J-Link / J-Trace User Guide



Software Version V4.50 Manual Rev. 0

Date: June 1, 2012

Document: UM08001



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.

© 2012 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

In den Weiden 11 D-40721 Hilden

Germany

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

Revisions

This manual describes the J-Link and J-Trace device.

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

Revision	Date	Ву	Explanation
V4.50a Rev. 0	120530	EL	Chapter "Target interfaces and adapters" * Section "Pinout for SWD" updated.
V4.47a Rev. 0	120419	AG	Chapter "Device specifics" * Section "Texas Instruments" updated.
V4.46 Rev. 0	120416	EL	Chapter "Support" updated.
V4.42 Rev. 0	120214	EL	Chapter "Working with J-Link" * Section "J-Link script files" updated.
V4.36 Rev. 1	110927	EL	Chapter "Flash download" added. Chapter "Flash breakpoints" added. Chapter "Target interfaces and adapters" * Section "20-pin JTAG/SWD connector" updated. Chapter "RDI" added. Chapter "Setup" updated. Chapter "Device specifics" updated. Several corrections / updates.

Revision	Date	Ву	Explanation
V4.36 Rev. 0	110909	AG	Chapter "Working with J-Link" * Section "J-Link script files" updated.
V4.26 Rev. 1	110513	KN	Chapter "Introduction"
V4.26 Rev. 0	110427	KN	* Section "J-Link / J-Trace models" corrected. Several corrections.
V4.24 Rev. 1	110228	AG	Chapter "Introduction" * Section "J-Link / J-Trace models" corrected. Chapter "Device specifics" * Section "ST Microelectronics" updated.
V4.24 Rev. 0	110216	AG	Chapter "Device specifics" * Section "Samsung" added. Chapter "Working with J-Link" * Section "Reset strategies" updated. Chapter "Target interfaces and adapters" * Section "9-pin JTAG/SWD connector" added.
V4.23d	110202	AG	Chapter "J-Link and J-Trace related software" * Section "J-Link software and documentation package in detail" updated. Chapter "Introduction" * Section "Built-in intelligence for supported CPU-cores" added.
V4.21g	101130	AG	Chapter "Working with J-Link" * Section "Reset strategies" updated. Chapter "Device specifics" * Section "Freescale" updated. Chapter "Flash download and flash breakpoints * Section "Supported devices" updated * Section "Setup for different debuggers (CFI flash)" updated.
V4.21	101025	AG	Chapter "Device specifics" * Section "Freescale" updated.
V4.20j	101019	AG	Chapter "Working with J-Link" * Section "Reset strategies" updated.
V4.20b	100923	AG	Chapter "Working with J-Link" * Section "Reset strategies" updated.
90	100818	AG	Chapter "Working with J-Link" * Section "J-Link script files" updated. * Section "Command strings" upadted. Chapter "Target interfaces and adapters" * Section "19-pin JTAG/SWD and Trace connector" corrected. Chapter "Setup" * Section "J-Link configurator added."
89	100630	AG	Several corrections.
88	100622	AG	Chapter "J-Link and J-Trace related software" * Section "SWO Analyzer" added.
87	100617	AG	Several corrections.
86	100504	AG	Chapter "Introduction" * Section "J-Link / J-Trace models" updated. Chapter "Target interfaces and adapters" * Section "Adapters" updated.
85	100428	AG	Chapter "Introduction" * Section "J-Link / J-Trace models" updated.
84	100324	KN	Chapter "Working with J-Link and J-Trace" * Several corrections Chapter Flash download & flash breakpoints * Section "Supported devices" updated

Revision	Date	Ву	Explanation
83	100223	KN	Chapter "Introduction" * Section "J-Link / J-Trace models" updated.
82	100215	AG	Chapter "Working with J-Link" * Section "J-Link script files" added.
81	100202	KN	Chapter "Device Specifics" * Section "Luminary Micro" updated. Chapter "Flash download and flash breakpoints" * Section "Supported devices" updated.
80	100104	KN	Chapter "Flash download and flash breakpoints * Section "Supported devices" updated
79	091201	AG	Chapter "Working with J-Link and J-Trace" * Section "Reset strategies" updated. Chapter "Licensing" * Section "J-Link OEM versions" updated.
78	091023	AG	Chapter "Licensing" * Section "J-Link OEM versions" updated.
77	090910	AG	Chapter "Introduction" * Section "J-Link / J-Trace models" updated.
76	090828	KN	Chapter "Introduction" * Section" Specifications" updated * Section "Hardware versions" updated * Section "Common features of the J-Link product family" updated Chapter "Target interfaces and adapters" * Section "5 Volt adapter" updated
75	090729	AG	Chapter "Introduction" * Section "J-Link / J-Trace models" updated. Chapter "Working with J-Link and J-Trace" * Section "SWD interface" updated.
74	090722	KN	Chapter "Introduction" * Section "Supported IDEs" added * Section "Supported CPU cores" updated * Section "Model comparison chart" renamed to "Model comparison" * Section "J-Link bundle comparison chart" removed
73	090701	KN	Chapter "Introduction" * Section "J-Link and J-Trace models" added * Sections "Model comparison chart" & "J-Link bundle comparison chart"added Chapter "J-Link and J-Trace models" removed Chapter "Hardware" renamed to "Target interfaces & adapters" * Section "JTAG Isolator" added Chapter "Target interfaces and adapters" * Section "Target board design" updated Several corrections
72	090618	AG	Chapter "Working with J-Link" * Section "J-Link control panel" updated. Chapter "Flash download and flash breakpoints" * Section "Supported devices" updated. Chapter "Device specifics" * Section "NXP" updated.
71	090616	AG	Chapter "Device specifics" * Section "NXP" updated.

Revision	Date	By	Explanation
70	090605	AG	Chapter "Introduction" * Section "Common features of the J-Link product family" updated.
69	090515	AG	Chapter "Working with J-Link" * Section "Reset strategies" updated. * Section "Indicators" updated. Chapter "Flash download and flash breakpoints" * Section "Supported devices" updated.
68	090428	AG	Chapter "J-Link and J-Trace related software" * Section "J-Link STM32 Commander" added. Chapter "Working with J-Link" * Section "Reset strategies" updated.
67	090402	AG	Chapter "Working with J-Link" * Section "Reset strategies" updated.
66	090327	AG	Chapter "Background information" * Section "Embedded Trace Macrocell (ETM)" updated. Chapter "J-Link and J-Trace related software" * Section "Dedicated flash programming utilities for J-Link" updated.
65	090320	AG	Several changes in the manual structure.
64	090313	AG	Chapter "Working with J-Link" * Section "Indicators" added.
63	090212	AG	Chapter "Hardware" * Several corrections. * Section "Hardware Versions" Version 8.0 added. Chapter "Working with J-Link and J-Trace"
62	090211	AG	 * Section "Reset strategies" updated. Chapter J-Link and J-Trace related software * Section "J-Link STR91x Commander (Command line tool)" updated. Chapter "Device specifics" * Section "ST Microelectronics" updated. Chapter "Hardware" updated.
61	090120	ΤQ	Chapter "Working with J-Link" * Section "Cortex-M3 specific reset strategies"
60	090114	AG	Chapter "Working with J-Link" * Section "Cortex-M3 specific reset strategies"
59	090108	KN	Chapter Hardware * Section "Target board design for JTAG" updated. * Section "Target board design for SWD" added.
58	090105	AG	Chapter "Working with J-Link Pro" * Section "Connecting J-Link Pro the first time" updated.
57	081222	AG	Chapter "Working with J-Link Pro" * Section "Introduction" updated. * Section "Configuring J-Link Pro via web interface" updated. Chapter "Introduction" * Section "J-Link Pro overview" updated.
56	081219	AG	Chapter "Working with J-Link Pro" * Section "FAQs" added. Chapter "Support and FAQs" * Section "Frequently Asked Questions" updated.
55	081218	AG	Chapter "Hardware" updated.

Revision	Date	Ву	Explanation
54	081217	AG	Chapter "Working with J-Link and J-Trace" * Section "Command strings" updated.
53	081216	AG	Chapter "Working with J-Link Pro" updated.
52	081212	AG	Chapter "Working with J-Link Pro" added. Chapter "Licensing" * Section "Original SEGGER products" updated.
51	081202	KN	Several corrections.
50	081030	AG	Chapter "Flash download and flash breakpoints" * Section "Supported devices" corrected.
49	081029	AG	Several corrections.
48	080916	AG	Chapter "Working with J-Link and J-Trace" * Section "Connecting multiple J-Links / J-Traces to your PC" updated.
47	080910	AG	Chapter "Licensing" updated.
46	080904	AG	Chapter "Licensing" added. Chapter "Hardware" Section "J-Link OEM versions" moved to chapter "Licensing"
45	080902	AG	Chapter "Hardware" Section "JTAG+Trace connector" JTAG+Trace connector pinout corrected. Section "J-Link OEM versions" updated.
44	080827	AG	Chapter "J-Link control panel" moved to chapter "Working with J-Link". Several corrections.
43	080826	AG	Chapter "Flash download and flash breakpoints" Section "Supported devices" updated.
42	080820	AG	Chapter "Flash download and flash breakpoints" Section "Supported devices" updated.
41	080811	AG	Chapter "Flash download and flash breakpoints" updated. Chapter "Flash download and flash breakpoints", section "Supported devices" updated.
40	080630	AG	Chapter "Flash download and flash breakpoints" updated. Chapter "J-Link status window" renamed to "J-Link control panel" Various corrections.
39	080627	AG	Chapter "Flash download and flash breakpoints" Section "Licensing" updated. Section "Using flash download and flash breakpoints with different debuggers" updated. Chapter "J-Link status window" added.
38	080618	AG	Chapter "Support and FAQs" Section "Frequently Asked Questions" updated Chapter "Reset strategies" Section "Cortex-M3 specific reset strategies" updated.
37	080617	AG	Chapter "Reset strategies" Section "Cortex-M3 specific reset strategies" updated.

Revision	Date	By	Explanation
36	080530	AG	Chapter "Hardware" Section "Differences between different versions" updated. Chapter "Working with J-Link and J-Trace" Section "Cortex-M3 specific reset strategies" added.
35	080215	AG	Chapter "J-Link and J-Trace related software" Section "J-Link software and documentation package in detail" updated.
34	080212	AG	Chapter "J-Link and J-Trace related software" Section "J-Link TCP/IP Server (Remote J-Link / J-Trace use)" updated. Chapter "Working with J-Link and J-Trace" Section "Command strings" updated. Chapter "Flash download and flash breakpoints" Section "Introduction" updated. Section "Licensing" updated. Section "Using flash download and flash breakpoints with different debuggers" updated.
33	080207	AG	Chapter "Flash download and flash breakpoints" added Chapter "Device specifics:" Section "ATMEL - AT91SAM7 - Recommended init sequence" added.
32	0080129	SK	Chapter "Device specifics": Section "NXP - LPC - Fast GPIO bug" list of device enhanced.
31	0080103	SK	Chapter "Device specifics": Section "NXP - LPC - Fast GPIO bug" updated.
30	071211	AG	Chapter "Device specifics": Section "Analog Devices" updated. Section "ATMEL" updated. Section "Freescale" added. Section "Luminary Micro" added. Section "NXP" updated. Section "OKI" added. Section "OKI" added. Section "ST Microelectronics" updated. Section "Texas Instruments" updated. Chapter "Related software": Section "J-Link STR91x Commander" updated
29	070912	SK	Chapter "Hardware", section "Target board design" updated.
28	070912	SK	Chapter "Related software": Section "J-LinkSTR91x Commander" added. Chapter "Device specifics": Section "ST Microelectronics" added. Section "Texas Instruments" added. Subsection "AT91SAM9" added.
28	070912	AG	Chapter "Working with J-Link/J-Trace": Section "Command strings" updated.
27	070827	ΤQ	Chapter "Working with J-Link/J-Trace": Section "Command strings" updated.
26	070710	SK	Chapter "Introduction": Section "Features of J-Link" updated. Chapter "Background Information": Section "Embedded Trace Macrocell" added. Section "Embedded Trace Buffer" added.

Revision	Date	By	Explanation
25	070516	SK	Chapter "Working with J-Link/J-Trace": Section "Reset strategies in detail" - "Software, for Analog Devices ADuC7xxx MCUs" updated - "Software, for ATMEL AT91SAM7 MCUs" added. Chapter "Device specifics" Section "Analog Devices" added. Section "ATMEL" added.
24	070323	SK	Chapter "Setup": "Uninstalling the J-Link driver" updated. "Supported ARM cores" updated.
23	070320	SK	Chapter "Hardware": "Using the JTAG connector with SWD" updated.
22	070316	SK	Chapter "Hardware": "Using the JTAG connector with SWD" added.
21	070312	SK	Chapter "Hardware": "Differences between different versions" supplemented.
20	070307	SK	Chapter "J-Link / J-Trace related software": "J-Link GDB Server" licensing updated. Chapter "J-Link / J-Trace related software" updated
19	070226	SK	and reorganized. Chapter "Hardware" "List of OEM products" updated
18	070221	SK	Chapter "Device specifics" added Subchapter "Command strings" added
17	070131	SK	Chapter "Hardware": "Version 5.3": Current limits added "Version 5.4" added Chapter "Setup": "Installating the J-Link USB driver" removed. "Installing the J-Link software and documentation pack" added. Subchapter "List of OEM products" updated. "OS support" updated
16	061222	SK	Chapter "Preface": "Company description" added. J-Link picture changed.
15	060914	00	Subchapter 1.5.1: Added target supply voltage and target supply current to specifications. Subchapter 5.2.1: Pictures of ways to connect J- Trace.
14	060818	ΤQ	Subchapter 4.7 "Using DCC for memory reads" added.
13	060711	00	Subchapter 5.2.2: Corrected JTAG+Trace connector pinout table.
12	060628	00	Subchapter 4.1: Added ARM966E-S to List of supported ARM cores.
11	060607	SK	Subchapter 5.5.2.2 changed. Subchapter 5.5.2.3 added.
10	060526	SK	ARM9 download speed updated. Subchapter 8.2.1: Screenshot "Start sequence" updated. Subchapter 8.2.2 "ID sequence" removed. Chapter "Support" and "FAQ" merged. Various improvements

Revision	Date	Ву	Explanation
9	060324	00	Chapter "Literature and references" added. Chapter "Hardware": Added common information trace signals. Added timing diagram for trace. Chapter "Designing the target board for trace" added.
8	060117	00	Chapter "Related Software": Added JLinkARM.dll. Screenshots updated.
7	051208	00	Chapter Working with J-Link: Sketch added.
6	051118	00	Chapter Working with J-Link: "Connecting multiple J-Links to your PC" added. Chapter Working with J-Link: "Multi core debug- ging" added. Chapter Background information: "J-Link firm- ware" added.
5	051103	ΤQ	Chapter Setup: "JTAG Speed" added.
4	051025	00	Chapter Background information: "Flash program- ming" added. Chapter Setup: "Scan chain configuration" added. Some smaller changes.
3	051021	ΤQ	Performance values updated.
2	051011	ΤQ	Chapter "Working with J-Link" added.
1	050818	ΤW	Initial version.

About this document

This document describes J-Link and J-Trace. It provides an overview over the major features of J-Link and J-Trace, gives you some background information about JTAG, ARM and Tracing in general and describes J-Link and J-Trace related software packages available from Segger. Finally, the chapter *Support and FAQs* on page 257 helps to troubleshoot common problems.

For simplicity, we will refer to J-Link ARM as J-Link in this manual.

For simplicity, we will refer to J-Link ARM Pro as J-Link Pro in this manual.

Typographic conventions

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
Keyword	Text that you enter at the command-prompt or that appears on the display (that is system functions, file- or pathnames).
Reference	Reference to chapters, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.

Table 1.1: Typographic conventions



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

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

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet efficient 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 develops 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)



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 e

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

emUSB USB devic

USB device stack



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

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	Introducti	on	19
	1.1	Requirements	20
	1.2	Supported OS	
	1.3	J-Link / J-Trace models	
	1.3.1	Model comparison	
	1.3.2	J-Link ARM	
	1.3.3	J-Link Ultra	
	1.3.4	J-Link ARM Pro	
	1.3.5	J-Link ARM Lite	
	1.3.6	J-Link Lite Cortex-M	
	1.3.7	J-Trace ARM	
	1.3.8	J-Trace for Cortex-M	
	1.3.9	Flasher ARM	
	1.3.10	J-Link ColdFire	
	1.4	Common features of the J-Link product family	
	1.4	Supported CPU cores	
	1.5	Built-in intelligence for supported CPU-cores	39
	1.6.1	Intelligence for supported CPO-cores	40
	-	Intelligence in the J-Link firmware	
	1.6.2	Intelligence on the PC-side (DLL)	
	1.6.3	Firmware intelligence per model	
	1.7	Supported IDEs	45
2	Licensing		47
	2.1	Introduction	48
	2.2	Software components requiring a license	
	2.2	License types	
	2.3.1	Built-in license	
	2.3.2	Key-based license	
	2.3.3	Device-based license	
	2.3.3	Legal use of SEGGER J-Link software	
	2.4	Use of the software with 3rd party tools	
		Original SEGGER products	54 57
	2.5		
	2.5.1	J-Link	
	2.5.2	J-Link Ultra	
	2.5.3	J-Link Pro	
	2.5.4	J-Trace	
	2.5.5	J-Trace for Cortex-M	
	2.5.6	Flasher ARM	
	2.6	J-Link OEM versions	
	2.6.1	Analog Devices: mIDASLink	
	2.6.2	Atmel: SAM-ICE	
	2.6.3	Digi: JTAG Link	
	2.6.4	IAR: J-Link / J-Link KS	
	2.6.5	IAR: J-Link Lite	
	2.6.6	IAR: J-Trace	
	2.6.7	NXP: J-Link Lite LPC Edition	
	2.6.8	SEGGER: J-Link Lite	
	2.7	J-Link OBs	
	2.8	Illegal Clones	62
2	l-l ink and	d J-Trace related software	63
J			.00

