM-ARM-DIR	ARM Ltd. Direct Connection
⊞ 7% VPB926EJ-S_U	Versatile Platform for ARM926EJ-S (USB port)
🖃 🖗 ARM-ARM-NU	RealViewICE
🕀 🚟 RealView ICE	ARM JTAG debug interface (TCP/IP)

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

F	DI Target List		× × × × × × × × × × × × × × × × × × ×	<											
	Use the check boxes to add or remove RDI targets from the connection manager:														
	Name	Version	Description												
	♥₽ Remote_A ♥	v1.2 v2.2.5 v1.4	Angel debug protocol (serial port) ARM JTAG debug interface (parallel port) ARM instruction set simulator												
		Add DLL	Reset list Configure Remove Duplicate												

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.

Create New RDI Target	×									
Enter a name and a description for the new entry in the connection list:										
Short Name (example - "Dual 7TDMI"):										
JLinkRDI										
Description (example - "Multi-ICE with two ARM7s"):										
J-Link ARM RDI Interface										
OK Cancel										

6. Back in the **RDI Target List** Dialog

Select **JLink-RDI** and click **Configure**. For configuration details refer to chapter *Configuration* on page 31.

R	DI Target List			×
	Use the check boxes to	) add or remove	e RDI targets from the connection manager:	
	Name	Version	Description	
	🗹 🖗 JLinkRDI		J-Link ARM RDI Interface	
	<b>⊠</b> ¢ <b>?</b> Remote_A	v1.2	Angel debug protocol (serial port)	
	🗹 🦿 Multi-ICE	v2.2.5	ARM JTAG debug interface (parallel port)	
	🗹 🔜 ARMulator	v1.4	ARM instruction set simulator	
		Add DLL	Reset list Configure Remove Duplicate	
			Close	

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.

J-Link RDI Configuration	?	×
General Init JTAG F	Flash Breakpoints CPU Log	
-JTAG speed		
○ Auto selection		
C Adaptive <u>c</u> locking		
⊙ 30 ▼ <u>k</u> Hz		
JTAG scan chain with	n multiple devices	
Position 0	<u>[Rilen</u> ]	
0 is closest to TDI.	Sum of IRLens of devices closer to TDI. IRLen of ARM chips is 4.	
-		
	⊻erify JTAG config	
	OK Cancel Apply	

8. In the **Connection control** dialog, expand the **JLink ARM RDI Interface** and select the ARM\_0 Processor. Close the **Connection Control** Window.

😓 Connection Control (Administrat\rvdebug.brd) 📃 🗖 🗙											
<u>H</u> elp											
Name	Description										
🖃 🚷 ARM-A-RR	ARM Ltd. RDI targets										
🕂 🖶 🕵 ARMulator	ARM instruction set simulator										
🗄 🕵 JLinkRDI.dll	J-Link ARM RDI Interface										
└ <b>@</b> ARM #120	ARM on localhost										
🕞 🙈 Server	Connection Broker										
🛛 🗄 🔁 localhost	Simulator Broker										
E- 💫 ARM-VIA-LP	Motorola/Macraigor Wiggler emulator										
🗄 🕵 MOT_WIGGLER	Macraigor Wiggler										
🖃 🙈 ARM-ARM-DIR	ARM Ltd. Direct Connection										
📗 🗄 🕵 VPB926EJ – S_U	Versatile Platform for ARM926EJ-S (USB port)										