	3.1	J-Link related software	. 64
	3.1.1	J-Link software and documentation package	64
	3.1.2	List of additional software packages	
	-		
	3.2	J-Link software and documentation package in detail	. 66
	3.2.1	J-Link Commander (Command line tool)	. 66
	3.2.2	SWO Analyzer	
	-	SWO Analyzer	. 07
	3.2.3	J-Link STR91x Commander (Command line tool)	. 67
	3.2.4	J-Link STM32 Commander (Command line tool)	. 69
	3.2.5	J-Link TCP/IP Server (Remote J-Link / J-Trace use)	
	3.2.6	J-Mem Memory Viewer	
	3.2.7	J-Flash ARM (Program flash memory via JTAG)	. 72
	3.2.8	J-Link RDI (Remote Debug Interface)	
	3.2.9	J-Link GDB Server	
	3.3	Dedicated flash programming utilities for J-Link	. 75
	3.3.1	Introduction	
	3.3.2	Supported Eval boards	
	3.3.3	Supported flash memories	. 76
	3.3.4	How to use the dedicated flash programming utilities	
	3.3.5	Using the dedicated flash programming utilities for production and commercia	I
	purposes	76	
	3.3.6	F.A.Q	77
	3.4	Additional software packages in detail	
	3.4.1	JTAGLoad (Command line tool)	. 78
	3.4.2	J-Link Software Developer Kit (SDK)	
	-		
	3.4.3	J-Link Flash Software Developer Kit (SDK)	
	3.5	Using the J-LinkARM.dll	. 79
	3.5.1	What is the JLinkARM.dll?	
	3.5.2	Updating the DLL in third-party programs	
	3.5.3	Determining the version of JLinkARM.dll	. 80
	3.5.4	Determining which DLL is used by a program	80
	5.5.1	Determining which DEE is used by a program	. 00
4	Catura		04
4	Setup		.81
4	•		
4	4.1	Installing the J-Link ARM software and documentation pack	. 82
4	•		. 82
4	4.1 4.1.1	Installing the J-Link ARM software and documentation pack	. 82 . 82
4	4.1 4.1.1 4.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface	. 82 . 82 . 85
4	4.1 4.1.1 4.2 4.2.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation	. 82 . 82 . 85 . 85
4	4.1 4.1.1 4.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation	. 82 . 82 . 85 . 85
4	4.1 4.1.1 4.2 4.2.1 4.2.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver	. 82 . 82 . 85 . 85 . 86
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface	. 82 . 82 . 85 . 85 . 86 . 88
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator	. 82 . 82 . 85 . 85 . 86 . 88 . 88
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface	. 82 . 82 . 85 . 85 . 86 . 88 . 88
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface	. 82 . 82 . 85 . 85 . 86 . 88 . 88
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs	. 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator	. 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification	. 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91 . 91
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator	. 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91 . 91
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1	Installing the J-Link ARM software and documentation pack Setup procedure	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91 . 91 . 93 . 93
4	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 88 . 88 . 90 . 91 . 91 . 93 . 93
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.5 4.5.1 4.6 4.6.1 Working v	Installing the J-Link ARM software and documentation pack Setup procedure	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 91 . 93 . 93
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working V 5.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB With J-Link and J-Trace Connecting the target system	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 93 . 93 . 95 . 96
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.5 4.5.1 4.6 4.6.1 Working v	Installing the J-Link ARM software and documentation pack Setup procedure	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 93 . 93 . 95 . 96
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working V 5.1 5.1.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface. Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification. Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Connecting the target system Power-on sequence	. 82 . 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 91 . 91 . 93 . 93 . 95 . 96
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Power-on sequence Verifying target device connection	. 82 . 82 . 85 . 85 . 88 . 88 . 88 . 90 . 91 . 91 . 91 . 93 . 93 . 95 . 96 . 96
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Connecting the target system Power-on sequence Verifying target device connection Problems	. 82 . 82 . 85 . 85 . 88 . 88 . 90 . 91 . 91 . 93 . 93 . 95 . 96 . 96 . 96
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Power-on sequence Verifying target device connection	. 82 . 82 . 85 . 85 . 88 . 88 . 90 . 91 . 91 . 93 . 93 . 95 . 96 . 96 . 96
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2	Installing the J-Link ARM software and documentation pack Setup procedure	. 82 . 82 . 85 . 85 . 88 . 88 . 88 . 90 . 91 . 91 . 93 . 95 . 96 . 96 . 96 . 97
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Connecting the target system Power-on sequence Verifying target device connection Problems Indicators Main indicator	. 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 91 . 93 . 93 . 95 . 96 . 96 . 96 . 97 . 97
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.5.1 4.6 4.6.1 Working V 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Connecting the target system Power-on sequence Verifying target device connection Problems Indicators Main indicator Input indicator	. 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 97 . 97 . 99
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1	Installing the J-Link ARM software and documentation pack Setup procedure Setting up the USB interface Verifying correct driver installation Uninstalling the J-Link USB driver Setting up the IP interface Configuring J-Link using J-Link Configurator Configuring J-Link using the webinterface FAQs J-Link Configurator Configure J-Links using the J-Link Configurator J-Link USB identification Connecting to different J-Links connected to the same host PC via USB with J-Link and J-Trace Connecting the target system Power-on sequence Verifying target device connection Problems Indicators Main indicator	. 82 . 85 . 85 . 86 . 88 . 88 . 90 . 91 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 97 . 97 . 99
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.5.1 4.6 4.6.1 Working V 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2 5.2.3	Installing the J-Link ARM software and documentation pack Setting up the USB interface	. 82 . 82 . 85 . 85 . 88 . 88 . 88 . 90 . 91 . 91 . 91 . 91 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 97 . 97 . 99 . 99
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2 5.2.3 5.3	Installing the J-Link ARM software and documentation pack	. 82 . 82 . 85 . 85 . 88 . 88 . 88 . 90 . 91 . 91 . 91 . 91 . 91 . 93 . 93 . 95 . 96 . 96 . 97 . 97 . 99 . 99 100
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2 5.2.1 5.2.2 5.2.3 5.3 5.3.1	Installing the J-Link ARM software and documentation pack	. 82 . 82 . 85 . 85 . 88 . 88 . 90 . 91 . 91 . 91 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 96 . 97 . 97 . 99 100 100
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2 5.2.3 5.3.1 5.3.1 5.3.2	Installing the J-Link ARM software and documentation pack	. 82 . 82 . 85 . 85 . 88 . 88 . 90 . 91 . 91 . 91 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 96 . 96 . 97 . 99 100 100 100
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2 5.2.1 5.2.2 5.2.3 5.3 5.3.1	Installing the J-Link ARM software and documentation pack	. 82 . 82 . 85 . 85 . 88 . 88 . 90 . 91 . 91 . 91 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 96 . 96 . 97 . 99 100 100 100
	4.1 4.1.1 4.2 4.2.1 4.2.2 4.3 4.3.1 4.3.2 4.4 4.5 4.5.1 4.6 4.6.1 Working v 5.1 5.1.1 5.1.2 5.1.3 5.2 5.2.1 5.2.2 5.2.3 5.3.1 5.3.1 5.3.2	Installing the J-Link ARM software and documentation pack	. 82 . 82 . 85 . 85 . 88 . 88 . 90 . 91 . 93 . 93 . 93 . 93 . 93 . 95 . 96 . 96 . 96 . 97 . 99 100 100 100 100

	5.4	SWD interface	
	5.4.1	SWD speed	
	5.4.2	SWO	
	5.5	Multi-core debugging	
	5.5.1	How multi-core debugging works	
	5.5.2	Using multi-core debugging in detail	
	5.5.3	Things you should be aware of	
	5.6	Connecting multiple J-Links / J-Traces to your PC	
	5.6.1	How does it work?	
	5.7	J-Link control panel	
	5.7.1	Tabs	
	5.8 5.8.1	Reset strategies Strategies for ARM 7/9 devices	
	5.8.2	Strategies for Cortex-M devices	
	5.8.2	Using DCC for memory access	
	5.9.1	What is required?	
	5.9.2	Target DCC handler	
	5.9.3	Target DCC abort handler	
	5.10	J-Link script files	
	5.10.1	Actions that can be customized	
	5.10.2	Script file API functions	
	5.10.3	Global DLL variables	
	5.10.4	Global DLL constants	
	5.10.5	Script file language	
	5.10.6	Script file writing example	
	5.10.7	Executing J-Link script files	
	5.11	Command strings	
	5.11.1	List of available commands	135
	F 1 1 2	Using command strings	141
	5.11.2	Using command strings	
	5.11.2 5.12	Switching off CPU clock during debug	143
	5.12 5.13	Switching off CPU clock during debug Cache handling	143 144
	5.12 5.13 5.13.1	Switching off CPU clock during debug Cache handling Cache coherency	143 144 144
	5.12 5.13 5.13.1 5.13.2	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area	143 144 144 144
	5.12 5.13 5.13.1 5.13.2 5.13.3	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores	143 144 144 144 144
	5.12 5.13 5.13.1 5.13.2	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area	143 144 144 144 144
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores	143 144 144 144 144 144
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores	143 144 144 144 144 144
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores	143 144 144 144 144 144 144 145 145
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Vnload Introduction Licensing	143 144 144 144 144 144 144 145 145 146 147
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices	143 144 144 144 144 144 144 145 145 146 147 148
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash)	143 144 144 144 144 144 144 144 145 146 147 148 149
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench	143 144 144 144 144 144 144 145 145 146 147 148 149 149
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.1 6.4.2	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Introduction Licensing Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK	143 144 144 144 144 144 145 145 146 147 148 149 149 149
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dow 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Vnload Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server	143 144 144 144 144 144 145 145 146 147 148 149 149 149 151
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dow 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.3 6.4.4	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Vnload Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link Commander	143 144 144 144 144 144 145 145 146 147 148 149 149 149 151 152
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.3 6.4.4 6.4.5	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI	143 144 144 144 144 144 145 145 146 147 148 149 149 149 151 152 152
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.3 6.4.4 6.4.5 6.5	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link Commander J-Link RDI Setup for various debuggers (CFI flash)	143 144 144 144 144 144 145 145 146 147 148 149 149 149 151 152 152 153
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5 6.5.1	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link Commander J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK	143 144 144 144 144 144 145 145 146 147 148 149 149 151 152 153 153
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5.1 6.5.1 6.5.2	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server	143 144 144 144 144 144 145 145 146 147 148 149 149 151 152 153 153 154
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dow 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5 6.5.1 6.5.2 6.5.3	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server J-Link GDB Server	143 144 144 144 144 144 145 145 146 147 148 149 149 151 152 153 153 154 154
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5.1 6.5.1 6.5.2	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server	143 144 144 144 144 144 145 145 146 147 148 149 149 151 152 153 153 154 154
	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dow 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5.1 6.5.2 6.5.3 6.6	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server J-Link GDB Server	143 144 144 144 144 144 145 146 147 148 149 149 151 152 153 153 154 154 155
	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dow 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5.1 6.5.2 6.5.3 6.6	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Cache handling of ARM9 cores	143 144 144 144 144 145 146 147 148 149 149 151 152 153 153 154 155 157
	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dow 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5 6.5.1 6.5.2 6.5.3 6.6 Flash bre	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores. Cache handling of ARM9 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server J-Link commander Using the DLL flash loaders in custom applications akpoints	143 144 144 144 144 144 145 146 147 148 149 149 151 152 153 153 154 155 157 158
6	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5 6.5.1 6.5.2 6.5.3 6.6 Flash bre 7.1 7.2 7.2.1	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores Vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server J-Link commander Using the DLL flash loaders in custom applications Introduction	143 144 144 144 144 144 145 146 147 148 149 149 151 152 153 154 155 157 158 159
	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5.1 6.5.2 6.5.3 6.6 Flash bre 7.1 7.2 7.2.1 7.3	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server J-Link commander Using the DLL flash loaders in custom applications akpoints Introduction Licensing Z4h flash breakpoint trial license Supported devices	143 144 144 144 144 144 145 146 147 148 149 149 151 152 153 153 154 155 157 158 159 159 160
	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5 6.5.1 6.5.2 6.5.3 6.6 Flash bre 7.1 7.2 7.2.1 7.3 7.4	Switching off CPU clock during debug	143 144 144 144 144 144 145 146 147 148 149 149 151 152 153 153 154 155 157 158 159 159 160
	5.12 5.13 5.13.1 5.13.2 5.13.3 5.13.4 Flash dov 6.1 6.2 6.3 6.4 6.4.1 6.4.2 6.4.3 6.4.4 6.4.5 6.5 6.5.1 6.5.2 6.5.3 6.6 Flash bre 7.1 7.2 7.2.1 7.3	Switching off CPU clock during debug Cache handling Cache coherency Cache clean area Cache handling of ARM7 cores Cache handling of ARM9 cores Cache handling of ARM9 cores vnload Introduction Licensing Supported devices Setup for various debuggers (internal flash) IAR Embedded Workbench Keil MDK J-Link GDB Server J-Link RDI Setup for various debuggers (CFI flash) IAR Embedded Workbench / Keil MDK J-Link GDB Server J-Link commander Using the DLL flash loaders in custom applications akpoints Introduction Licensing Z4h flash breakpoint trial license Supported devices	143 144 144 144 144 144 145 146 147 149 149 151 152 153 153 154 155 157 158 159 160 161 161

16

	7.5	FAQ	.162
8	RDI		163
	8.1	Introduction	.164
	8.1.1	Features	.164
	8.2	Licensing	
	8.3	Setup for various debuggers	
	8.3.1	IAR Embedded Workbench IDE	
	8.3.2	ARM AXD (ARM Developer Suite, ADS)	
	8.3.3	ARM RVDS (RealView developer suite)	
	8.3.4	GHS MULTI	
	8.3.5	KEIL MDK (µVision IDE)	
	8.4	Configuration	
	8.4.1	Configuration file JLinkRDI.ini	
	8.4.2	Using different configurations	
	8.4.3	Using multiple J-Links simulatenously	
	8.4.4	Configuration dialog	
	8.5	Semihosting	
	8.5.1	Overview	
	8.5.2	The SWI interface	-
	8.5.3	Implementation of semihosting in J-Link RDI	
	8.5.4	Semihosting with AXD.	
	8.5.5	Unexpected / unhandled SWIs	
		•	
9	•	ecifics	
	9.1	Analog Devices	.196
	9.1.1	ADuC7xxx	
	9.2	ATMEL	
	9.2.1	AT91SAM7	.199
	9.2.2	AT91SAM9	.201
	9.3	DSPGroup	
	9.4	Ember	.203
	9.5	Energy Micro	
	9.6	Freescale	.205
	9.6.1	Kinetis family	
	9.6.2	Unlocking	
	9.6.3	Tracing	.206
	9.7	Fujitsu	.207
	9.8	Itron	.208
	9.9	Luminary Micro	.209
	9.9.1	Unlocking LM3Sxxx devices	.210
	9.10	NXP	
	9.10.1	LPC ARM7-based devices	.212
	9.10.2	Reset (Cortex-M3 based devices)	.213
	9.10.3	LPC288x flash programming	.213
	9.11	OKI	.214
	9.12	Renesas	.215
	9.13	Samsung	.216
	9.13.1	S3FN60D	
	9.14	ST Microelectronics	.217
	9.14.1	STR91x	
	9.14.2	STM32F10x	
	9.15	Texas Instruments	
	9.15.1	AM335x	
	9.16	Toshiba	
10	Target ir	iterfaces and adapters	223
	10.1	20-pin JTAG/SWD connector	
	10.1.1	Pinout for JTAG	.224
	10.1.2	Pinout for SWD	

$10.2 \\ 10.2.1 \\ 10.2.2 \\ 10.2.3 \\ 10.2.4 \\ 10.3 \\ 10.3.1 \\ 10.4 \\ 10.5$	 38-pin Mictor JTAG and Trace connector Connecting the target board Pinout Assignment of trace information pins between ETM architecture versions Trace signals 19-pin JTAG/SWD and Trace connector Target power supply 9-pin JTAG/SWD connector Adapters 	229 230 232 232 234 235 236		
11 Backgro	ound information			
$11.1 \\ 11.1.2 \\ 11.1.3 \\ 11.1.4 \\ 11.2 \\ 11.2.1 \\ 11.2.1 \\ 11.2.2 \\ 11.2.3 \\ 11.3 \\ 11.4 \\ 11.4.1 \\ 11.4.2 \\ 11.4.3 \\ 11.4.3 \\ 11.4.4 \\ 11.5 \\ 11.5.1 \\ 11.5.1 \\ 11.5.2 \\ 11$	JTAG Test access port (TAP) Data registers Instruction register The TAP controller Embedded Trace Macrocell (ETM) Trigger condition Code tracing and data tracing J-Trace integration example - IAR Embedded Workbench for ARM Embedded Trace Buffer (ETB) Flash programming How does flash programming via J-Link / J-Trace work? Data download to RAM Data download via DCC Available options for flash programming J-Link / J-Trace firmware Firmware update Invalidating the firmware.	240 240 241 243 243 243 243 243 243 248 248 248 248 248 250 250		
12 Designii	ng the target board for trace	253		
12.1 12.1.1 12.1.2 12.1.3 12.1.4 12.2 12.2.1 12.3	Overview of high-speed board design Avoiding stubs Minimizing Signal Skew (Balancing PCB Track Lengths) Minimizing Crosstalk Using impedance matching and termination Terminating the trace signal Rules for series terminators Signal requirements	254 254 254 254 255 255		
13 Support	and FAQs	257		
13.1 13.1.1 13.2 13.2.1 13.2.2 13.3 13.4	Measuring download speed Test environment Troubleshooting General procedure Typical problem scenarios Contacting support Frequently Asked Questions	258 259 259 259 261		
14 Glossar	14 Glossary			
15 Literatu	re and references	269		

Chapter 1 Introduction

This chapter gives a short overview about J-Link and J-Trace.

1.1 Requirements

Host System

To use J-Link or J-Trace you need a host system running Windows 2000 or later. For a list of all operating systems which are supported by J-Link, please refer to *Supported OS* on page 21.

Target System

A target system with a supported CPU is required.

You should make sure that the emulator you are looking at supports your target CPU. For more information about which J-Link features are supported by each emulator, please refer to *Model comparison* on page 23.

1.2 Supported OS

J-Link/J-Trace can be used on the following operating systems:

- Microsoft Windows 2000
- Microsoft Windows XP
- Microsoft Windows XP x64
- Microsoft Windows Vista
- Microsoft Windows Vista x64
- Windows 7
- Windows 7 x64

1.3 J-Link / J-Trace models

J-Link / J-Trace is available in different variations, each designed for different purposes / target devices. Currently, the following models of J-Link / J-Trace are available:

- J-Link ARM
- J-Link Ultra
- J-Link ARM Pro
- J-Trace ARM
- J-Trace for Cortex-M

In the following, the different J-Link / J-Trace models are described and the changes between the different hardware versions of each model are listed. To determine the hardware version of your J-Link / J-Trace, the first step should be to look at the label at the bottom side of the unit. J-Links / J-Traces have the hardware version printed on the back label.

If this is not the case with your J-Link / J-Trace, start $\tt JLink.exe.$ As part of the initial message, the hardware version is displayed.

🔜 C:\Program Files\SEGGER\JLinkARM_V402d\JLink.exe	_ 🗆 🗙
SEGGER J-Link Commander U4.02d <'?' for help> Compiled Mar 12 2009 15:39:38	_
DLL`version V4.02d, compiled Mar 12 2009 15:39:15 Firmware: J-Link ARM V8 compiled Mar 12 2009 15:28:03	
Hardware: U8.00 S/N : 1	
UTarget = 0.000U JTAG speed: 5 kHz	
J-Link>_	-

1.3.1 Model comparison

The following tables show the features which are included in each J-Link / J-Trace model.

Hardware features

	J-Link	J-Link Pro	J-Trace for Cortex-M	J-Trace
USB	yes	yes	yes	yes
Ethernet	no	yes	no	no
Supported cores	ARM7/9/11, Cortex-A5/A8, Cortex-M0/M1/ M3/M4, Cor- tex-A5/A8/A9/ R4	ARM7/9/11, Cortex-A5/A8, Cortex-M0/M1/ M3/M4, Cor- tex-A5/A8/A9/ R4	ARM 7/9 (no tracing), Cor- tex-M0/M1/ M3/M4	ARM 7/9
JTAG	yes	yes	yes	yes
SWD	yes	yes	yes	no
SWO	yes	yes	yes	no
ETM Trace	no	no	yes	yes

Software features

Software features are features implemented in the software primarily on the host. Software features can either come with the J-Link or be added later using a license string from Segger.

	J-Link	J-Link Pro	J-Trace for Cortex-M	J-Trace
J-Flash	yes(opt)	yes	yes(opt)	yes(opt)
Flash breakpoints ²	yes(opt)	yes	yes(opt)	yes(opt)
Flash download ¹	yes(opt)	yes	yes(opt)	yes(opt)
GDB Server	yes(opt)	yes	yes(opt)	yes(opt)
RDI	yes(opt)	yes	yes(opt)	yes(opt)

¹ Most IDEs come with its own flashloaders, so in most cases this feature is not essential for debugging your applications in flash. The J-Link flash download (FlashDL) feature is mainly used in debug environments where the debugger does not come with an own flashloader (for example, the GNU Debugger). For more information about how flash download via FlashDL works, please refer to *Flash download* on page 145.

² In order to use the flash breakpoints with J-Link no additional license for flash download is required. The flash breakpoint feature allows setting an unlimited number of breakpoints even if the application program is not located in RAM, but in flash memory. Without this feature, the number of breakpoints which can be set in flash is limited to the number of hardware breakpoints (typically two for ARM 7/9, up to six for Cortex-M) For more information about flash breakpoints, please refer to *Flash breakpoints* on page 157.

Introduction

1.3.2 J-Link ARM

J-Link is a JTAG emulator designed for ARM cores. It connects via USB to a PC running Microsoft Windows 2000 or later. For a complete list of all operating systems which are supported, please refer to *Supported OS* on page 21. J-Link has a built-in 20-pin JTAG connector, which is compatible with the standard 20-pin connector defined by ARM.

1.3.2.1 Additional features

- Direct download into flash memory of most popular microcontrollers supported
- Full-speed USB 2.0 interface
- Serial Wire Debug supported *
- Serial Wire Viewer supported *
- Download speed up to 720 KBytes/second **
- JTAG speed up to 12 MHz
- RDI interface available, which allows using J-Link with RDI compliant software
- * = Supported since J-Link hardware version 6

** = Measured with J-Link Rev.5, ARM7 @ 50 MHz, 12MHz JTAG speed.

1.3.2.2 Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for J-Link ARM. All values are valid for J-Link ARM hardware version 8.

General				
Supported OS	For a complete list of all operating sys- tems which are supported, please refer to <i>Supported OS</i> on page 21.			
Electromagnetic compatibility (EMC)	EN 55022, EN 55024			
Operating temperature	+5°C +60°C			
Storage temperature	-20°C +65 °C			
Relative humidity (non-condensing)	Max. 90% rH			
Mechanical				
Size (without cables)	100mm x 53mm x 27mm			
Weight (without cables)	70g			
Available interfaces				
USB interface	USB 2.0, full speed			
Target interface	JTAG 20-pin (14-pin adapter available)			
JTAG/SWD Interface, Electrical				
Power supply	USB powered Max. 50mA + Target Supply current.			
Target interface voltage (V _{IF})	1.2V 5V			
Target supply voltage	4.5V 5V (if powered with 5V on USB)			
Target supply current	Max. 300mA			
Reset Type	Open drain. Can be pulled low or tristated.			
Reset low level output voltage (V _{OL})	$V_{OL} \le 10\%$ of V_{IF}			
For the whole target voltage range (1.2V <= V_{IF} <= 5V)				

Table 1.1: J-Link ARM specifications



LOW level input voltage (V_{IL})	$V_{IL} \le 40\%$ of V_{IF}		
HIGH level input voltage (V_{IH})	$V_{IH} >= 60\%$ of V_{IF}		
For 1.8V <=	• V _{IF} <= 3.6V		
LOW level output voltage (V _{OL}) with a load of 10 kOhm	$V_{OL} \le 10\%$ of V_{IF}		
HIGH level output voltage (V _{OH}) with a load of 10 kOhm	$V_{OH} >= 90\%$ of V_{IF}		
For 3.6 <=	= V _{IF} <= 5V		
LOW level output voltage (V _{OL}) with a load of 10 kOhm	$V_{OL} \le 20\%$ of V_{IF}		
HIGH level output voltage (V _{OH}) with a load of 10 kOhm	$V_{OH} >= 80\%$ of V_{IF}		
JTAG/SWD Interface, Timing			
SWO sampling frequency	Max. 6 MHz		
Data input rise time (T _{rdi})	T _{rdi} <= 20ns		
Data input fall time (T _{fdi})	T _{fdi} <= 20ns		
Data output rise time (T _{rdo})	T _{rdo} <= 10ns		
Data output fall time (T _{fdo})	T _{fdo} <= 10ns		
Clock rise time (T _{rc})	T _{rc} <= 10ns		
Clock fall time (T _{fc})	T _{fc} <= 10ns		
Table 1.1: J-Link ARM specifications			

1.3.2.3 Download speed

The following table lists performance values (Kbytes/s) for writing to memory (RAM):

Hardware	ARM7 via JTAG	ARM9 via JTAG	Cortex-M3 via SWD
J-Link Rev. 6 — 8	720 Kbytes/s	550 Kbytes/s	180 Kbytes/s
$J^{-}LIIIK IKeV. 0 = 0$	(12MHz JTAG)	(12MHz JTAG)	(12 MHz SWD)

Table 1.2: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 258.

The actual speed depends on various factors, such as JTAG/SWD, clock speed, host CPU core etc.

1.3.2.4 Hardware versions

Versions 1-4

Obsolete.

Version 5.0

Identical to version 4.0 with the following exception:

- Uses a 32-bit RISC CPU.
- Maximum download speed (using DCC) is over 700 Kbytes/second.
- JTAG speed: Maximum JTAG frequency is 12 MHz; possible JTAG speeds are: 48 MHz / n, where n is 4, 5, ..., resulting in speeds of:

12.000 MHz (n = 4) 9.600 MHz (n = 5) 8.000 MHz (n = 6) 6.857 MHz (n = 7)

6.000 MHz (n = 8)

```
5.333 MHz (n = 9)
```

- 4.800 MHz (n = 10)
- Supports adaptive clocking.

Version 5.2

Identical to version 5.0 with the following exception:

• Target interface: RESET is open drain

Version 5.3

Identical to version 5.2 with the following exception:

 5V target supply current limited 5V target supply (pin 19) of Kick-Start versions of J-Link is current monitored and limited. J-Link automatically switches off 5V supply in case of over-current to protect both J-Link and host computer. Peak current (<= 10 ms) limit is 1A, operating current limit is 300mA.

Version 5.4

Identical to version 5.3 with the following exception:

• Supports 5V target interfaces.

Version 6.0

Identical to version 5.4 with the following exception:

- Outputs can be tristated (Effectively disabling the JTAG interface)
- Supports SWD interface.
- SWD speed: Software implementation. 4 MHz maximum SWD speed.
- J-Link supports SWV (Speed limited to 500 kHz)

Version 7.0

Identical to version 6.0 with the following exception:

• Uses an additional pin to the UART unit of the target hardware for SWV support (Speed limited to 6 MHz).

Version 8.0

Identical to version 7.0 with the following exception:

• SWD support for non-3.3V targets.

1.3.3 J-Link Ultra

J-Link Ultra is a JTAG/SWD emulator designed for ARM/Cortex and other supported CPUs. It is fully compatible to the standard J-Link and works with the same PC software. Based on the highly optimized and proven J-Link, it offers even higher speed as well as target power measurement capabilities due to the faster CPU, built-in FPGA and High speed USB interface. It connects via USB to a PC running Microsoft Windows 2000 or later. For a complete list of all operating systems which are supported, please refer to Supported OS on page 19.. J-Link Ultra has a built-in 20-pin JTAG/SWD connector.

1.3.3.1 Additional features

- Fully compatible to the standard J-Link
- Very high performance for all supported CPU cores
- Hi-Speed USB 2.0 interface
- JTAG speed up to 25 MHz
- Serial Wire Debug (SWD) supported
- Serial Wire Viewer (SWV) supported
- SWV: UART and Manchester encoding supported
- SWO sampling frequencies up to 25 MHz
- Target power can be supplied
- Target power consumption can be measured with high accuracy. External ADC can be connected via SPI

1.3.3.2 Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for J-Link Ultra. All values are valid for J-Link Ultra hardware version 1.

Note: Some specifications, especially speed, are likely to be improved in the future with newer versions of the J-Link software (freely available).

General				
Supported OS	For a complete list of all operating systems which are supported, please refer to <i>Supported OS</i> on page 21.			
Electromagnetic compatibility (EMC)	EN 55022, EN 55024			
Operating temperature	+5°C +60°C			
Storage temperature	-20°C +65 °C			
Relative humidity (non-condensing)	Max. 90% rH			
Mechanical				
Size (without cables)	100mm x 53mm x 27mm			
Weight (without cables)	73g			
Available interfaces				
USB interface	USB 2.0, Hi-Speed			
Target interface	JTAG/SWD 20-pin			
External (SPI) analog power measure- ment interface	4-pin (Pins 14, 16, 18 and 20 of the 20- pin JTAG/SWD interface)			
JTAG/SWD Inte	rface, Electrical			
Target interface voltage (V _{IF})	1.8V 5V			
Target supply voltage	4.5V 5V			
Target supply current	Max. 300mA			
Reset Type	Open drain. Can be pulled low or tristated.			

Table 1.3: J-Link Ultra specifications



Reset low level output voltage (V_{OL})	$V_{OL} \le 10\%$ of V_{IF}			
For the whole target voltage range (1.8V <= V_{IF} <= 5V)				
LOW level input voltage (V_{IL})	$V_{IL} \le 40\%$ of V_{IF}			
HIGH level input voltage (V _{IH})	$V_{IH} >= 60\%$ of V_{IF}			
For 1.8V <= V _{IF} <= 3.6V				
LOW level output voltage (V _{OL}) with a load of 10 kOhm	V_{OL} <= 10% of V_{IF}			
HIGH level output voltage (V _{OH}) with a load of 10 kOhm	$V_{OH} >= 90\%$ of V_{IF}			
For 3.6 <	= V _{IF} <= 5V			
LOW level output voltage (V _{OL}) with a load of 10 kOhm	$V_{OL} \le 20\%$ of V_{IF}			
HIGH level output voltage (V _{OH}) with a load of 10 kOhm	$V_{OH} >= 80\%$ of V_{IF}			
JTAG/SWD Interface, Timing				
SWO sampling frequency	Max. 25 MHz			
Data input rise time (T _{rdi})	T _{rdi} <= 20ns			
Data input fall time (T _{fdi})	T _{fdi} <= 20ns			
Data output rise time (T _{rdo})	T _{rdo} <= 10ns			
Data output fall time (T _{fdo})	T _{fdo} <= 10ns			
Clock rise time (T _{rc})	T _{rc} <= 10ns			
Clock fall time (T _{fc})	T _{fc} <= 10ns			
Analog power measurement interface				
Sampling frequency	50 kHz			
Resolution	1 mA			
External (SPI) analog interface				
SPI frequency	Max. 4 MHz			
Samples/sec	Max. 50000			
Resolution	Max. 16-bit			

Table 1.3: J-Link Ultra specifications

1.3.4 J-Link ARM Pro

J-Link Pro is a JTAG emulator designed for ARM cores. It is fully compatible to J-Link and connects via Ethernet/USB to a PC running Microsoft Windows 2000 or later. For a complete list of all operating systems which are supported, please refer to Supported OS on page 19. Additional support for Cortex-R4 and Cortex-R8 cores will be available in the near future. J-Link Pro comes with licenses for all J-Link related SEGGER software products which allows using J-Link Pro "out-of-the-box".

1.3.4.1 Additional features

- Fully compatible to J-Link ARM
- More memory for future firmware extensions (ARM11, X-Scale, Cortex R4 and Cortex A8)
- Additional LEDs for power and RESET indication
- Comes with web interface for easy TCP/IP configuration (built-in web server)
- Built-in GDB Server (planned to be implemented in the near future)
- Serial Wire Debug supported



- Serial Wire Viewer supported
- Download speed up to 720 KBytes/second ** (higher download speeds will be available in the near future)
- DCC speed up to 800 Kbytes/second **
- Comes with licenses for: J-Link ARM RDI, J-Link ARM FlashBP, J-Link ARM FlashDL, J-Link ARM GDB Server and J-Flash ARM.
- Embedded Trace Buffer (ETB) support
- Galvanic isolation from host via Ethernet
- RDI interface available, which allows using J-Link with RDI compliant software
- ** = Measured with J-Link Pro Rev. 1.1, ARM7 @ 50 MHz, 12MHz JTAG speed.

1.3.4.2 Download speed

The following table lists performance values (Kbytes/s) for writing to memory (RAM):

Hardware	ARM7	ARM9	Cortex-M3
	via JTAG	via JTAG	via SWD
Rev. 1 via USB	720 Kbytes/s	550 Kbytes/s	190 Kbytes/s
	(12 MHz JTAG)	(12 MHz JTAG)	(12 MHz SWD)
Rev. 1 via TCP/IP	720 Kbytes/s	550 Kbytes/s	190 Kbytes
	(12 MHz JTAG)	(12 MHz JTAG)	(12 MHz SWD)

Table 1.4: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 258.

The actual speed depends on various factors, such as JTAG/SWD, clock speed, host CPU core etc.

1.3.4.3 Hardware versions

Version 1.1

Compatible to J-Link ARM.

 Provides an additional Ethernet interface which allows to communicate with J-Link via TCP/IP.

1.3.5 J-Link ARM Lite

J-Link ARM Lite is a fully functional OEM-version of J-Link ARM. If you are selling evaluation-boards, J-Link ARM Lite is an inexpensive emulator solution for you. Your customer receives a widely acknowledged JTAG-emulator which allows him to start right away with his development.



1.3.5.1 Additional features

- Very small form factor
- Fully software compatible to J-Link ARM
- Any ARM7/9/11, Cortex-A5/A8, Cortex-M0/M1/M3/M4, Cortex-R4 core supported
- JTAG clock up to 4 MHz
- SWD, SWO supported for Cortex-M devices
- Flash download into supported MCUs
- Standard 20-pin 0.1 inch JTAG connector (compatible to J-Link ARM)

1.3.5.2 Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for J-Link ARM Lite. All values are valid for J-Link ARM hardware version 8.

General				
Supported OS	For a complete list of all operating sys- tems which are supported, please refer to <i>Supported OS</i> on page 21.			
Electromagnetic compatibility (EMC)	EN 55022, EN 55024			
Operating temperature	+5°C +60°C			
Storage temperature	-20°C +65 °C			
Relative humidity (non-condensing)	Max. 90% rH			
Size (without cables)	28mm x 26mm x 7mm			
Weight (without cables)	6g			
Mech	anical			
USB interface	USB 2.0, full speed			
Target interface	JTAG 20-pin (14-pin adapter available)			
JTAG/SWD Interface, Electrical				
Power supply	USB powered Max. 50mA + Target Supply current.			
Target interface voltage (V _{IF})	3.3V			
Target supply voltage	4.5V 5V (if powered with 5V on USB)			
Target supply current	Max. 300mA			
LOW level input voltage (V _{IL})	Max. 40% of V _{IF}			
HIGH level input voltage (V _{IH})	Min. 60% of V _{IF}			
JTAG/SWD Int	terface, Timing			
Data input rise time (T _{rdi})	Max. 20ns			
Data input fall time (T _{fdi})	Max. 20ns			
Data output rise time (T _{rdo})	Max. 10ns			
Data output fall time (T _{fdo})	Max. 10ns			
Clock rise time (T _{rc})	Max. 10ns			
Clock fall time (T _{fc})	Max. 10ns			

Table 1.5: J-Link ARM Lite specifications

1.3.6 J-Link Lite Cortex-M

J-Link Lite Cortex-M is a specific OEM-version of SEGGER J-Link Lite which is designed to be used with Cortex-M devices. If you are selling evaluation-boards, J-Link Lite CortexM is an inexpensive emulator solution for you. Your customer receives a



widely acknowledged JTAG/SWD-emulator which allows him to start right away with his development.

- Very small form factor
- Fully software compatible to J-Link
- Any Cortex-M0/M1/M3/M4 core supported
- JTAG clock up to 4 MHz
- SWD, SWO supported
- Flash download into supported MCUs
- Standard 9- & 19-pin 0.05" Samtec FTSH connector
- 3.3V target interface voltage

1.3.6.1 Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for J-Link Lite Cortex-M.

General			
Supported OS	For a complete list of all operating sys- tems which are supported, please refer to <i>Supported OS</i> on page 21.		
Electromagnetic compatibility (EMC)	EN 55022, EN 55024		
Operating temperature	+5°C +60°C		
Storage temperature	-20°C +65 °C		
Relative humidity (non-condensing)	Max. 90% rH		
Size (without cables)	41mm x 34mm x 8mm		
Weight (without cables)	6g		
Mechanical			
USB interface	USB 2.0, full speed		
Target interface	19-pin 0.05" Samtec FTSH connector 9-pin 0.05" Samtec FTSH connector		
JTAG/SWD Inte	JTAG/SWD Interface, Electrical		
Power supply	USB powered Max. 50mA + Target Supply current.		
Target interface voltage (V _{IF})	3.3V		
Target supply voltage	4.5V 5V		
Target supply current	Max. 300mA		
LOW level input voltage (V_{IL})	Max. 40% of V_{IF}		
HIGH level input voltage (V_{IH})	Min. 60% of V _{IF}		
JTAG/SWD Interface, Timing			
Data input rise time (T _{rdi})	Max. 20ns		
Data input fall time (T _{fdi})	Max. 20ns		
Data output rise time (T _{rdo})	Max. 10ns		
Data output fall time (T _{fdo})	Max. 10ns		
Clock rise time (T _{rc})	Max. 10ns		
Clock fall time (T _{fc})	Max. 10ns		

Table 1.6: J-Link Lite Cortex-M specifications

Introduction

1.3.7 J-Trace ARM

J-Trace is a JTAG emulator designed for ARM cores which includes trace (ETM) support. It connects via USB to a PC running Microsoft Windows 2000 or later. For a complete list of all operating systems which are supported, please refer to Supported OS on page 19. J-Trace has a built-in 20-pin JTAG connector and a built in 38-pin JTAG+Trace connector, which are compatible to the standard 20-pin connector and 38-pin connector defined by ARM.

1.3.7.1 Additional features

- Supports tracing on ARM7/9 targets
- JTAG speed up to 12 MHz
- Download speed up to 420 Kbytes/second *
- DCC speed up to 600 Kbytes/second *

 * = Measured with J-Trace, ARM7 @ 50 MHz, 12MHz JTAG speed.