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

RVDEBUG <start_str71x> = @ARM_0;ARM-A-RR</start_str71x>						- 🗆 ×
File Edit Find View Project Tools Debug Help						
🗋 🕞 🗐 🗴 🖻 🖻 🛯 🖓 🖓 🖓 🖓 👘 👘 🐄 🚺 🛧 📕 🛫 🕮 🖓	D 00 12 .			🕮 🌮 🛣	State:	Stopped
	- + + <u>A</u> r				1	
			Renister	γ		<b>.</b>
Click to Load 'C:\temp\embOS Start STR71x\RAM\Start STR71x.axf'			סס	00000000	P1	
			R2	000000000	R3	000000
			R4	00000000	R5	000000
			R6	00000000	R7	000000
	R8	00000000	R9	000000		
			R10	00000000	R11	000000
			R12	00000000	SP	000000
			LR	000000000	PC	000000
			NZCW	FTO TRO S	TATEM	ODF
Dsm Src {rtosinit_str71x.c+{ vectors.s+/			• •	Core Debu	1g /	
▼ Name Value ▼ Name Value		<ru><!--</td--><td> </td><td></td><td></td><td></td></ru>				
<pre>c0x0000000 </pre>		<noaddr></noaddr>				
		<noaddr></noaddr>				
		<noaddr></noaddr>				
		<noaddr></noaddr>				
		<noaddr></noaddr>				
A D Call Stock (Longle / Station 1 / 1 ) A D Mitchel (Mitchel / Mitchel / Mi	lië l	<noaddr></noaddr>				
	اع الك	<no1ddrs< td=""><td></td><td></td><td></td><td></td></no1ddrs<>				
connect, route 2						
↓ > connect 10						
Advanced_info searched in: Local Advanced_info						
Warning: Vector catching specification is not supported by target						
Warning: No stack/heap or top of memory defined - using defaults.						
Connected Target is: ARM						
Vehicle: ARM MultiP, RDI v1.51 via DLL						
Mode: Little Endian						
Stop>						
Crnd / StallO / Build / FileFind / SrcCtrl /*Log /			<			
Currently opened file	Ln 1, Col 1	1			N	UM//

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.

Create N	New Connection Method	
Name:	J-Link	
Туре:	Custom	•
	Create Ca	incel

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.

Connection Editor									
Name:	J-Link								
Туре:	Custom								
Log Connection to file:									
MULTI Targe	t Setup Script: 🛛 🖌 🚰								
Connect for:  C Download (Download and debug application) C Attach (Debug application already on target) C Board Setup (Debug board initialization sequence)									
Server:	rdiserv								
Arguments:	-config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll"								
Connect	OK Cancel Revert Apply								

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

Connection E	ditor										
Name:	[J-Link										
Туре:	Custom										
🔲 Log Conr	nection to file:										
MULTI Targe	et Setup Script:										
Connect for:	<ul> <li>Download (Download and debug application)</li> <li>Attach (Debug application already on target)</li> <li>Board Setup (Debug board initialization sequence)</li> </ul>										
Server:	Idiserv										
Arguments:	-config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll"										
mode=down	mode=download rdiserv -config -dll "C\Program Files\SEGGER\ULinkARM_V350g\ULinkRDI.dll"										
Connect	OK Cancel Revert Apply										

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.