1.3.7.2 Specifications for J-Trace

General		
Supported OS	For a complete list of all operating sys- tems which are supported, please refer to <i>Supported OS</i> on page 21.	
Electromagnetic Compatibility (EMC)	EN 55022, EN 55024	
Operating Temperature	+5°C +40°C	
Storage Temperature	-20°C +65 °C	
Relative Humidity (non-condensing)	<90% rH	
Size (without cables)	123mm x 68mm x 30mm	
Weight (without cables)	120g	
Mechanical		
USB Interface	USB 2.0, full speed	
Target Interface	JTAG 20-pin (14-pin adapter available) JTAG+Trace: Mictor, 38-pin	
JTAG/SWD Interface, Electrical		
Power Supply	USB powered < 300mA	
Supported Target interface voltage	3.0 - 3.6 V (5V adapter available)	
Table 1 7: 1-Trace specifications		

Table 1.7: J-Trace specifications

1.3.7.3 Download speed

The following table lists performance values (Kbytes/s) for writing to memory (RAM):

Hardware	ARM7 via JTAG	ARM9 via JTAG
J-Trace Rev. 1	420.0 Kbytes/s (12MHz JTAG)	280.0 Kbytes/s (12MHz JTAG)

 Table 1.8: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 258.

The actual speed depends on various factors, such as JTAG, clock speed, host CPU core etc.

1.3.7.4 Hardware versions

Version 1

This J-Trace uses a 32-bit RISC CPU. Maximum download speed is approximately 420 KBytes/second (600 KBytes/second using DCC).

1.3.8 J-Trace for Cortex-M

J-Trace for Cortex-M is a JTAG/SWD emulator designed for Cortex-M cores which includes trace (ETM) support. J-Trace for Cortex-M can also be used as a J-Link and it also supports ARM7/9 cores. Tracing on ARM7/9 targets is not supported.

1.3.8.1 Additional features

- Has all the J-Link functionality
- Supports tracing on Cortex-M targets

1.3.8.2 Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for J-Trace for Cortex-M. All values are valid for the latest hardware version of J-Trace for Cortex-M.

General		
Supported OS	For a complete list of all operating sys- tems which are supported, please refer to Supported OS on page 19.	
Electromagnetic compatibility (EMC)	EN 55022, EN 55024	
Operating temperature	+5°C +60°C	
Storage temperature	-20°C +65 °C	
Relative humidity (non-condensing)	Max. 90% rH	
Size (without cables)	123mm x 68mm x 30mm	
Weight (without cables)	120g	
Mechanical		
USB interface	USB 2.0, Hi-Speed	
Target interface	JTAG/SWD 20-pin (14-pin adapter available) JTAG/SWD + Trace 19-pin	
JTAG/SWD Inte	rface, Electrical	
Power supply	USB powered Max. 50mA + Target Supply current.	
Target interface voltage (V _{IF})	1.2V 5V	
Target supply voltage	4.5V 5V (if powered with 5V on USB)	
Target supply current	Max. 300mA	
LOW level input voltage (V_{IL})	Max. 40% of V _{IF}	
HIGH level input voltage (V_{IH})	Min. 60% of V _{IF}	
JTAG/SWD Interface, Timing		
Data input rise time (T _{rdi})	Max. 20ns	
Data input fall time (T _{fdi})	Max. 20ns	
Data output rise time (T _{rdo})	Max. 10ns	
Data output fall time (T _{fdo})	Max. 10ns	
Clock rise time (T _{rc})	Max. 10ns	

Table 1.9: J-Trace for Cortex-M3 specifications

34



Clock fall time (T _{fc})	Max. 10ns	
Trace Interface, Electrical		
Power supply	USB powered Max. 50mA + Target Supply current.	
Target interface voltage (V _{IF})	1.2V 5V	
Voltage interface low pulse (V_{IL})	Max. 40% of $V_{\rm IF}$	
Voltage interface high pulse (V_{IH})	Min. 60% of V _{IF}	
Trace Interface, Timing		
TRACECLK low pulse width (T _{wl})	Min. 2ns	
TRACECLK high pulse width (T_{wh})	Min. 2ns	
Data rise time (T _{rd})	Max. 3ns	
Data fall time (T _{fd})	Max. 3ns	
Clock rise time (T _{rc})	Max. 3ns	
Clock fall time (T _{fc})	Max. 3ns	
Data setup time (T _s)	Min. 3ns	
Data hold time (T _h)	Min. 2ns	
color 1 0: 1 Types for Contex M2 encodimentions		

Table 1.9: J-Trace for Cortex-M3 specifications

1.3.8.3 Download speed

The following table lists performance values (Kbytes/s) for writing to memory (RAM):

Hardware	Cortex-M3
J-Trace for Cortex-M3 V2	190 Kbytes/s (12MHz SWD) 760 KB/s (12 MHz JTAG)
J-Trace for Cortex-M V3.1	190 Kbytes/s (12MHz SWD) 1440 KB/s (25 MHz JTAG)

 Table 1.10: Download speed differences between hardware revisions

The actual speed depends on various factors, such as JTAG, clock speed, host CPU core etc.

1.3.8.4 Hardware versions

Version 2

Obsolete.

Version 3.1

Identical to version 2.0 with the following exceptions:

- Hi-Speed USB
- Voltage range for trace signals extended to 1.2 3.3 V
- Higher download speed

Flasher

1.3.9 Flasher ARM

Flasher ARM is a programming tool for microcontrollers with onchip or external Flash memory and ARM core. Flasher ARM is designed for programming flash targets with the J-Flash software or stand-alone. In addition to that Flasher ARM has all of the J-Link functionality. For more information about Flasher ARM, please refer to UM08007, Flasher ARM User's Guide.

1.3.9.1 Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for Flasher ARM.

Ge	neral		
Supported OS	For a complete list of all operating sys- tems which are supported, please refer to Supported OS on page 19.		
Mechanical			
USB interface	USB 2.0, full speed		
Target interface	JTAG/SWD 20-pin		
JTAG Interface, Electrical			
Power supply	USB powered Max. 50mA + Target Supply current.		
Target interface voltage (V_{IF})	1.2V 5V		
Target supply voltage	4.5V 5V (if powered with 5V on USB)		
Target supply current	Max. 300mA		
For the whole target voltage range (1.8V <= V _{IF} <= 5V)			
LOW level input voltage (V_{IL})	Max. 40% of V _{IF}		
HIGH level input voltage (V _{IH})	Min. 60% of V _{IF}		
For 1.8V <= V _{IF} <= 3.6V			
LOW level output voltage (V_{OL}) with a load of 10 kOhm	Max. 10% of $V_{\rm IF}$		
HIGH level output voltage (V _{OH}) with a load of 10 kOhm	Min. 90% of V _{IF}		
For 3.6 <= V _{IF} <= 5V			
LOW level output voltage (V _{OL}) with a load of 10 kOhm	Max. 20% of V _{IF}		
HIGH level output voltage (V _{OH}) with a load of 10 kOhm	Min. 80% of V _{IF}		
SWD Interface, Electrical			
Power supply	USB powered Max. 50mA + Target Supply current.		
Target interface voltage (V_{IF})	1.2V 5V (SWD interface is 5V tolerant but can output a maximum of 3.3V SWD signals)		
Target supply voltage	4.5V 5V (if powered with 5V on USB)		

Table 1.11: Flasher ARM specifications

Target supply current	Max. 300mA
LOW level input voltage (V_{IL})	Max. 0.8V
HIGH level input voltage (V_{IH})	Min. 2.0V
LOW level output voltage (V _{OL}) with a	Max. 0.5V
load of 10 kOhm	
HIGH level output voltage (V_{OH}) with a	Min. 2.85V
load of 10 kOhm	

Table 1.11: Flasher ARM specifications

1.3.10 J-Link ColdFire

J-Link ColdFire is a BDM emulator designed for ColdFire® cores. It connects via USB to a PC running Microsoft Windows 2000, Windows XP, Windows 2003, or Windows Vista. J-Link ColdFire has a built-in 26-pin BDM connector, which is compatible to the standard 26-pin connector defined by Freescale. For more information about J-Link ColdFire BDM 26, please refer to UM08009, J-Link ColdFire BDM26 User's Guide.



1.4 Common features of the J-Link product family

- USB 2.0 interface (Full-Speed/Hi-Speed, depends on J-Link model)
- Any ARM7/9/11 (including thumb mode), Cortex-A5/A8, Cortex-M0/M1/M3/M4, Cortex-R4 core supported
- Automatic core recognition
- Maximum JTAG speed 12/25 MHz (depends on J-Link model)
- Seamless integration into the IAR Embedded Workbench® IDE
- No power supply required, powered through USB
- Support for adaptive clocking
- All JTAG signals can be monitored, target voltage can be measured
- Support for multiple devices
- Fully plug and play compatible
- Standard 20-pin JTAG/SWD connector, 19-pin JTAG/SWD and Trace connector, standard 38-pin JTAG+Trace connector
- USB and 20-pin ribbon cable included
- Memory viewer (J-Mem) included
- TCP/IP server included, which allows using J-Trace via TCP/IP networks
- RDI interface available, which allows using J-Link with RDI compliant software
- Flash programming software (J-Flash) available
- Flash DLL available, which allows using flash functionality in custom applications
- Software Developer Kit (SDK) available
- Full integration with the IAR C-SPY® debugger; advanced debugging features available from IAR C-SPY debugger.
- 14-pin JTAG adapter available
- J-Link 19-pin Cortex-M Adapter available
- J-Link 9-pin Cortex-M Adapter available
- Adapter for 5V JTAG targets available for hardware revisions up to 5.3
- Optical isolation adapter for JTAG/SWD interface available
- Target power supply via pin 19 of the JTAG/SWD interface (up to 300 mA to target with overload protection), alternatively on pins 11 and 13 of the Cortex-M 19-pin trace connector

1.5 Supported CPU cores

J-Link / J-Trace has been tested with the following cores, but should work with any ARM7/9/11, Cortex-M0/M1/M3/M4 and Cortex-A5/A8/A9/R4 core. If you experience problems with a particular core, do not hesitate to contact Segger.

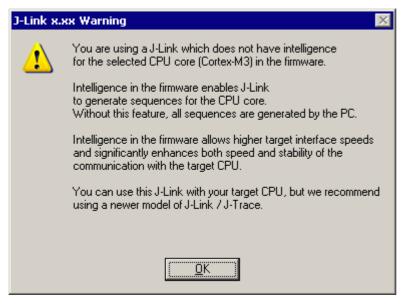
- ARM7TDMI (Rev 1)
- ARM7TDMI (Rev 3)
- ARM7TDMI-S (Rev 4)
- ARM720T
- ARM920T
- ARM922T
- ARM926EJ-S
- ARM946E-S
- ARM966E-S
- ARM1136JF-S
- ARM1136J-S
- ARM1156T2-S
- ARM1156T2F-S
- ARM1176JZ-S
- ARM1176JZF
- ARM1176JZF-S
- Cortex-A5
- Cortex-A8
- Cortex-A9
- Cortex-M0
- Cortex-M1
- Cortex-M3
- Cortex-M4
- Cortex-R4

1.6 Built-in intelligence for supported CPU-cores

In general, there are two ways two ways to support a CPU-core in the J-Link software:

- 1. Intelligence in the J-Link firmware
- 2. Intelligence on the PC-side (DLL)

Having the intelligence in the firmware is ideal since it is much more powerful and robust. The J-Link PC software automatically detects which implementation level is supported for the connected CPU-core. If Intelligence in the firmware is available, it is used. If you are using a J-Link that does not have intelligence in the firmware and only PC-side intelligence is available for the connected CPU, a warning message is shown.



1.6.1 Intelligence in the J-Link firmware

On newer J-Links, the intelligence for a new CPU-core is also available in the J-Link firmware which means, for these J-Links the target sequences are no longer generated on the PC-side but directly inside the J-Link. Having the intelligence in the firmware leads to improved stability and higher performance.

1.6.2 Intelligence on the PC-side (DLL)

This is the basic implementation level for support of a CPU-core. This implementation is not J-Link model dependend, since no intelligence for the CPU-core is necessary in the J-Link firmware. This means, all target sequences (JTAG/SWD/...) are generated on the PC-side and the J-Link simply sends out these sequences and sends the result back to the DLL. Using this way of implementation also allows old J-Links to be used with new CPU cores as long as a DLL-Version is used which has intelligence for the CPU.

But there is one big disadvantage of implementing the CPU core support on the DLLside: For every sequence which shall be send to the target a USB or Ethernet transaction is triggered. The long latency especially on a USB connection significantly affects the performance of J-Link. This is true especially, when performing actions where J-Link has to wait for the CPU frequently. An example is a memory read/write operation which needs to be followed by status read operations or repeated until the memory operation is completed. Performing this kind of task with only PC-side intelligence will have to either make some assumption like: Operation is completed after a given number of cycles or will have to make a lot of USB/Ethernet transactions. The first option (fast mode) will not work under some circumstances such as low CPU speeds, the second (slow mode) will be more reliable but very slow due to the high number of USB/Ethernet transactions. It simply boils down to: The best solution is having intelligence in the emulator itself!

1.6.2.1 Limitations of PC-side implementations

• Instability, especially on slow targets

Due to the fact that a lot of USB transactions would cause a very bad performance of J-Link, on PC-side implementations the assumption is made that the CPU/Debug interface is fast enough to handle the commands/requests without the need of waiting. So, when using the PC-side-intelligence, stability can not be guaranteed in all cases, especially if the target interface speed (JTAG/SWD/...) is significantly higher than the CPU speed.

• Poor performance

Since a lot more data has to be transferred over the host interface (typ. USB), the resulting download speed is typically much lower than for implementations with intelligence in the firmware, even if the number of transactions over the host interface is limited to a minimum (fast mode).

• No support

Please understand that we can not give any support if you are running into problems when using a PC-side implementation.

Note: Due to these limitations, we recommend to use PC-side implementations for evaluation only.

1.6.3 Firmware intelligence per model

There are different models of J-Link / J-Trace which have built-in intelligence for different CPU-cores. In the following, we will give you an overview about which model of J-Link / J-Trace has intelligence for which CPU-core.

1.6.3.1 Current models

The table below lists the firmware CPU support for J-Link & J-Trace models currently available.

J-Link / J-Trace model	Version	ARM 7/9	ARM 11	Cortex- A/R	Cort	ex-M	Renesas RX600
		JTAG	JTAG	JTAG	JTAG	SWD	JTAG
J-Link	8						
J-Link Pro	3	\bigcirc		Ø			I
J-Link Ultra	1			S			
J-Link Lite	8		I	0		I	
J-Link Lite Cortex-M	8	×	⊗	\bigotimes		I	⊗
J-Link Lite RX	8	⊗	⊗	8	×	×	
J-Trace ARM	1	(⊗	⊗	╳	×	⊗
J-Trace for Cortex-M	3	⊗	⊗	8			♦

Table 1.12: Built-in intelligence of current J-Links

1.6.3.2 Older models

The table below lists the firmware CPU support for older J-Link & J-Trace models which are not sold anymore.

J-Link / J-Trace model	Version	ARM 7/9	ARM 11	Cortex- A/R	Cor	tex-M	Renesas RX600
		JTAG	JTAG	JTAG	JTAG	SWD	JTAG
J-Link	3	⊗	⊗	⊗	8	not sup- ported	⊗
J-Link	4	⊗	⊗	⊗	8	not sup- ported	\bigotimes
J-Link	5	(×	8	8	not sup- ported	\bigotimes
J-Link	6	()	×	X	×	0	\bigotimes
J-Link	7		×	X	×	0	\bigotimes
J-Link Pro	1			0	♦	0	
J-Trace for Cortex-M	1	⊗	⊗	(۲	0	\bigotimes

Table 1.13: Built-in intelligence of older J-Link models

1.7 Supported IDEs

J-Link / J-Trace can be used with different IDEs. Some IDEs support J-Link directly, for other ones additional software (such as J-Link RDI) is necessary in order to use J-Link. The following tables list which features of J-Link / J-Trace can be used with the different IDEs.

ARM7/9

IDE	Debug support ⁴	Flash download	Flash breakpoints	Trace support ³
IAR EWARM	yes	yes	yes	yes
Keil MDK	yes	yes	yes	no
Rowley	yes	yes	no	no
CodeSourcery	yes	no	no	no
Yargato (GDB)	yes	yes	yes	no
RDI compliant toolchains such as RVDS/ADS	yes ¹	yes ¹	yes ¹	no

ARM Cortex-M3

IDE	Debug support ⁴	Flash download	Flash breakpoints	Trace support ³	SWO support
IAR EWARM	yes	yes	yes	yes	yes
Keil MDK	yes	yes	yes	yes	yes
Rowley	yes	yes	no	no	no
CodeSourcery	yes	no	no	no	no
Yargato (GDB)	yes	yes	yes	no	no

ARM11

ARM11 has currently been tested with IAR EWARM only.

IDE	Debug support ⁴	Flash download	Flash breakpoints	Trace support ³
IAR EWARM	yes	no ²	no ²	no
Rowley	yes	no ²	no	no
Yargato (GDB)	yes	no ²	no ²	no

- ¹ Requires J-Link RDI license for download of more than 32KBytes
- ² Coming soon
- ³ Requires emulator with trace support
- ⁴ Debug support includes the following: Download to RAM, memory read/write, CPU register read/write, Run control (go, step, halt), software breakpoints in RAM and hardware breakpoints in flash memory.

CHAPTER 1

Chapter 2 Licensing

This chapter describes the different license types of J-Link related software and the legal use of the J-Link software with original SEGGER and OEM products.

2.1 Introduction

J-Link functionality can be enhanced by the features J-Flash, RDI, flash download and flash breakpoints (FlashBP). The flash breakpoint feature does not come with J-Link and need an additional license. In the following the licensing options of the software will be explained.

2.2 Software components requiring a license

There are different software components which need an additional license:

- J-Flash
- J-Link RDI
- Flash breakpoints (FlashBP)

For more information about J-Link RDI licensing procedure / license types, please refer to the *J-Link RDI User Guide* (UM08004), chapter *Licensing*.

For more information about J-Flash licensing procedure / license types, please refer to the *J-Flash User Guide* (UM08003), chapter *Licensing*.

In the following the licensing procedure and license types of the flash breakpoint feature are explained.

2.3 License types

For each of the software components which require an additional license, there are three types of licenses:

Built-in License

This type of license is easiest to use. The customer does not need to deal with a license key. The software automatically finds out that the connected J-Link contains the built-in license(s). This is the type of license you get if you order J-Link and the license at the same time, typically in a bundle.

Key-based license

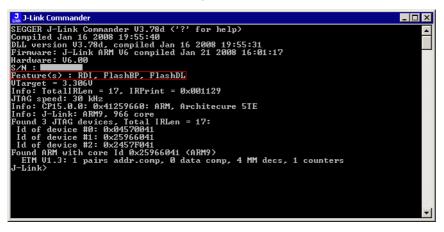
This type of license is used if you already have a J-Link, but want to enhance its functionality by using flash breakpoints. In addition to that, the key-based license is used for trial licenses. To enable this type of license you need to obtain a license key from SEGGER. Free trial licenses are available upon request from *www.segger.com*. This license key has to be added to the J-Link license management. How to enter a license key is described in detail in *Licensing* on page 159. Every license can be used on different PCs, but only with the J-Link the license is for. This means that if you want to use flash breakpoints with other J-Links, every J-Link needs a license.

Device-based license

The device-based license comes with the J-Link software and is available for some devices. For a complete list of devices which have built-in licenses, please refer to *Device list* on page 52. The device-based license has to be activated via the debugger. How to activate a device-based license is described in detail in the section *Activating a device-based license* on page 52.