*	C: \\	₩ork\	Basic	.ghs - M	ULTI D	) ebug	ger																				-		×
Eile	D	ebug	⊻iew	Browse	Targel	t <u>I</u> oo	ls <u>C</u>	onfig	Wind	dows	Help	)																	
_					-	<b>-</b>							0	0	0	0	× 10	NIG	a I			1							
1			<b>,</b> ,		<b>e</b> 1	ě 1	<u>•</u> ]		<b>•</b>	₿ţ	E₽₽	し	9	Q	Q	U.	{ ₿	4	0	2		J							
46			- 77	*																									
47			vo	<mark>id</mark> wai	t ( 🛛	70id	)																						
48	1		- (7,	/* Beg	in																								
49	2			unsi	gned	int	wai	ting	g_ti	me	;																		
		•	0x3	20027c	wai	it:				b5	00	PUS	н	( L)	R)														
		•	0x3	20027e	wai	it+Ox	(2:			bO	84	SUB		SP	, SF	P, 1	16												
50	3			chan	ge_sp	peed	0	;																					
L.,		. •	0x)	200280	wai	it+Ox	(4:		f7f	fff	d4	BL		Ox:	ffff	iffe	18 (c	hang	ge_	spe	ed i	0x2	0022	2C)	)				
51	4	•		for(	waiti	ing_t	ime	= 0	); ឃ	ait	ing_	time	< :	LedS	peed	1; u	Jait	ing_	ti	ne+	+) ;								
		•	UX.	200284	wai	LC+UX	.8:			20	00	MOV		RU	<w8< td=""><td>aiti</td><td>.ng_</td><td>C 1 me</td><td>≥,</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></w8<>	aiti	.ng_	C 1 me	≥,	0									
		· -	UX.	400286	wai	LC+UX	a:			eu	00	B		UX	ງເພຣ	1104	loxe	: (0)	(20)	12.8	a) )								—
		STOP	щo Ux.	200288	wai	LC+UX	(C:			30	01	ADD		RU	<w8< td=""><td>a101</td><td>.ng_</td><td>time</td><td>e&gt;,</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></w8<>	a101	.ng_	time	e>,	1									
			UX.	20028a	wai	LC+UX	(e:			49.	44 01-	LDR		RI	, [PC	., 13	61	'erec	15p	eea	(0)	200	314)	9					
			0.0	200280	wai	LC+UX	.10:			60	00	CWD		R3	, [RI	.,0]				<b>D</b> 2									
			0.0	200208	wai	LC+UX	.14.			42	90 4-	- CHP		- RU	< W8	1101	ing ing	CIME	22, 	кз - //		020							
52	5		- υx. /	4 End	wai	LC+UX	(14)			us.	La	всс		UX.			. 4 ( )	arca	FUX	s ()	JX20	1020	011						
52	5		0.21	200202	wei	+ ±0.v	.16.			ъо	n4	100		зÞ	gr	. 1	6												
		1	0.2	2002.92	wai	i++0x	18.			be	09	POP		10	, Dr 31	· · ·													
			0x	200296	wai	it.+Ox	(1a)			47	18	BX		R3															
53												2.2																	-
																													dah.
Mi	ked	-	File:	main.c									-	Proc:	wait	:										-		<b>;</b>	⇒
															-													· ·	É.
r	1n:	isne	a exe	eutin	g set	up s	cri	рс. Лата		1	11																		-
D	OWI	nica	d ao	progr	aun ce	:xt a	ma	uate	а. г	ica.	se w	art.	••																
<i>v</i>	1001	nina		North)	Pogia	aha																							
1 L 10	uni mu	ning ⊤r∖l	.01	WOLK (	Dabit	. gus	,																						
- P	.01	. 1/																											
																													-
┛			_				_		_		_																		Add.
<u> </u>		Trg*	1/0	Sri																				S	TOPF	PED IN	ISIDE		

#### 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 <code>JLinkRDI.dll</code> has been tested with KEIL  $\mu Vision3$  IDE version 3.34. There should be no problems with other versions of KEIL  $\mu Vision.$ 

All screenshots are taken from KEIL  $\mu$ Vision3 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.

Options for Target 'LPC2378 Flash'	×
Device Target Output Listing User C/C++ Asm	Linker Debug Utilities
C Use Simulator Settings	Lise: RDI Interface Driver     Settings     ULINK ARM Debugger     ULINK Cartex-M3 Debugger
Load Application at Startup     Run to main() Initialization File:	✓ Load RDI Interface Driver Luminary Eval Board Initializatiq Signum Systems JTAGjet
Restore Debug Session Settings     Breakpoints     Vatchpoints & PA     Memory Display	Restore Debug Session Settings ✓ Breakpoints ✓ Toolbox ✓ Watchpoints ✓ Memory Display
CPU DLL: Parameter:           SARM.DLL         -cLPC2100	Driver DLL: Parameter:
Dialog DLL: Parameter: DARMP.DLL -pLPC2378	Dialog DLL: Parameter: TARMP.DLL -pLPC2378
OK Ca	ancel Defaults Help

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.

RDI Interface Driver Setup		×
Browse for RDI Driver DLL		
C:\Program Files\SEGGER\JLinkARN	M_V359a\JLinkRDI.dll	
Browse for ToolConf File		
Debug		
Cache Options		
Cache <u>L</u> ode	Configure <u>R</u> DI Driver	
	OK Cancel	<u>H</u> elp

The J-Link RDI Configuration dialog will be opened. For detailed information about the configuration 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**).