2.3.1 Built-in license

This type of license is easiest to use. The customer does not need to deal with a license key. The software automatically finds out that the connected J-Link contains the built-in license(s). To check what licenses the used J-Link have, simply open the J-Link commander (JLink.exe). The J-Link commander finds and lists all of the J-Link's licenses automatically, as can be seen in the screenshot below.



This J-Link for example, has built-in licenses for RDI, <code>J-Link ARM FlashDL</code> and <code>FlashBP</code>.

2.3.2 Key-based license

When using a key-based license, a license key is required in order to enable the J-Link flash breakpoint feature. License keys can be added via the license manager. How to enter a license via the license manager is described in *Licensing* on page 159. Like the built-in license, the key-based license is only valid for one J-Link, so if another J-Link is used it needs a separate license.

2.3.2.1 Entering a key-based license

The easiest way to enter a license is the following:

Open the J-Link control panel window, go to the **General** tab and choose License.



Now the J-Link license manager will open and show all licenses, both key-based and built-in licenses of J-Link.

3-	Link ARM Licen	se management	×
	Licenses installed	on PC:	
	Serial number	Feature	Expires
	Licenses in emula	tor:	
	Serial number	Features	
	Currently active lic	censes	
	Add license	Delete license	ОК
		Elenere molennee	- OK

Now choose **Add license** to add one or more new licenses. Enter your license(s) and choose **OK**. Now the licenses should have been added.

Serial number	Feature	Expires	
	FlashBP FlashDL	Never Never	
icenses in emul	ator:		
Serial number	Features		
Currently active li	censes FlashBP, FlashDL		

2.3.3 Device-based license

The device-based license is a free license, available for some devices. It's already included in J-Link, so no keys are necessary to enable this license type. To activate a device based license, the debugger needs to select a supported device.

2.3.3.1 Activating a device-based license

In order to activate a device-based license, the debugger needs to select a supported device. To check if the debugger has selected the right device, simply open the J-Link control panel and check the **device** section in the **General** tab.



2.3.3.2 Device list

The following list contains all devices which are supported by the device-based license $% \left({{{\left[{{{\left[{{\left[{{\left[{{\left[{{{\left[{{{\left[{{{\left[{{{\left[{{{\left[{{{\left[{{{\left[{{{\left[{{{}}}} \right]}}} \right.}$

Manufacturer	Name	Licenses
NXP	LPC2101	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2102	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2103	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2104	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2105	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2106,	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2109	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2114	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2119	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2124	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2129	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2131	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2132	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2134	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2136	J-Link ARM FlashDL, J-Link ARM FlashBP

Table 2.1: Device list

Manufacturer	Name	Licenses
NXP	LPC2138	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2141	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2142	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2144	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2146	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2148	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2194	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2212	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2214	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2292	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2294	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2364	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2366	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2368	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2378	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2468	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2478	J-Link ARM FlashDL, J-Link ARM FlashBP

Table 2.1: Device list

2.4 Legal use of SEGGER J-Link software

The software consists of proprietary programs of SEGGER, protected under copyright and trade secret laws. All rights, title and interest in the software are and shall remain with SEGGER. For details, please refer to the license agreement which needs to be accepted when installing the software. The text of the license agreement is also available as entry in the start menu after installing the software.

Use of software

SEGGER J-Link software may only be used with original SEGGER products and authorized OEM products. The use of the licensed software to operate SEGGER product clones is prohibited and illegal.

2.4.1 Use of the software with 3rd party tools

For simplicity, some components of the J-Link software are also distributed from partners with software tools designed to use J-Link. These tools are primarily debugging tools, but also memory viewers, flash programming utilities but also software for other purposes. Distribution of the software components is legal for our partners, but the same rules as described above apply for their usage: They may only be used with original SEGGER products and authorized OEM products. The use of the licensed software to operate SEGGER product clones is prohibited and illegal.

2.5 Original SEGGER products

The following products are original SEGGER products for which the use of the J-Link software is allowed:

2.5.1 J-Link

J-Link is a JTAG emulator designed for ARM cores. It connects via USB to a PC running Microsoft Windows 2000, Windows XP, Windows 2003, Windows Vista or Windows 7. J-Link has a builtin 20-pin JTAG connector, which is compatible with the standard 20-pin connector defined by ARM.

Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 52.



2.5.2 J-Link Ultra



J-Link Ultra is a JTAG/SWD emulator designed for ARM/Cortex and other supported CPUs. It is fully compatible to the standard J-Link and works with the same PC software. Based on the highly optimized and proven J-Link, it offers even higher speed as well as target power measurement capabilities due to the faster CPU, built-in FPGA and High speed USB interface. It connects via USB to a PC running Microsoft Windows 2000, Windows XP, Windows 2003, Windows Vista or Windows 7. J-Link Ultra has a built-in 20-pin JTAG/SWD connector.

Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 52.

2.5.3 J-Link Pro

J-Link Pro is a JTAG emulator designed for ARM cores. It connects via USB or Ethernet to a PC running Microsoft Windows 2000, Windows XP, Windows 2003, Windows Vista or Windows 7. J-Link has a built-in 20-pin JTAG connector, which is compatible with the standard 20-pin connector defined by ARM.

Licenses

Comes with built-in licenses for all J-Link related software products: J-Link ARM FlashDL, FlashBP, RDI, J-Link GDB Server and J-Flash.

2.5.4 J-Trace

J-Trace is a JTAG emulator designed for ARM cores which includes trace (ETM) support. It connects via USB to a PC running Microsoft Windows 2000, Windows XP, Windows 2003, Windows Vista or Windows 7. J-Trace has a built-in 20-pin JTAG connector and a built in 38-pin JTAG+Trace connector, which is compatible with the standard 20-pin connector and 38-pin connector defined by ARM.

Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 52.



2.5.5 J-Trace for Cortex-M

J-Trace for Cortex-M is a JTAG/SWD emulator designed for Cortex-M cores which include trace (ETM) support. J-Trace for Cortex-M can also be used as a regular J-Link and it also supports ARM7/9 cores. Please note that tracing on ARM7/9 targets is not supported by J-Trace for Cortex-M. In order to use ETM trace on ARM7/9 targets, a J-Trace is needed.

Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 52.

2.5.6 Flasher ARM

Flasher ARM is a programming tool for microcontrollers with onchip or external Flash memory and ARM core. Flasher ARM is designed for programming flash targets with the J-Flash software or stand-alone. In addition to that Flasher ARM has all of the J- Link functionality. Flasher ARM connects via USB or via RS232 interface to a PC, running Microsoft Windows 2000, Windows XP, Windows 2003 or Windows Vista. Flasher ARM has a built-in 20-pin JTAG connector, which is compatible with the standard 20-pin connector defined by ARM.





2.6 J-Link OEM versions

There are several different OEM versions of J-Link on the market. The OEM versions look different, but use basically identical hardware. Some of these OEM versions are limited in speed, some of these can only be used with certain chips and some of these have certain add-on features enabled, which normally requires license. In any case, it should be possible to use the J-Link software with these OEM versions. However, proper function cannot be guaranteed for OEM versions. SEGGER Microcontroller does not support OEM versions; support is provided by the respective OEM.

2.6.1 Analog Devices: mIDASLink

mIDASLink is an OEM version of J-Link, sold by Analog Devices.

Limitations

mIDASLink works with Analog Devices chips only. This limitation can NOT be lifted; if you would like to use J-Link with a device from an other manufacturer, you need to buy a separate J-Link.

Licenses

Licenses for RDI, J-Link ARM FlashDL and FlashBP are included. Other licenses can be added.



2.6.2 Atmel: SAM-ICE

SAM-ICE is an OEM version of J-Link, sold by Atmel.

Limitations

SAM-ICE works with Atmel devices only. This limitation can NOT be lifted; if you would like to use J-Link with a device from an other manufacturer, you need to buy a separate J-Link.

Licenses

Licenses for RDI and GDB Server are included. Other licenses can be added.



2.6.3 **Digi: JTAG Link**

Digi JTAG Link is an OEM version of J-Link, sold by Digi International.

Limitations

Digi JTAG Link works with Digi devices only. This limitation can NOT be lifted; if you would like to use J-Link with a device from an other manufacturer, you need to buy a separate J-Link.

Licenses

License for GDB Server is included. Other licenses can be added.

IAR: J-Link / J-Link KS 2.6.4

IAR J-Link / IAR J-Link KS are OEM versions of J-Link, sold by IAR.

Limitations

IAR J-Link / IAR J-Link KS can not be used with Keil MDK. This limitation can NOT be lifted; if you would like to use J-Link with Keil MDK, you need to buy a separate J-Link. IAR J-Link does not support kickstart power.

Licenses

No licenses are included. All licenses can be added.

IAR: J-Link Lite 2.6.5

IAR J-Link Lite is an OEM version of J-Link, sold by IAR.

Limitations

IAR J-Link Lite can not be used with Keil MDK. This limitation can NOT be lifted; if you would like to use J-Link with Keil MDK, you need to buy a separate J-Link.

JTAG speed is limited to 4 MHz.

Licenses

No licenses are included. All licenses can be added.

IAR J-Link is only delivered and supported as part of Starter-Kits. It is not Note: sold to end customer directly and not guaranteed to work with custom hardware.





Digi

TAG



2.6.6 IAR: J-Trace

IAR J-Trace is an OEM version of J-Trace, sold by IAR.

Limitations

IAR J-Trace can not be used with Keil MDK. This limitation can NOT be lifted; if you would like to use J-Trace with Keil MDK, you need to buy a separate J-Trace.

Licenses

No licenses are included. All licenses can be added.

2.6.7 NXP: J-Link Lite LPC Edition

J-Link Lite LPC Edition is an OEM version of J-Link, sold by NXP.

Limitations

J-Link Lite LPC Edition only works with NXP devices. This limitation can NOT be lifted; if you would like to use J-Link with a device from an other manufacturer, you need to buy a separate J-Link.

Licenses

No licenses are included.

2.6.8 SEGGER: J-Link Lite

J-Link ARM Lite is a fully functional OEM-version of SEGGER J-Link ARM. If you are selling evaluation-boards, J-Link ARM Lite is an inexpensive emulator solution for you. Your customer receives a widely acknowledged JTAG-emulator which allows him to start right away with his development.

Limitations

JTAG speed is limited to 4 MHz

Licenses

No licenses are included. All licenses can be added.

Note

J-Link ARM Lite is only delivered and supported as part of Starter Kits. It is not sold to end customer and not guaranteed to work with custom hardware.







2.7 J-Link OBs

J-Link OBs (J-Link On Board) are single chip versions of J-Link which are used on various evalboards. It is legal to use J-Link software with these boards, provided that the eval board manufacturer has obtained a license from SEGGER. The following list shows the eval board manufacturer which are allowed to use J-Link OBs:

- IAR Systems
- Embedded Artists

2.8 Illegal Clones

Clones are copies of SEGGER products which use the copyrighted SEGGER Firmware without a license. It is strictly prohibited to use SEGGER J-Link software with illegal clones of SEGGER products. Manufacturing and selling these clones is an illegal act for various reasons, amongst them trademark, copyright and unfair business practise issues.

The use of illegal J-Link clones with this software is a violation of US, European and other international laws and is prohibited.

If you are in doubt if your unit may be legally used with SEGGER J-Link software, please get in touch with us.

End users may be liable for illegal use of J-Link software with clones.

Chapter 3

J-Link and J-Trace related software

This chapter describes Segger's J-Link / J-Trace related software portfolio, which covers nearly all phases of the development of embedded applications. The support of the remote debug interface (RDI) and the J-Link GDBServer allows an easy J-Link integration in all relevant toolchains.

3.1 J-Link related software

3.1.1 J-Link software and documentation package

J-Link is shipped with a bundle of applications. Some of the applications require an additional license, free trial licenses are available upon request from *www.segger.com*.

Software	Description
JLinkARM.dll	DLL for using J-Link / J-Trace with third-party programs.
JLink.exe	Free command-line tool with basic functionality for target analysis.
JLinkSTR91x	Free command-line tool to configure the ST STR91x cores. For more information please refer to <i>J-Link STR91x Commander (Command line tool)</i> on page 67
JLinkSTM32	Free command-line tool for STM32 devices. Can be used to disable the hardware watchdog and to unsecure STM32 devices (override read-protection).
J-Link TCP/IP Server	Free utility which provides the possibility to use J-Link / J-Trace remotely via TCP/IP.
J-Mem memory viewer	Free target memory viewer. Shows the memory content of a running target and allows editing as well.
J-Flash	Stand-alone flash programming application. Requires an addi- tional license. For more information about J-Flash please refer to J-Flash ARM User's Guide (UM08003).
RDI support	Provides Remote Debug Interface (RDI) support. This allows the user to use J-Link with any RDI-compliant debugger. (Addi- tional license required)
J-Link Configurator	GUI-based configuration tool for J-Link. Allows configuration of USB identification as well as TCP/IP identification of J-Link. For more information about the J-Link Configurator, please refer to <i>J-Link Configurator</i> on page 91.
J-Link GDB Server	The J-Link GDB Server is a remote server for the GNU Debug- ger (GDB). For more information about J-Link GDB Server, please refer to J-Link GDB Server User's Guide (UM08005).
J-Link GDB Server command line ver- sion	Command line version of the J-Link GDB Server. Same func- tionality as the GUI version.
Dedicated flash programming utili- ties	Free dedicated flash programming utilities for the following eval boards: Cogent CSB737, ST MB525, Toshiba TOPAS 910.

Table 3.1: J-Link /	J-Trace related softwa	are
•		

3.1.2 List of additional software packages

The software packages listed below are available upon request from *www.seg-ger.com*.

Software	Description
JTAGLoad	Command line tool that opens an \mathtt{svf} file and sends the data in it via J-Link / J-Trace to the target.
J-Link Software Developer Kit (SDK)	The J-Link Software Developer Kit is needed if you want to write your own program with J-Link / J-Trace.
J-Link Flash Soft- ware Developer Kit (SDK)	An enhanced version of the JLinkARM.DLL, which contains additional API functions for flash programming.

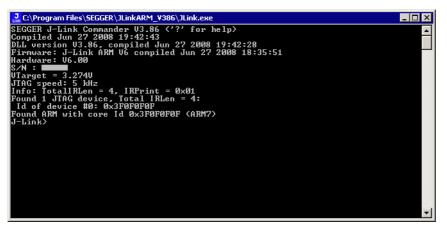
Table 3.2: J-Link / J-Trace additional software packages

3.2 J-Link software and documentation package in detail

The J-Link / J-Trace software documentation package is supplied together with J-Link / J-Trace and may also be downloaded from *www.segger.com*.

3.2.1 J-Link Commander (Command line tool)

J-Link Commander (JLink.exe) is a tool that can be used for verifying proper installation of the USB driver and to verify the connection to the ARM chip, as well as for simple analysis of the target system. It permits some simple commands, such as memory dump, halt, step, go and ID-check, as well as some more in-depths analysis of the state of the ARM core and the ICE breaker module.



3.2.1.1 Using command script files

J-Link commander can also be used in script mode which allows the user to use J-Link commander for batch processing and without user interaction. When using J-Link commander in script mode, the path to a script file is passed to it. The syntax in the script file is the same as when using regular commands in J-Link commander (one line per command).

Example

JLink.exe C:\script.jlink

Contents of script.jlink:

r h exec device = STM32F103ZE loadbin C:\firmware.bin,0x08000000

3.2.2 SWO Analyzer

SWO Analyzer (SWOAnalyzer.exe) is a tool that analyzes SWO output. Status and summary of the analysis are output to standard out, the details of the analysis are stored in a file.



Usage

SWOAnalyzer.exe <SWOfile>

This can be achieved by simply dragging the SWO output file created by the J-Link DLL onto the executable.

Creating an SWO output file

In order to create the SWO output file, which is th input file for the SWO Analyzer, the J-Link config file needs to be modified.

It should contain the following lines:

[SWO]

```
SWOLogFile="C:\TestSWO.dat"
```

3.2.3 J-Link STR91x Commander (Command line tool)

J-Link STR91x Commander (JLinkSTR91x.exe) is a tool that can be used to configure STR91x cores. It permits some STR9 specific commands like:

- Set the configuration register to boot from bank 0 or 1
- Erase flash sectors
- Read and write the OTP sector of the flash
- Write-protect single flash sectors by setting the sector protection bits
- Prevent flash from communicate via JTAG by setting the security bit

All of the actions performed by the commands, excluding writing the OTP sector and erasing the flash, can be undone. This tool can be used to erase the flash of the controller even if a program is in flash which causes the ARM core to stall.

C:\Work\JLin	kARM\Output\Debug\JLinkSTR91x.exe									
Available commands are:										
fsize	Set the size of the primary flash manually. Syntax: fsize 0111213, where 0 selects a 256 Kbytes device, 1 a 512 Kbytes device, 2 a 1024 KBytes device and 3 a 2048 Kbytes device									
showconf mem	Show configuration register content and security status Read memory Syntax: mem <addr>, <numbytes></numbytes></addr>									
erase	Erase flash sectors (OTP can not be erased). Syntax: erase {SectorMaskL}, {SectorMaskH}									
	SectorMaskL = Bits 0-8 mask sectors 0-8 of bank 0 SectorMaskH = Bits 0-4 mask sectors 0-4 of bank 1 Bit 1? masks the configuration sector Bit 18 masks the User-Code sector All other bits are ignored									
	Erase flash bank Ø									
	Erase flash bank 1									
erase all setb	Perform a full chip erase Boot from flash bank x (0 and 1 are available) Sutax: setb (int)									
blank	Blank check all flash sectors									
secure	Set the security bit. Protects device from read or debug access through the JTAG port (can only be cleared by a full chip erase).									
unsecure protect	Unsecure the device. Content of configuration register is saved. Protect flash sectors.									
	Syntax: protect <bankøsectormask>, <bank1sectormask> BankØSectorMask: Bits 0-8 mask flash sectors 0-8 of bank 0 Bank1SectorMask: Bits 0-4 mask flash sectors 0-4 of bank 1</bank1sectormask></bankøsectormask>									
unprotect	Unprotect flash sectors. Syntax: unprotect (Bank0SectorMask), (Bank1SectorMask) Bank0SectorMask: Bits 0-8 mask flash sectors 0-8 of bank 0 Bank1SectorMask: Bits 0-4 mask flash sectors 0-4 of bank 1									
readotp	Read OTP sectors									
writeotp 	Write words to the OTP sectors. Syntax: writeotp <word1>, [<word2>,, <word8>] Quit</word8></word2></word1>									
q 										
J-Link>_	v									

When starting the STR91x commander, a command sequence will be performed which brings MCU into Turbo Mode.

"While enabling the Turbo Mode, a dedicated test mode signal is set and controls the GPIOs in output. The IOs are maintained in this state until a next JTAG instruction is send." (ST Microelectronics)

Enabling Turbo Mode is necessary to guarantee proper function of all commands in the STR91x Commander.

3.2.4 J-Link STM32 Commander (Command line tool)

J-Link STM32 Commander (JLinkSTM32.exe) is a free command line tool which can be used to disable the hardware watchdog of STM32 devices which can be activated by programming the option bytes. Moreover the J-Link STM32 Commander unsecures a read-protected STM32 device by re-programming the option bytes.

Note: Unprotecting a secured device or will cause a mass erase of the flash memory.