🕎 Blinky - μVision3 - [Disassembly]				- 🗆 ×
Rile Edit View Project Debug Flash Periph	erals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp			_8×
🎦 😂 🖬 🗿 🐰 🖻 🛍 🗅 오 오 🎼	∉ ∧ % % % <b>% </b> ፟ <mark>%</mark>	]	• A # ← → 12   \$   <b>(</b>   <b>□ )</b> + → 14   <b>•</b> • • •	
R#   E↓ S (?) (?) (?) (?)   ↔ ½± ()±	Q 💭 🖤 🔰 🗖 🗄 🔤 🥕			
Project Workspace 👻 🗙	332: Vectors	LDR	PC, Reset_Addr	
Register Value	⇒ 0x00000000 E59FF018	LDR	PC,[PC,#0x0018]	
	0x0000004 E59FF018		PC.[PC.#0x0018]	
RU UXUUUUUUUU R1 0v0000000	334:	LDR	PC, SWI_Addr	
R2 0x0000000	0x0000008 E59FF018	LDR	PC,[PC,#0x0018]	
	335: 0×000000C F59FF018		PC, PHDT_HOOP PC,[PC,#AxAA18]	
R4 0x00000000	336:	LDR	PC, DAbt_Addr	
R5 0x00000000	0x00000010 E59FF018	LDR	PC,[PC,#0x0018]	
B7 0v0000000	337:	NOP	; Reserved Vector	
	0x00000014 B9206E50	STMLTDB	R0!, {R4,R6,R9-R11,R13-R14}	
R9 0x0000000	339:	LDR	PC, [PC, #-0x0120] ; Vector from VicVectAdo	Ir 👘
R10 0x00000000	0x00000018 E51FF120		PC,[PC,#-0x0120] PC, EI0 0ddr	
R11 UXUUUUUUUU	340.	LVN	rc, riq_Huur	
B13 (SP) 0x0000000	342: Reset_Addr	DCD	Reset_Handler	
	343: Undef_Addr	DCD	Undef_Handler	
R15 (PC) 0x0000000	344: SWI_HUUr 345: PAbt Addr	DCD	Swi_Handler PAbt Handler	
	346: DAbt_Addr	DCD	DAbt_Handler	
The Ux00000010	347:	DCD	0 ; Reserved Address	
E Fast Interrupt	348: IRU_Addr 349• FIO Addr	DCD	IKU_Handler FIO_Handler	
Interrupt		000		<b>_</b>
	E LPC2300.s E Blinky.c	ECD_4bit.c	c   🗐 IHU.c   📄 Abstract.txt 🙀 Disassembly	
Load "C:\\Keil303o\\AR	M\\RV30\\Boards\\Ke	eil\\MC	B230 Address: 0x3fffc000	<b>_</b>
			0x3FFFC000; 0000000 000000	00
			0x3FFFC008: 0000000 000000	00
			0x3FFFC010; 600000DF 000000	00
			0x3FFFC018: 0000000 000000	00
			0x3FFFC020: 600000DF 000000	00
3			0x3FFFC028: 00000000 000000	00
			0x3FFFC030: 0000000 000000	00
ASSIGN BreakDisable Br	eakEnable BreakKill	1	0x3FFFC038: 600000DF 000000	00
â.				Memor
For Help, press F1			RDI Interface Driver t1: 0.0000000 sec	

#### Configuring flash download via J-Link RDI

You can use the J-Link RDI flash download feature instead of the standard  $\mu\text{Vision-flash}$  loader.

**Note:** This feature requires an additional licence. A free trial license is available 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.

Options for Target 'LPC2378 Flash'
Device Target Output Listing User C/C++ Asm Linker Debug Utilities
Configure Flash Menu Command
Use Target Driver for Flash Programming
RDI Interface Driver 💌 Settings 🔽 Update Target before Debugging
Init File:Edit
C Use External Tool for Flash Programming
Command: LPC210x_ISP.EXE
Arguments: "#H" "X \$D COM1: 38400 1
Run Independent
OK Cancel Defaults Help

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.

Options for Target 'LPC2378 Flash'
Device Target Output Listing User C/C++ Asm Linker Debug Utilities
Configure Flash Menu Command
O Use Target Driver for Flash Programming
RDI Interface Driver 💽 Settings 🔽 Update Target before Debugging
Init File: Edit
Use External Tool for Flas Select Flash Programmer Command: LPC210x_ISP Arguments: "#H" "X \$D C Run Indep OK
OK Cancel Defaults Help

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 <code>JLinkRDI.ini</code>. This file is located in the same directory as <code>JLinkRDI.dll</code>.