3.2.5 J-Link TCP/IP Server (Remote J-Link / J-Trace use)

The J-Link TCP/IP Server allows using J-Link / J-Trace remotely via TCP/IP. This enables you to connect to and fully use a J-Link / J-Trace from another computer. Performance is just slightly (about 10%) lower than with direct USB connection.

J-Link TCP/IF	9 Server				х
IPStat	Not connec	ted]
	В	W		WR	
This connection	0		Ο	0	ī
Total	0		Ο	0	ī
USBStat	Not connec	ted]
	🗖 Stay on	top	At	oout	
Status					
Waiting for clien	it on port 190	20.			

The J-Link TCP/IP Server also accepts commands which are passed to the J-Link TCP/ IP Server via the command line.

3.2.5.1 List of available commands

The table below lists the commands accepted by the J-Link TCP/IP Server

Command	Description							
port	Selects the IP port on which the J-Link TCP/IP Server is listening.							
usb	Selects a usb port for communication with J-Link.							

Table 3.3: Available commands

3.2.5.2 port

Syntax

-port <Portno.>

Example

To start the J-Link TCP/IP Server listening on port 19021 the command should look as follows:

-port 19021

3.2.5.3 usb

Syntax

-usb <USBIndex>

Example

Currently usb 0-3 are supported, so if the J-Link TCP/IP Server should connect to the J-Link on usb port 2 the command should look as follows:

-usb 2

3.2.6 J-Mem Memory Viewer

J-Mem displays memory contents of ARM-systems and allows modifications of RAM and SFRs (Special Function Registers) while the target is running. This makes it possible to look into the memory of an ARM chip at run-time; RAM can be modified and SFRs can be written. You can choose between 8/16/32-bit size for read and write accesses. J-Mem works nicely when modifying SFRs, especially because it writes the SFR only after the complete value has been entered.

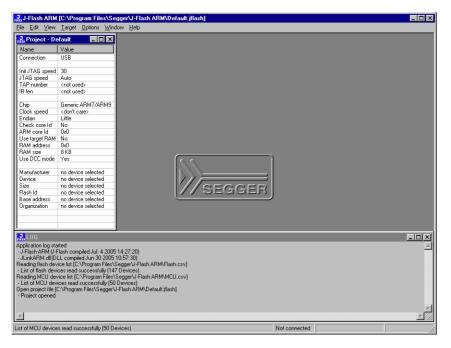
J-Mem																	_ 🗆 🗵
<u>File T</u> arget	Opti	ons	<u>H</u> elp														
Address: 0x0	-		×1	x2	×4	B	efresh										
Address	Й	1	2	3	4	5	6	-	8	9	A	В	С	D	E	F	ASCII
00000000	06	00	ЙЙ	EA	FE	FF	FF	EA	FE	FF	FF	EA	FE	FF	FF	ĒA	
00000010	FE	FF	FF	EA	FE	FF	FF	EA	5C	07	00	EA	70	07	ØЙ	EA	
00000020	50	DØ	9F	E5	50	00	9F	E5	ØF	ĒØ	ÂØ	E1	10	FF	2F	E1	PP
00000030	40	00	9F	E5	D1	FØ	21	E3	40	80	9F	E5	D2	FØ	21	E3	et.et.
00000040	00	DØ	AØ	E1	60	00	40	E2	13	FØ	21	E3	00	DØ	AØ	E1	`.et
00000050	2C	00	9F	E5	ØF	EØ	AØ	E1	10	FF	2F	E1	24	ΕØ	9F	E5	,/.\$
00000060	24	00	9F	E5	10	FF	2F	E1	FE	FF	FF	EA	FE	FF	FF	EA	\$/
00000070	FE	FF	FF	EA	FE	FF	FF	EA	00	00	21	00	91	00	00	00	
00000080	00	FØ	FF	FF	25	01	00	00	68	00	00	00	DD	12	00	00	h
00000090	00	B5	1A	48	9F	21	C9	43	19	4 A	ØA	60	19	49	80	22	HC.J.`.I."
000000A0	12	02	ØA	60	18	49	01	62	81	6E	C9	07	FC	D5	17	49	`.I.b.nI
000000B0	C1	62	81	6E	49	07	FC	D5	81	6E	09	07	FC	D5	04	21	.b.nI!
00000000	01	63	81	6E	09	07	FC	D5	01	6 B	03	22	ØA	43	02	63	.c.nk.".C.c
000000D0	81	6E	09	07	FC	D5	ØE	48	ØE	49	01	60	01	20	05	EØ	.nH.I.`
000000E0	04	21	41	43	ØA	4 A	0C	4B	53	50	40	10	1F	28	F7	DB	AC.J.KSPC
000000F0	ØA	48	ØB	49	01	60	01	BC	00	47	CØ	46	00	FC	FF	FF	.H.I.`G.F
00000100	00	01	30	00	44	FD	FF	FF	01	06	00	00	05	1C	19	10	0.D
00000110	80	FØ	FF	FF	6C	00	00	00	70	00	00	00	34	F1	FF	FF	1p4
00000120	74	00	00	00	12	40	13	48	70	B4	81	BØ	11	10	12	1D	tJ.Hp
00000130	ØA	EØ	ØB	68	54	68	15	68	00	2B	03	DØ	5B	1E	E6	5C	hTh.h.+[\
00000140	EE	54	FB	D1	ØC	31	ØC	32	81	42	ØC	D2	53	68	14	68	.T1.2.BSh.h 💌
Ready										C	onneo	ted		ARM	core	id: 31	F0F0F0F Speed: 4000 kHz //

3.2.7 J-Flash ARM (Program flash memory via JTAG)

J-Flash ARM is a software running on Windows 2000, Windows XP, Windows 2003 or Windows Vista systems and enables you to program your flash EEPROM devices via the JTAG connector on your target system.

J-Flash ARM works with any ARM7/9 system and supports all common external flashes, as well as the programming of internal flash of ARM microcontrollers. It allows you to erase, fill, program, blank check, upload flash content, and view memory functions of the software with your flash devices.

J-Flash requires a additional license from Segger. Even without a license key you can still use J-Flash ARM to open project files, read from connected devices, blank check target memory, verify data files and so on. However, to actually program devices via J-Flash ARM and J-Link / J-Trace you are required to obtain a license key from us. Evaluation licenses are available free of charge. For further information go to our website or contact us directly.



Features

- Works with any ARM7/ARM9 chip
- ARM microcontrollers (internal flash) supported
- Most external flash chips can be programmed
- High-speed programming: up to 300 Kbytes/second (depends on flash device)
- Very high-speed blank check: Up to 16 Mbytes/sec (depends on target)
- Smart read-back: Only non-blank portions of flash transferred and saved
- Easy to use, comes with projects for standard eval boards.

3.2.8 J-Link RDI (Remote Debug Interface)

The J-Link RDI software is an remote debug interface for J-Link. It makes it possible to use J-Link with any RDI compliant debugger. The main part of the software is an RDI-compliant DLL, which needs to be selected in the debugger. There are two additional features available which build on the RDI software foundation. Each additional features requires an RDI license in addition to its own license. Evaluation licenses are available free of charge. For further information go to our website or contact us directly.

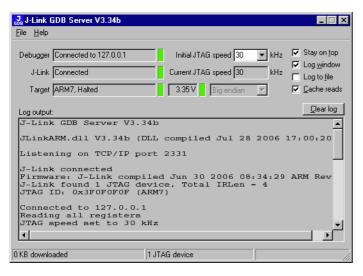
Note: The RDI software (as well as flash breakpoints and flash downloads) do not require a license if the target device is an LPC2xxx. In this case the software verifies that the target device is actually an LPC 2xxx and have a device-based license.

3.2.8.1 Flash download and flash breakpoints

Flash download and flash breakpoints are supported by J-Link RDI. For more information about flash download and flash breakpoints, please refer to *J-Link RDI User's Guide (UM08004)*, chapter *Flash download* and chapter *Breakpoints in flash memory*.

3.2.9 J-Link GDB Server

GDB Server is a remote server for the GNU Debugger GDB. GDB and GDB Server communicate via a TCP/IP connection, using the standard GDB remote serial protocol. The GDB Server translates the GDB monitor commands into J-Link commands.



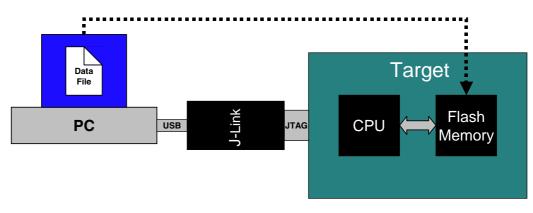
The GNU Project Debugger (GDB) is a freely available debugger, distributed under the terms of the GPL. It connects to an emulator via a TCP/IP connection. It can connect to every emulator for which a GDB Server software is available. The latest Unix version of the GDB is freely available from the GNU committee under:

http://www.gnu.org/software/gdb/download/

J-Link GDB Server is distributed free of charge.

3.3 Dedicated flash programming utilities for J-Link

The SEGGER J-Link comes with dedicated flash programming utilities (DFPU) for a number of popular Eval boards. These utilities are designed to program a .bin file into the flash memory of the target hardware, with J-Link. Each dedicated flash programming utility works only with the Eval board it was designed for.



3.3.1 Introduction

Using the dedicated flash programming utilities which come with J-Link, is permitted for development purposes only. As long as the dedicated flash programming tools are used for development purposes only, no additional license is required. If you want to use the dedicated flash programming utilities for commercial and production purposes, you need to obtain a license from SEGGER. SEGGER also offers to create dedicated flash programming utilities for custom hardware. When starting a dedicated flash programming utility, a message box appears which tells the user about the purpose of the dedicated flash programming utility:



3.3.2 Supported Eval boards

The list below shows the Eval boards for which dedicated flash programming utilities have been already developed. Simple flash programming utilities for other, popular Eval boards are on the schedule.

CPU / MCU	Eval board manufacturer	Eval board name	Flash memory
Atmel AT91SAM9263	Cogent	CSB737	Typically 65 MB external NOR flash
ST STM32F103RBT6	ST Microelectron- ics	MB525	Typically 128 KB internal flash
Toshiba TMPA910CRXBG	Toshiba	TOPAS910	Typically 32 MB external NOR flash
NXP LPC3250 Phytec		PCM-967	Typically 32 MB external NAND flash (ST NAND256R3A)

Table 3.4:

3.3.3 Supported flash memories

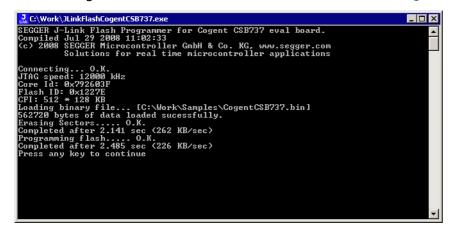
The dedicated flash programming utilities for J-Link can be created for the following flash memories:

- External NOR flash
- Internal flash
- NAND flash
- Data flash
- SPI flash

In order to use external NOR flash, a CFI compliant flash memory has to be used because the flash programming utilities use the CFI information to detect the flash size and sectorization.

3.3.4 How to use the dedicated flash programming utilities

The dedicated flash programming utilities are very simple to use. Every tool expects a path to a data file (*.bin) passed as a command line parameter, on startup. If no path is passed the flash programming utility searches for a data in the Samples\directory. This .bin file has to be named as shown in the table above. For example, for the Cogent CSB737 Eval board this file is named: CogentCSB737.bin.



3.3.5 Using the dedicated flash programming utilities for production and commercial purposes

If you want to use dedicated flash programming utilities for production and commercial purposes you need to obtain a license from SEGGER. In order to obtain a license for a dedicated flash programming utility, there are two options:

- Purchasing the source code of an existing dedicated flash programming utility
- Purchasing the source code of a dedicated flash programming utility for custom hardware

The source code can be compiled using a Microsoft Visual C++ V6 or newer compiler. It contains code which is executed on the target device (RAMCODE). This RAMCODE may not be used with debug probes other than J-Link.

3.3.5.1 Purchasing the source code of an existing dedicated flash programming utility

Purchasing the source code of an existing dedicated flash programming utility (described above) allows you to use the dedicated flash programming utility for production and commercial purposes. Making the resulting executable publicly available is not permitted.

For more information about the pricing for the source code of existing dedicated flash programming utilities, please refer to the price list on our website *http://www.segger.com/pricelist_jlink.html#8.20.01*.

3.3.5.2 Purchasing the source code of a dedicated flash programming utility for custom hardware

SEGGER also offers to design dedicated flash programming utilities for custom hardware for which you will also need to obtain a license. The resulting executable may be used for organization internal purposes only.

3.3.6 F.A.Q.

- Q: Q: Can the dedicated flash programming utilities be used for commercial purposes?
- A: A: Yes, you can buy the source code of one or more of the flash programming utilities which makes it possible to use them for commercial and production purposes.
- Q: Q: I want to use the dedicated flash programming utilities with my own hardware. Is that possible?
- A: A: The free dedicated flash programming utilities which come with J-Link do not support custom hardware.mIn order to use your own hardware with a dedicated flash programming utility, SEGGER offers to create dedicated flash programming utilities for custom hardware
- Q: Q: Do I need a license to use the dedicated flash programming utilities?
- A: A: As long as you use the dedicated flash programming utilities, which come with J-Link, for development purposes only, you do not need an additional license. In order to use them for commercial and/or production purposes you need to obtain a license from SEGGER.
- Q: Q: Which file types are supported by the dedicated flash programming utilities?
- A: A: Currently, the dedicated flash programming utilities support *.bin files.
- Q: Q: Can I use the dedicated flash programming utilities with other debug probes than J-Link?
- A: A: No, the dedicated flash programming utilities only work with J-Link

3.4 Additional software packages in detail

The packages described in this section are not available for download. If you wish to use one of them, contact SEGGER Microcontroller Systeme directly.

3.4.1 JTAGLoad (Command line tool)

JTAGLoad is a tool that can be used to open an svf (Serial vector format) file. The data in the file will be sent to the target via J-Link / J-Trace.



3.4.2 J-Link Software Developer Kit (SDK)

The J-Link Software Developer Kit is needed if you want to write your own program with J-Link / J-Trace. The J-Link DLL is a standard Windows DLL typically used from C programs (Visual Basic or Delphi projects are also possible). It makes the entire functionality of J-Link / J-Trace available through its exported functions, such as halt-ing/stepping the ARM core, reading/writing CPU and ICE registers and reading/writing memory. Therefore it can be used in any kind of application accessing an ARM core. The standard DLL does not have API functions for flash programming. However, the functionality offered can be used to program flash. In this case, a flash loader is required. The table below lists some of the included files and their respective purpose.

Files	Contents
GLOBAL.h JLinkARMDLL.h	Header files that must be included to use the DLL functions. These files contain the defines, typedef names, and function dec- larations.
JLinkARM.lib	A Library that contains the exports of the JLink DLL.
JLinkARM.dll	The DLL itself.
Main.c	Sample application, which calls some JLinkARM DLL functions.
JLink.dsp JLink.dsw	Project files of the sample application. Double click JLink.dsw to open the project.
JLinkARMDLL.pdf	Extensive documentation (API, sample projects etc.).
Table 3.5: J-Link SDK	

3.4.3 J-Link Flash Software Developer Kit (SDK)

This is an enhanced version of the JLinkARM.DLL which contains additional API functions for flash programming. The additional API functions (prefixed JLINKARM_FLASH_) allow erasing and programming of flash memory. This DLL comes with a sample executable, as well as with source code of this executable and a Microsoft Visual C/C++ project file. It can be an interesting option if you want to write your own programs for production purposes.

3.5 Using the J-LinkARM.dll

3.5.1 What is the JLinkARM.dll?

The J-LinkARM.dll is a standard Windows DLL typically used from C or C++, but also Visual Basic or Delphi projects. It makes the entire functionality of the J-Link / J-Trace available through the exported functions.

The functionality includes things such as halting/stepping the ARM core, reading/ writing CPU and ICE registers and reading/writing memory. Therefore, it can be used in any kind of application accessing an ARM core.

3.5.2 Updating the DLL in third-party programs

The JLinkARM.dll can be used by any debugger that is designed to work with it. Some debuggers, like the IAR C-SPY[®] debugger, are usually shipped with the JLinkARM.dll already installed. Anyhow it may make sense to replace the included DLL with the latest one available, to take advantage of improvements in the newer version.

3.5.2.1 Updating the JLinkARM.dll in the IAR Embedded Workbench for ARM (EWARM)

It's recommended to use the J-Link DLL updater to update the JLinkARM.dll in the IAR Embedded Workbench. The IAR Embedded Workbench IDE is a high-performance integrated development environment with an editor, compiler, linker, debugger. The compiler generates very efficient code and is widely used. It comes with the J-LinkARM.dll in the arm\bin subdirectory of the installation directory. To update this DLL, you should backup your original DLL and then replace it with the new one.

Typically, the DLL is located in C:\Program Files\IAR Systems\Embedded Workbench 6.n\arm\bin\.

After updating the DLL, it is recommended to verify that the new DLL is loaded as described in *Determining which DLL is used by a program* on page 80.

J-Link DLL updater

The J-Link DLL updater is a tool which comes with the J-Link software and allows the user to update the JLinkARM.dll in all installations of the IAR Embedded Workbench, in a simple way. The updater is automatically started after the installation of a J-Link software version and asks for updating old DLLs used by IAR. The J-Link DLL updater can also be started manually. Simply enable the checkbox left to the IAR installation which has been found. Click **Ok** in order to update the JLinkARM.dll used by the IAR installation.

he following 3rd	-party applications us	ing JLinkARM.dll have	been found:				
IAR Embedde	d Workbench for AF	RM 4.40A (DLL V3.20h	in "C:\Tool\C\IAR\AF	RM_V440A\ARM\bin'')			
IAR Embedde	d Workbench for AP	RM 4.41A (DLL V3.80c	in "C:\Tool\C\IAR\AF	RM_V441A\ARM\bin'')			
IAR Embedde	d Workbench for AP	RM 4.42A (DLL V3.84 in	"C:\Tool\C\IAR\AR	M_V442A\ARM\bin'')			
🗹 IAR Embedde	d Workbench for AP	RM 4.31A (DLL V3.82 in	"C:\Tool\C\IAR\AR	M_V431A\ARM\bin'')			
IAR Embedde	ed Workbench for AF	RM 4.30A (DLL V3.80c)	in "C:\Tool\C\IAR\AF	RM_V430A\ARM\bin'')			
IAR Embedde	ed Workbench for AF	RM 5.10 (DLL V3.78d in	"C:\Tool\C\IAR\ARI	M_V510\ARM\bin'')			
IAR Embedde	ed Workbench for AF	RM 5.20 (DLL V3.85f in	"C:\Tool\C\IAR\ARM	1_V520_beta885\ARM\b	in'')		
VIAR Embedded Workbench for ARM 5.20 [DLL V3.85j in "C:\Too\C\IAR\ARM V520 beta902\ARM\bin"]							
IAR Embedde	d Workbench for AP	RM 5.11 (DLL V3.78 in '	C:\Tool\C\IAR\ARM	_V511_BETA_607\ARM	\bin'')		
AR Embedded Workbench for ARM 5.11 (DLL V3.85h in "C:\Tool\C\IAR\ARM_V511_9799\ARM\bin")							
🗹 IAR Embedde	ed Workbench for AP	RM 5.20 (DLL V3.81k in	n ''C:\Program Files\IA	R Systems\Embedded W	/orkbench 5.0 (E	WARM 5.20.X	ALPHA)\ARM
Select All	Select None						
elect the ones y	ou would like to repla	ace by this version.					
		and kept in the same fo	lder, allowing manual	"undo".			
	do not replace existi						
ou can alwaus r	perform this operation	i at a later time via start.	menu.				Cancel

3.5.3 Determining the version of JLinkARM.dll

To determine which version of the JLinkARM.dll you are facing, the DLL version can be viewed by right clicking the DLL in explorer and choosing Properties from the context menu. Click the Version tab to display information about the product version.

jlinkarm.dll Properties		
General Version Security Summary		
File version: 3.0.4.0		
Description: SEGGER J-Link ARM interface DLL		
Copyright: Copyright © 2004, 2005		
Copyright Copyright 2004, 2005		
OK Cancel Apply		

3.5.4 Determining which DLL is used by a program

To verify that the program you are working with is using the DLL you expect it to use, you can investigate which DLLs are loaded by your program with tools like Sysinternals' Process Explorer. It shows you details about the DLLs, used by your program, such as manufacturer and version.

💐 Process Explorer - Sysi	internals: www.sysinternals	s.com	
File Options View Proces	ss Find DLL Help		
🖬 😰 📰 🖺 🎫 🤞	🥴 🖆 メ 🖍 枪 🤁		
Process	PID CPU D	escription Compan	
🖃 📰 System Idle Process	0 99		
Interrupts	n/a Ha	ardware Interrupts	
DPCs	n/a De	eferred Procedu	
🕀 📰 System	8		
🖃 🖳 explorer.exe	1148 Wi	indows Explorer Microsoft	
2 procexp.exe		sinternals Proc Sysintern	
laridePM.exe	1460 IAF	R Embedded IAR Syst	
Name 🛆	Description	Company Name	Version
indicdll.dll	Keyboard Language Indicator	Shell Microsoft Corporation	5.00.2920.0000
jlinkarm.dll	SEGGER J-Link ARM interface	e DLL SEGGER Microcontrolle	r Systeme GmbH 3.00.0004.0000
Kernel.dll	IAR C-SPY Debugger Kernel	IAR Systems	4.06.0000.0000
kernel32.dll	Windows NT BASE API Client	t DLL Microsoft Corporation	5.00.2195.6688
locale.nls			
LogWindow.dll	IAR Log Window	IAR Systems	4.06.0000.0000
Iz32.dll	LZ Expand/Compress API DLI		5.00.2195.6611
MFC71.dll	MFCDLL Shared Library - Reta		7.10.3077.0000
mpr.dll	Multiple Provider Router DLL	Microsoft Corporation	5.00.2195.6611
CPU Usage: 1% Commit Ch	arge: 12.24% Processes: 34		/ 30 3077 0000

Process Explorer is - at the time of writing - a free utility which can be downloaded from *www.sysinternals.com*.

Chapter 4 Setup

This chapter describes the setup procedure required in order to work with J-Link / J-Trace. Primarily this includes the installation of the J-Link software and documentation package, which also includes a kernel mode J-Link USB driver in your host system.

4.1 Installing the J-Link ARM software and documentation pack

J-Link is shipped with a bundle of applications, corresponding manuals and some example projects and the kernel mode J-Link USB driver. Some of the applications require an additional license, free trial licenses are available upon request from *www.segger.com*.

Refer to chapter *J*-*Link and J*-*Trace related software* on page 63 for an overview about the J-Link software and documentation pack.

4.1.1 Setup procedure

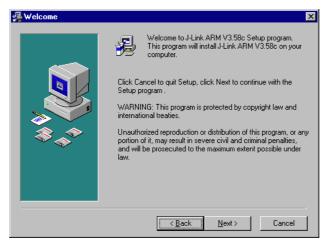
To install the J-Link ARM software and documentation pack, follow this procedure:

Note: We recommend to check if a newer version of the J-Link software and documentation pack is available for download before starting the installation. Check therefore the J-Link related download section of our website: http://www.seager.com/download ilink.html

 Before you plug your J-Link / J-Trace into your computer's USB port, extract the setup tool Setup_JLinkARM_V<VersionNumber>.zip. The setup wizard will install the software and documentation pack that also includes the certified J-Link USB driver. Start the setup by double clicking Setup_JLinkARM_V<Version-Number>.exe. The license Agreement dialog box will be opened. Accept the terms with the Yes button.

🚰 License Agreement	×
是	Please read the following license agreement. Use the scroll bar to view the rest of this agreement.
	Important - Read carefully: This license is a legal agreement between YOU (either an individual or a single entity) and SEGGER Microcontroller Systeme GmbH (called SEGGER). By downloading and/or using J-Link ARM software, you agree to be bound by the terms of this agreement.
\$\$_\$\$	1. LICENSE AGREEMENT In this agreement "Licensor" shall mean SEGGER except under the following circumstancest If Licensee acquired the product as a bundled component of
	Do you accept all the terms of the preceding license agreement? If so, click on the Yes push button. If you select No, Setup will close.
	Yes No

2. The **Welcome** dialog box is opened. Click **Next** > to open the **Choose Destina**tion Location dialog box.



3. Accept the default installation path C:\Program Files\SEG-GER\JLinkARM_V<VersionNumber> or choose an alternative location. Confirm your choice with the **Next >** button.

覺 Choose Destination Lo	ocation X
	Setup will install J-Link ARM V3.58c in the following folder. To install into a different folder, click Browse, and select
	another folder. You can choose not to install J-Link ARM V3.58c by clicking
	Cancel to exit Setup.
	Destination Folder C:\\SEGGER\JLinkARM_V358c <u>Br</u> owse
	< <u>B</u> ack

4. The **Choose options** dialog is opened. The **Create entry in start menu** and the **Add shortcuts to desktop** option are preselected. Accept or deselect the options and confirm the selection with the **Next >** button.

🚭 Choose options	×
	Choose options for creating shortcuts
	Create entry in start menu
	Add shortcuts to desktop
	< <u>B</u> ack <u>Next</u> > Cancel

5. The installation process will be started.

Installing	
	Current File Copying file: C:VLinkARM_V358c\USBDriver\JLinkx64.sys All Files Time Remaining 0 minutes 0 seconds
	< <u>B</u> ack <u>N</u> ext⇒ Cancel

6. The **Installation Complete** dialog box appears after the copy process. Close the installation wizard with the **Finish** > button.

🛃 Installation Complete	×
	J-Link ARM V3.58c has been successfully installed. Press the Finish button to exit this installation.
	< <u>B</u> ack <u>Einish > Cancel</u>

The J-Link software and documentation pack is successfully installed on your PC. 7. Connect your J-Link via USB with your PC. The J-Link will be identified and after

a short period the J-Link LED stops rapidly flashing and stays on permanently.

4.2 Setting up the USB interface

After installing the J-Link ARM software and documentation package it should not be necessary to perform any additional setup sequences in order to configure the USB interface of J-Link.

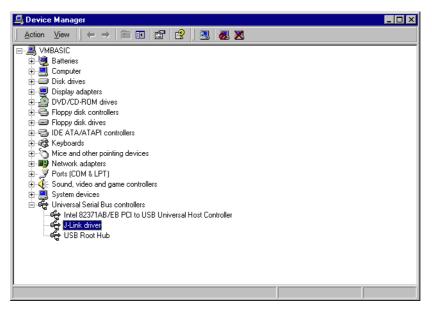
4.2.1 Verifying correct driver installation

To verify the correct installation of the driver, disconnect and reconnect J-Link / J-Trace to the USB port. During the enumeration process which takes about 2 seconds, the LED on J-Link / J-Trace is flashing. After successful enumeration, the LED stays on permanently.

Start the provided sample application $_{JLink.exe}$, which should display the compilation time of the J-Link firmware, the serial number, a target voltage of 0.000V, a complementary error message, which says that the supply voltage is too low if no target is connected to J-Link / J-Trace, and the speed selection. The screenshot below shows an example.



In addition you can verify the driver installation by consulting the Windows device manager. If the driver is installed and your J-Link / J-Trace is connected to your computer, the device manager should list the J-Link USB driver as a node below "Universal Serial Bus controllers" as shown in the following screenshot:



Right-click on the driver to open a context menu which contains the command **Properties**. If you select this command, a **J-Link driver Properties** dialog box is opened and should report: **This device is working properly**.

J-Link driver Propertie	s ? X	
General Driver		
J-Link driver		
Device type:	Universal Serial Bus controllers	
Manufacture	r: Segger	
Location:	J-Link	
Device status		
This device is working properly.		
	Iroubleshooter	
Device usage:		
Use this device (enat	le)	
	OK Cancel	

If you experience problems, refer to the chapter *Support and FAQs* on page 257 for help. You can select the **Driver** tab for detailed information about driver provider, version, date and digital signer.

J-Link dri	ver Properties	? ×
General	Driver	
¢¢	J-Link driver	
	Driver Provider:	Segger
	Driver Date:	07-01-09
	Driver Version:	2.6.5.0
	Digital Signer:	Microsoft Windows Hardware Compatibility Publ
Details.	To uninstall the driv	iver files loaded for this device, click Driver er files for this device, click Uninstall. To update e, click Update Driver. <u>Uninstall</u> <u>Update Driver</u>
		OK Cancel

4.2.2 Uninstalling the J-Link USB driver

If J-Link / J-Trace is not properly recognized by Windows and therefore does not enumerate, it makes sense to uninstall the J-Link USB driver.

This might be the case when:

- The LED on the J-Link / J-Trace is rapidly flashing.
- The J-Link / J-Trace is recognized as **Unknown Device** by Windows.

To have a clean system and help Windows to reinstall the J-Link driver, follow this procedure:

- 1. Disconnect J-Link / J-Trace from your PC.

쪎 Add/Remove Programs - 🗆 × 17 Sort by: Name • Currently installed programs: ۸ Change o Remove 🔜 J-Link ARM V3.66a Windows Driver Package - Segger (jlink) USB (01/09/2007 2.6.5.0) 20 Click here for support information. 🔬 Add Ne To change this program or remove it from your computer, click Change/Remove. <u>⊂</u>hange/Remove S -

(jlink) USB and click the Change/Remove button.

- 3. Confirm the uninstallation process.



4.3 Setting up the IP interface

Some emulators of the J-Link family have (or future members will have) an additional Ethernet interface, to communicate with the host system. These emulators will also come with a built-in web server which allows configuration of the emulator via web interface. In addition to that, you can set a default gateway for the emulator which allows using it even in large intranets. For simplicity the setup process of J-Link Pro (referred to as J-Link) is described in this section.

4.3.1 Configuring J-Link using J-Link Configurator

The J-Link software and documentation package comes with a free GUI-based utility called J-Link Configurator which auto-detects all J-Links that are connected to the host PC via USB & Ethernet. The J-Link Configurator allows the user to setup the IP interface of J-Link. For more information about how to use the J-Link Configurator, please refer to *J-Link Configurator* on page 91.

4.3.2 Configuring J-Link using the webinterface

All emulators of the J-Link family which come with an Ethernet interface also come with a built-in web server, which provides a web interface for configuration. This enables the user to configure J-Link without additional tools, just with a simple web browser. The **Home** page of the web interface shows the serial number, the current IP address and the MAC address of the J-Link.

SEGGER	J-Link Pro Webserver	SEGGER Microcontroller
Home Network information Network configuration System information Emulator status About	Home Emulator information: Firmware build: Dec 22 2008 09:24:26 Serial Number: Network information: Configuration type: User assigned IP Address: 192.168.90.11 /16 Gateway: 192.168.1.1	

The **Network configuration** page allows configuration of network related settings (IP address, subnet mask, default gateway) of J-Link. The user can choose between **automatic** IP assignment (settings are provided by a DHCP server in the network) and **manual** IP assignment by selecting the appropriate radio button.

SEGGER	J-Link Pro Webserver	SEGGER Microcontroller
Home Network information Network configuration System information Emulator status About	Network configuration C Automatic © Manual DHCP IP address: 192.168.90.11 Subnet mask: 255.255.0.0 Gateway: 192.168.1.1 Change	

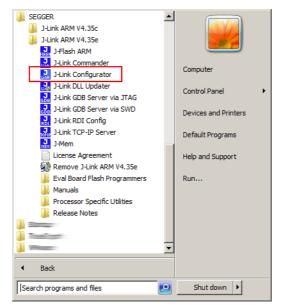
4.4 FAQs

- Q: How can I use J-Link with GDB and Ethernet?
- A: You have to use the J-Link GDB Server in order to connect to J-Link via GDB and Ethernet.

J-Link / J-Trace (UM08001)

4.5 J-Link Configurator

Normally, no configuration is required, especially when using J-Link via USB. For special cases like: having multiple older J-Links connected to the same host PC in parallel, they need to be re-configured to be identified by their real serial number when enumerating on the host PC. This is the default identification method for current J-Links (J-Link with hardware version 8 or later). For re-configuration of old J-Links or for configuration of the IP settings (use DHCP, IP address, subnet mask, ...) of a J-Link supporting the Ethernet interface, SEGGER provides a GUI-based tool, called J-Link Configurator. The J-Link Configurator is part of the J-Link software and documentation package and can be used free of charge.



4.5.1 Configure J-Links using the J-Link Configurator

A J-Link can be easily configured by selecting the appropriate J-Link from the emulator list and using right click -> Configure.

SEGGER J-Link Configuration V4.35e (beta)					
Emulators connected via L	JSB:			Refresh rate	Normal 💌
# A Pr	oduct	Nickname SN	USB Identification	Host Firmware Emulati	or Firmware
SE O	GGER J-Link ARM Pro V3 00 Configur Update 1 Replace		SN 173000305	2011 Sep 6 16:37 2011 S	ep 6 16:37
Emulators connected via T	rce/ie-			<u>S</u> ele	ect all Select none
# / Product	Nickname SN	IP Address MAC Address	Host Firmware Emula	tor Firmware React. Tin	ne Connections 🔺
 0 J-Link ARM 1 J-Link ARM 2 J-Link ARM 3 J-Link ARM 4 J-Link ARM 5 J-Link ARM 6 J-Link ARM 7 J-Link ARM 8 J-Link ARM 9 J-Link ARM 10 J-Link ARM 10 J-Link Config 25636F J-Link Config 25636F J-Link Config 	-Pro V3.00 173001008 -Pro V3.00 173001040 -Pro V3.00 173001041 -Pro V3.00 173001041 -Pro V3.00 173001043 -Pro V3.00 173001043 -Pro V3.00 173001043 -Pro V3.00 173001045 -Pro V3.00 173001045 -Pro V3.00 173001046 -Pro V3.00 173001046 -Pro V3.00 173001046 -Pro V3.00 173001049	192.168.8.2 (00:22:C7:02:04:13 192.168.8.3 (00:22:C7:02:04:14 192.168.8.5 (00:22:C7:02:04:14 192.168.8.5 (00:22:C7:02:04:14 192.168.4.253 00:22:C7:02:04:14	0 2011 Sep 6 16:37 2011 4 1 2011 Sep 6 16:37 2011 3 2011 Sep 6 16:37 2011 3 2 2011 Sep 6 16:37 2011 4 2 2011 Sep 6 16:37 2011 3 2 2011 Sep 6 16:37 2011 4 2 2011 Sep 6 16:37 2011 3 3 2011 Sep 6 16:37 2011 3 4 2011 Sep 6 16:37 2011 3 5 2011 Sep 6 16:37 2011 3 6 2011 Sep 6 16:37 2011 3 2011 Sep 6 16:37 2011 3 201 3 2011 Sep 6 16:37 2011 3 201 3	Aug 18 19:57 (Old) 0.4917 Aug 18 19:57 (Old) 0.528 Aug 18 19:57 (Old) 0.528 Aug 12 517:24 (Old) 0.4017 ep 6 18:37 0.5227 Aug 11 17:30 (Old) 0.4067 Aug 11 17:30 (Old) 0.4067 Aug 11 7:30 (Old) 0.4067 Aug 11 7:30 (Old) 0.4067 Ep 6 18:37 0.4267 ep 6 18:37 0.5267 Aug 16 17:24 (Old) 0.4037 Sele	ns 0 ns 0
			Update fir	mware of selected emulators	Close
Ready	Searching for	emulators: Ready		20 em	ulators found

CHAPTER 4

In order to configure a J-Link to use the new USB identification method (reporting the real serial number) simply select "Real SN" as USB identification method and click the OK button. The same dialog also allows configuration of the IP settings of the connected J-Link if it supports the Ethernet interface.

C	onfigure J-Linl	¢	×
	General		
	Product	SEGGER J-Link ARM Pro V3.00	1
	SN	173000305	
	Nickname		
	– USB Identifical	ion	
	Real SN	▼ Real SN 173000305	1
		n c (DHCP) ⓒ Manual 192 , 168 , 90 , 33	
		255 . 255 . 0 . 0	
	Gateway	255 . 255 . 255 . 255	
		Cancel	

Note: When re-configuring older J-Links which use the old enumeration method (USB identification: USB 0 - USB 3) you can only have 1 J-Link connected which uses the old method at the same time. So re-configuration has to be done one at a time.

92

4.6 J-Link USB identification

In general, when using USB, there are two ways in which a J-Link can be identified:

- By serial number
- By USB address

Default configuration of J-Link is: Identification by serial number. Identification via USB address is used for compatibility and not recommended.

Background information

"USB address" really means changing the USB-Product Id (PID).

The following table shows how J-Links enumerate in the different identification modes.

Identification	PID	Serial number	
Serial number (default)	0x0101	Serial number is real serial number of the J-Link or user assigned.	
USB address 0 (Deprecated)	0x0101	123456	
USB address 1 (Deprecated)	0x0102	123456	
USB address 2 (Deprecated)	0x0103	123456	
USB address 3 (Deprecated)	0x0104	123456	
Table 4.1. L. Link enumeration in different identification modes			

Table 4.1: J-Link enumeration in different identification modes

4.6.1 Connecting to different J-Links connected to the same host PC via USB

In general, when having multiple J-Links connected to the same PC, the J-Link to connect to is explicitly selected by its serial number. Most software/debuggers provide an extra field to type-in the serial number of the J-Link to connect to:

The following screenshot shows the connection dialog of the J-Flash software:



The following screenshot shows the connection dialog of IAR EWARM:

Category:		Factory Settings
General Options		
C/C++ Compiler		
Assembler	Setup Connection Breakpoints	
Output Converter		
Custom Build Build Actions	Communication	5000
Linker		5800xxxx
Debugger	○ <u>I</u> CP/IP: IP address ▼	
Simulator	IP address: aaa.bbb.ccc.ddd Serial po:	
Angel	IP address: aaa.bbb.ccc.ddd Serial no:	
GDB Server	Interface JTAG scan chain	
IAR ROM-monitor	● JTAG □ JTAG scan chain with multipl	e targets
J-Link/J-Trace	TAP number: 0	
TI Stellaris	C <u>S</u> WD □ Scan chain contains no	n ARM devices
Macraigor		
PE micro	Preceeding bits: 0	
RDI		
ST-LINK	Log communication	
Third-Party Driver TI XDS100	\$PR0J_DIR\$\cspycomm.log	
11 XD3100		

For debuggers / software which does not provide such a functionality, the J-Link DLL automatically detects that mulliple J-Links are connected to the PC and shows a selection dialog which allows the user to select the appropriate J-Link he wants to connect to.