# 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  $proj_DIR$ , rdl\), 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

J-Link RDI Configuration 🛛 😯 🗙
General       Init       JTAG       Flash       Breakpoints       CPU       Log         J-Link-RDI is an RDI compliant software for J-Link         ARM. It requires a license (RDI), which can be obtained from SEGGER (www.segger.com).         This software is also capable of programming the flash memory of several ARM micros, which can be used to download your program to flash (Requires the add. license "FlashDL") and to set an unlimited number of software breakpoints in flash (Requires the add. license "FlashBP").         Connection to J-Link         Image: I
Location of config file  Location of config file  License <u>License</u> <u>Reset Config</u>
OK Cancel <u>A</u> pply

#### 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.

×				management	DI License m	l-Link R
	Expiration	er	Serial number		e	Featu
	 		1			
	mber	Display <u>s</u> erial nu	cense	<u>D</u> elete lice	d license	

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

Add license						x
Please enter y	our license(s)!					
			(	)K	Cancel	

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

J-	J-Link RDI License management 🛛 🔀				
	Feature	Serial number	Expiration		
	RDI	1	never expires		
	Add license Delete lice	ense Display serial nu	mber OK		

## 3.2.2 Init

J-Link RDI Configuration	? ×
General Init JTAG Flash Breakpoints CPU Log	
Use macro file	
OK Cancel	Apply

#### 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		
<pre>SetJTAGSpeed(x);</pre>	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);	Reade a 8/16/22 hit value		
Read16(Addr);	Redus a 0/10/32 bit value, Addr = address to read (as bey value)		
Read32(Addr);	Addr – address to read (as nex value)		
<pre>Verify8(Addr, Data);</pre>	Verifies a 8/16/32 bit value, Addr = address to verify (as hex value)		
<pre>Verify16(Addr, Data);</pre>			
<pre>Verify32(Addr, Data);</pre>	Data = data to verify (as hex value)		
<pre>Write8(Addr, Data);</pre>	Writes a 8/16/32 bit value,		
<pre>Write16(Addr, Data);</pre>	Addr = address to write (as hex value)		
Write32(Addr, Data);	Data = data to write (as hex value)		
<pre>WriteVerify8(Addr, Data);</pre>	Writes and verifies a 8/16/32 bit value,		
<pre>WriteVerify16(Addr, Data);</pre>	Addr = address to write (as hex value)		
<pre>WriteVerify32(Addr, Data);</pre>	Data = data to write (as hex value)		
<pre>WriteRegister(Reg, Data);</pre>	Writes a register		
<pre>WriteJTAG_IR(Cmd);</pre>	Writes the JTAG instruction register		
<pre>WriteJTAG_DR(nBits, Data);</pre>	Writes the JTAG data register		

 Table 3.1: Macro file commands

# 3.2.4 Example of macro file
# 3.2.5 JTAG

J-Link RDI Configuration	? 🗙
General Init JTAG F	Flash Breakpoints CPU Log
JTAG speed	
C <u>A</u> uto selection	
C Adaptive <u>c</u> locking	
⊙ 30 <u> </u>	
JTAG scan chain with	n multiple devices
Position 0	[Rilen 0
0 is closest to TDI.	Sum of IRLens of devices closer to TDI. IRLen of ARM chips is 4.
	<u>V</u> erify JTAG config
	OK Cancel <u>A</u> pply

## 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

J-Link RDI Configuration
General Init JTAG Flash Breakpoints CPU Log
Enable flash programming
Allows programming the flash. This is required to download a program into flash memory or to set software breakpoints in flash (flash breakpoints).
Device Atmel AT91SAM7S64 Clock speed 48000000 Hz
RAM 16 KB @ address 0x200000
Flash 64 KB @ address 0x100000
Flash is mirrored @ address 0x0
Cache flash contents Allows caching of flash contents. This avoids reading data twice and speeds up the transfer between debugger and target.
Allow flash download
Allows program download to flash. Your debugger does not need to have a flash loader. This feature requires an additional license (FlashDL).
✓ Show info window during download
OK Cancel <u>Apply</u>

## 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

J-Link RDI Configuration
General Init JTAG Flash Breakpoints CPU Log
□ Use software breakpoints
Software breakpoints (as opposed to hardware breakpoints) are breakpoints which modify program memory. This allows setting an unlimited number of breakpoints if the program is located in RAM.  Use flash breakpoints  Allows setting an unlimited number of breakpoints if the program is located in RAM or flash, which is extremely valuable when debugging a program located in flash.  This feature is available only if flash programming is enabled!
Show info window during program
OK Cancel <u>A</u> pply

## 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.

J-Link flash programming Programming sector 0 (128 Bytes @ addr 0x00000000)

# 3.2.8 CPU

	figuration					<u> </u>
eneral   Init	JTAG Flas	sh   Breakpoints	; CPU	Log	1	
🔽 Allow in	struction set <u>s</u> imu	lation				
Allows the This does r especially (	mulator to simula ot normally have when using flash l	ate individual insti any disadvantag breakpoints.	uctions v jes and m	vhen sing Jakes det	le stepping bugging mud	instructions. ch faster,
- Reset strate	gy					
J-Link supp way of rese	orts different rese tting and halting	et strategies. This an ARM core bel	is neces iore it star	sary beca ts to exe	ause there is cute instruc	s no single tions.
Hardware,	halt after reset (r	normal)	•	<u>D</u> elay a	after reset 🛛	0 ms
The hardw continuous release; the The numbe higher the be halted b after reset If a pause	re RESET pin is tries to halt the CPU can in mos of instructions e TAG speed, the efore executing a elease. as been specifie his can be useful	used to reset the CPU. This typica it systems execut executed depend faster the CPU c any instruction, be d, J-Link, waits fo if a bootloader w	CPU. Af illy halts the s primarily an be hal acause the r the spear	ter reset i ne CPU s instruction on the J ted. Som ie start of cified time	release, J-Li shortly after i s before it is TAG speed e CPUs car the CPU is before tryin sh or ROM t	nk reset : the n actually delayed
be started	fter reset.			Jes in na:		needs to

## 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 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(0x02,0000000) 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(0000000,0004) -- Data: 0D 00 00 EA - Writing 0x4 bytes @ 0x0000000 >195> 060:226 (0001) ARM ReadMem (00000000,0004) -- Data: 0C 00 00 EA 060:227 (0001) ARM\_WriteMem(FFFFF00,0004) -- 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 (FFFFF6C,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 (FFFF130,0004) -- Data: 00 00 00 060:234 (0001) ARM\_ReadMem (FFFF130,0004) -- Data: 00 00 00 00 060:236 (0000) ARM\_ReadMem (FFFFF130,0004) -- Data: 060:237 (0000) ARM\_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 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 @ 0x0000000 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 B 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 3

# 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

J-Link RDI User Guide

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

#### Licensing 4.5

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

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

**CHAPTER 4** 

# 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.

J-Link RD	I Warning 🛛 🗙
•	Software interrupt (SWI) 0x0 occured in ARM mode @ address 0x002002C4. This SWI is not used for semihosting, but causes the CPU core to be halted. Do you want the core to be automatically restarted when this happens ?
	NOTE: Clicking on 'yes' will prevent this message from popping up, but the core will still be halted every time. If your application requires semihosting as well as having its own SWI handler, you should set the 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.
	For more information on semihosting and SWIs, please refer to the ARM ADS debug target guide.
	Yes No Cancel

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	Туре	Explanation
ТСК	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 asyn- chronous 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. Fr	inction a	nd manning of EmbeddedICE registers

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 floatingpoint 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.

## тск

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

# A

Adaptive clocking7 Application Program Interface7	2 2
ARM	
Processor modes6	3
Registers6	3
Thumb instruction set6	4

# В

Big-endian	 72
-	

# С

Cache cleanin	g	72
Coprocessor	-	72

# D

Dirty data	72
Dynamic Linked Library (DLL)	72

# Е

EmbeddedICE	 64,	72

# Η

# Ι

ICache	72 72
ID	
IEEE 1149.1	72
Image	72
In-Circuit Emulator	72
Instruction Register	72
IR	73

# J

J-Link	
FAQs	56, 68
Features	
Joint Test Action Group (JTAG)	73

JTAG
<b>L</b> Little-endian73
M Memory coherency
<b>N</b> nSRST
Open collector
PProcessor Core73Program Status Register (PSR)73
<b>R</b> Remapping
<b>S</b> Scan Chain
<b>T</b> TAP Controller

Test Access Port (	TAP)		7	4
Transistor-transis	tor lo	ogic (TTL)	7	4

#### W

Watchpoint64	ŀ,	74
Word	••	74