SEGGER J-Link V4.15y (beta) -	Emulator selection
	Please select the emulator you want to connect to:
	# USB Identification
	O USB O
۵۵ <i>زرانیدا</i> ۱۰ ^۳ ۰۰	1 SN 4294967295
	OK Cancel

So even in IDEs which do not have an selection option for the J-Link, it is possible to connect to different J-Links.

Chapter 5 Working with J-Link and J-Trace

This chapter describes functionality and how to use J-Link and J-Trace.

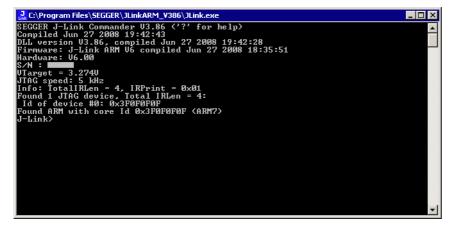
5.1 Connecting the target system

5.1.1 **Power-on sequence**

In general, J-Link / J-Trace should be powered on before connecting it with the target device. That means you should first connect J-Link / J-Trace with the host system via USB and then connect J-Link / J-Trace with the target device via JTAG. Power-on the device after you connected J-Link / J-Trace to it.

5.1.2 Verifying target device connection

If the USB driver is working properly and your J-Link / J-Trace is connected with the host system, you may connect J-Link / J-Trace to your target hardware. Then start JLink.exe which should now display the normal J-Link / J-Trace related information and in addition to that it should report that it found a JTAG target and the target's core ID. The screenshot below shows the output of JLink.exe. As can be seen, it reports a J-Link with one JTAG device connected.



5.1.3 Problems

If you experience problems with any of the steps described above, read the chapter *Support and FAQs* on page 257 for troubleshooting tips. If you still do not find appropriate help there and your J-Link / J-Trace is an original SEGGER product, you can contact SEGGER support via e-mail. Provide the necessary information about your target processor, board etc. and we will try to solve your problem. A checklist of the required information together with the contact information can be found in chapter *Support and FAQs* on page 257 as well.

5.2 Indicators

J-Link uses indicators (LEDs) to give the user some information about the current status of the connected J-Link. All J-Links feature the main indicator. Some newer J-Links such as the J-Link Pro / Ultra come with additional input/output Indicators. In the following, the meaning of these indicators will be explained.

5.2.1 Main indicator

For J-Links up to V7, the main indicator is single color (Green). J-Link V8 comes with a bi-color indicator (Green & Red LED), which can show multiple colors: green, red and orange.

5.2.1.1 Single color indicator (J-Link V7 and earlier)

Indicator status	Meaning
GREEN, flashing at 10 Hz	Emulator enumerates.
GREEN, flickering	Emulator is in operation. Whenever the emulator is exe- cuting a command, the LED is switched off temporarily. Flickering speed depends on target interface speed. At low interface speeds, operations typically take longer and the "OFF" periods are typically longer than at fast speeds.
GREEN, constant	Emulator has enumerated and is in Idle mode.
GREEN, switched off for 10ms once per second	J-Link heart beat. Will be activated after the emulator has been in idle mode for at least 7 seconds.
GREEN, flashing at 1 Hz	Emulator has a fatal error. This should not normally happen.

Table 5.1: J-Link single color main indicator

5.2.1.2 Bi-color indicator (J-Link V8)

Indicator status	Meaning
GREEN, flashing at 10 Hz	Emulator enumerates.
GREEN, flickering	Emulator is in operation. Whenever the emulator is exe- cuting a command, the LED is switched off temporarily. Flickering speed depends on target interface speed. At low interface speeds, operations typically take longer and the "OFF" periods are typically longer than at fast speeds.
GREEN, constant	Emulator has enumerated and is in Idle mode.
GREEN, switched off for 10ms once per second	J-Link heart beat. Will be activated after the emulator has been in idle mode for at least 7 seconds.
ORANGE	Reset is active on target.
RED, flashing at 1 Hz	Emulator has a fatal error. This should not normally happen.

Table 5.2: J-Link single color LED main color indicator

5.2.2 Input indicator

Some newer J-Links such as the J-Link Pro/Ultra come with additional input/output Indicators. The input indicator is used to give the user some information about the status of the target hardware.

5.2.2.1 Bi-color input indicator

Indicator status	Meaning
GREEN	Target voltage could be measured. Target is connected.
ORANGE	Target voltage could be measured. RESET is pulled low (active) on target side.
RED	RESET is pulled low (active) on target side. If no target is connected, reset will be also active on target side.

Table 5.3: J-Link bi-color input indicator

5.2.3 Output indicator

Some newer J-Links such as the J-Link Pro/Ultra come with additional input/output Indicators. The output indicator is used to give the user some information about the emulator-to-target connection.

5.2.3.1 Bi-color output indicator

Indicator status	Meaning
OFF	Target power supply via Pin 19 is not active.
GREEN	Target power supply via Pin 19 is active.
ORANGE	Target power supply via Pin 19 is active. Emulator pulls RESET low (active).
RED	Emulator pulls RESET low (active).

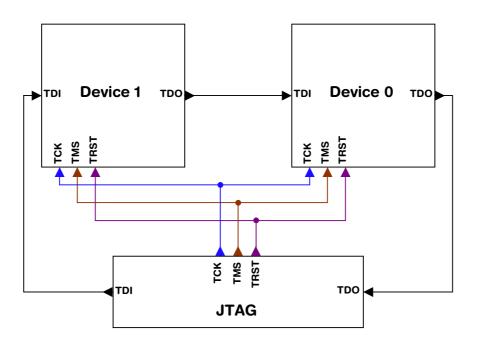
Table 5.4: J-Link bi-color output indicator

5.3 JTAG interface

By default, only one ARM device is assumed to be in the JTAG scan chain. If you have multiple devices in the scan chain, you must properly configure it. To do so, you have to specify the exact position of the ARM device that should be addressed. Configuration of the scan is done by the target application. A target application can be a debugger such as the IAR C-SPY® debugger, ARM's AXD using RDI, a flash programming application such as SEGGER's J-Flash, or any other application using J-Link / J-Trace. It is the application's responsibility to supply a way to configure the scan chain. Most applications offer a dialog box for this purpose.

5.3.1 Multiple devices in the scan chain

J-Link / J-Trace can handle multiple devices in the scan chain. This applies to hardware where multiple chips are connected to the same JTAG connector. As can be seen in the following figure, the TCK and TMS lines of all JTAG device are connected, while the TDI and TDO lines form a bus.



Currently, up to 8 devices in the scan chain are supported. One or more of these devices can be ARM cores; the other devices can be of any other type but need to comply with the JTAG standard.

5.3.1.1 Configuration

The configuration of the scan chain depends on the application used. Read *JTAG interface* on page 100 for further instructions and configuration examples.

5.3.2 Sample configuration dialog boxes

As explained before, it is responsibility of the application to allow the user to configure the scan chain. This is typically done in a dialog box; some sample dialog boxes are shown below.

SEGGER J-Flash configuration dialog

This dialog box can be found at **Options|Project** settings.

Project settings ? 🗙
General Target Interface CPU Flash Production
JTAG
JTAG speed before init JTAG speed after init C Auto selection C Auto selection C Adaptive glocking C Adaptive glocking Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Transmission of the selection Image: Trans
JTAG scan chain information
Devicename ID IRLen TD0 IRLen ID IRLen ID
TDI Add Insert Delete Edit Up Down
OK Cancel Apply

SEGGER J-Link RDI configuration dialog box

This dialog can be found under **RDI|Configure** for example in IAR Embedded Workbench®. For detailed information check the IAR Embedded Workbench user guide.

J-Link RDI Configuration ? 🗙
General Init JTAG Flash Breakpoints CPU
JTAG speed
Image: Second chain with multiple devices Position Image: Second chain with multiple devices Position Image: Second chain with multiple devices 0 is closest to TDI. Image: Second chain with multiple devices 0 is closest to TDI. Sum of IRLens of devices closer to TDI. IIRLen of ARM chips is 4.
<u>⊻erifyJTAG config</u>
OK Cancel Apply

IAR J-Link configuration dialog box

This dialog box can be found under Project | Options.

Options for node "at91:	sam7s-ek"
Category: General Options C/C++ Compiler Assembler Output Converter Custom Build Build Actions Linker Debugger Simulator Angel GDB Server IAR ROM-monitor J-Lint/J-Trace LMI FTDI Macraigor RDI Third-Party Driver	Factory Settings Setup Connection Breakpoints Communication USB Device 0 USB Device 0 ICP/IP asa.bbb.ccc.ddd Interface JTAG scan chain Interface JTAG scan chain JTAG scan chain ITAP number: 0 Scan chain contains non-ARM devices Preceeding bits: 0 Log gommunication \$TOOLKIT_DIR\$\copycomm.log
	OK Cancel

5.3.3 Determining values for scan chain configuration

When do I need to configure the scan chain?

If only one device is connected to the scan chain, the default configuration can be used. In other cases, J-Link / J-Trace may succeed in automatically recognizing the devices on the scan chain, but whether this is possible depends on the devices present on the scan chain.

How do I configure the scan chain?

2 values need to be known:

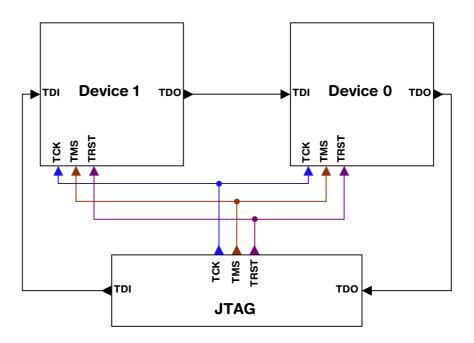
- The position of the target device in the scan chain
- The total number of bits in the instruction registers of the devices before the target device (IR len).

The position can usually be seen in the schematic; the IR len can be found in the manual supplied by the manufacturers of the others devices.

ARM7/ARM9 have an IR len of four.

Sample configurations

The diagram below shows a scan chain configuration sample with 2 devices connected to the JTAG port.



Examples

The following table shows a few sample configurations with 1,2 and 3 devices in different configurations.

Device 0 Chip(IR len)	Device 1 Chip(IR len)	Device 2 Chip(IR len)	Position	IR len
ARM(4)	-	-	0	0
ARM(4)	Xilinx(8)	-	0	0
Xilinx(8)	ARM(4)	-	1	8
Xilinx(8)	Xilinx(8)	ARM(4)	2	16

Table 5.5: Example scan chain configurations

Device 1 Chip(IR len)	Device 2 Chip(IR len)	Position	IR len
Xilinx(8)	ARM(4)	0	0
Xilinx(8)	ARM(4)	2	12
ARM(4)	Xilinx(8)	1	8
	Chip(IR len) Xilinx(8) Xilinx(8)	Chip(IR len)Chip(IR len)Xilinx(8)ARM(4)Xilinx(8)ARM(4)	Chip(IR len)Chip(IR len)PositionXilinx(8)ARM(4)0Xilinx(8)ARM(4)2

Table 5.5: Example scan chain configurations

The target device is marked in blue.

5.3.4 JTAG Speed

There are basically three types of speed settings:

- Fixed JTAG speed
- Automatic JTAG speed
- Adaptive clocking.

These are explained below.

5.3.4.1 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.

5.3.4.2 Automatic JTAG speed

Selects the maximum JTAG speed handled by the TAP controller.

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

5.3.4.3 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.

If you use the adaptive clocking feature, transmission delays, gate delays, and synchronization requirements result in a lower maximum clock frequency than with nonadaptive clocking.

5.4 SWD interface

The J-Link support ARMs Serial Wire Debug (SWD). SWD replaces the 5-pin JTAG port with a clock (SWDCLK) and a single bi-directional data pin (SWDIO), providing all the normal JTAG debug and test functionality. SWDIO and SWCLK are overlaid on the TMS and TCK pins. In order to communicate with a SWD device, J-Link sends out data on SWDIO, synchronous to the SWCLK. With every rising edge of SWCLK, one bit of data is transmitted or received on the SWDIO.

5.4.1 SWD speed

Currently only fixed SWD speed is supported by J-Link. The target is clocked at a fixed clock speed. The SWD speed which is used for target communication should not exceed **target CPU speed * 10**. The maximum SWD speed which is supported by J-Link depends on the hardware version and model of J-Link. For more information about the maximum SWD speed for each J-Link / J-Trace model, please refer to *J*-*Link / J-Trace models* on page 22.

5.4.2 SWO

Serial Wire Output (SWO) support means support for a single pin output signal from the core. The Instrumentation Trace Macrocell (ITM) and Serial Wire Output (SWO) can be used to form a Serial Wire Viewer (SWV). The Serial Wire Viewer provides a low cost method of obtaining information from inside the MCU.

Usually it should not be necessary to configure the SWO speed because this is usually done by the debugger.

5.4.2.1 Max. SWO speeds

The supported SWO speeds depend on the connected emulator. They can be retrieved from the emulator. Currently, the following are supported:

Emulator	Speed formula	Resulting max. speed
J-Link V6	6MHz/n, n >= 12	500kHz
J-Link V7/V8	6MHz/n, n >= 1	6MHz
J-Link Pro	6MHz/n, n >= 1	6MHz

Table 5.6: J-Link supported SWO input speeds

5.4.2.2 Configuring SWO speeds

The max. SWO speed in practice is the max. speed which both, target and J-Link can handle. J-Link can handle the frequencies described in *SWO* on page 105 whereas the max. deviation between the target and the J-Link speed is about 3%.

The computation of possible SWO speeds is typically done in the debugger. The SWO output speed of the CPU is determined by TRACECLKIN, which is normally the same as the CPU clock.

Example1

Target CPU running at 72 MHz. n is be between 1 and 8192.

Possible SWO output speeds are:

72MHz, 36MHz, 24MHz, ...

J-Link V7: Supported SWO input speeds are: 6MHz / n, n>= 1:

6MHz, 3MHz, 2MHz, 1.5MHz, ...

Permitted	combinations are:	

SWO output	SWO input	Deviation percent
6MHz, n = 12	6MHz, n = 1	0
3MHz, n = 24	3MHz, n = 2	0
		<= 3
2MHz, n = 36	2MHz, n = 3	0

Table 5.7: Permitted SWO speed combinations

Example 2

Target CPU running at 10 MHz.

Possible SWO output speeds are:

10MHz, 5MHz, 3.33MHz, ...

J-Link V7: Supported SWO input speeds are: 6MHz / n, n>= 1:

6MHz, 3MHz, 2MHz, 1.5MHz, ...

Permitted combinations are:

SWO output	SWO input	Deviation percent
2MHz, n = 5	2MHz, n = 3	0
1MHz, n = 10	1MHz, n = 6	0
769kHz, n = 13	750kHz, n = 8	2.53
	•••	•••

Table 5.8: Permitted SWO speed combinations

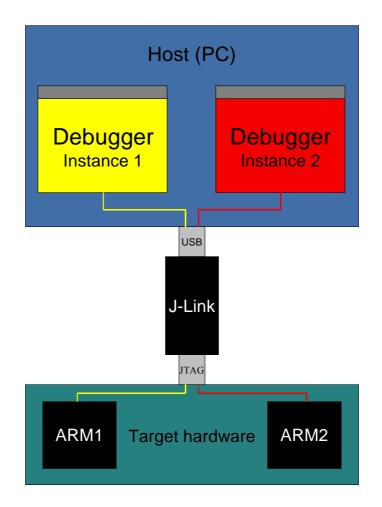
5.5 Multi-core debugging

J-Link / J-Trace is able to debug multiple cores on one target system connected to the same scan chain. Configuring and using this feature is described in this section.

5.5.1 How multi-core debugging works

Multi-core debugging requires multiple debuggers or multiple instances of the same debugger. Two or more debuggers can use the same J-Link / J-Trace simultaneously. Configuring a debugger to work with a core in a multi-core environment does not require special settings. All that is required is proper setup of the scan chain for each debugger. This enables J-Link / J-Trace to debug more than one core on a target at the same time.

The following figure shows a host, debugging two ARM cores with two instances of the same debugger.



Both debuggers share the same physical connection. The core to debug is selected through the JTAG-settings as described below.

5.5.2 Using multi-core debugging in detail

- 1. Connect your target to J-Link / J-Trace.
- 2. Start your debugger, for example IAR Embedded Workbench for ARM.
- 3. Choose Project Options and configure your scan chain. The picture below shows the configuration for the first ARM core on your target.

Options for node "BTL_	AT91_¥430"
Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI Third-Party Driver	Factory Settings Setup Connection Communication Image: Communication Image: USB Image: Communication Image: Image: Communication Image: Communication Image: Image: Communication Image: Communication Image: Communication Image: Communication Image: Communication Image: Communication Image: Communication Image: Communication Image: Communication Image: Communication
	OK Cancel

- 4. Start debugging the first core.
- 5. Start another debugger, for example another instance of IAR Embedded Workbench for ARM.

6. Choose Project | Options and configure your second scan chain. The following dialog box shows the configuration for the second ARM core on your target.

C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI Third-Party Driver Log communication	25
---	----

7. Start debugging your second core.

Example:

Core #1	Core #2	Core #3	TAP number debugger #1	TAP number debugger #2
ARM7TDMI	ARM7TDMI-S	ARM7TDMI	0	1
ARM7TDMI	ARM7TDMI	ARM7TDMI	0	2
ARM7TDM I-S	ARM7TDMI-S	ARM7TDMI-S	1	2

Table 5.9: Multicore debugging

Cores to debug are marked in blue.

5.5.3 Things you should be aware of

Multi-core debugging is more difficult than single-core debugging. You should be aware of the pitfalls related to JTAG speed and resetting the target.

5.5.3.1 JTAG speed

Each core has its own maximum JTAG speed. The maximum JTAG speed of all cores in the same chain is the minimum of the maximum JTAG speeds.

For example:

- Core #1: 2MHz maximum JTAG speed
- Core #2: 4MHz maximum JTAG speed
- Scan chain: 2MHz maximum JTAG speed

5.5.3.2 Resetting the target

All cores share the same RESET line. You should be aware that resetting one core through the RESET line means resetting all cores which have their RESET pins connected to the RESET line on the target.

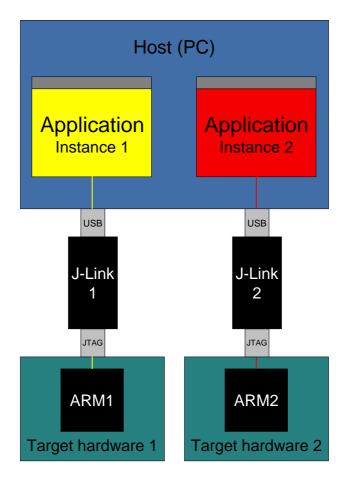
5.6 Connecting multiple J-Links / J-Traces to your PC

In general, it is possible to have an unlimited number of J-Links / J-Traces connected to the same PC. Current J-Link models are already factory-configured to be used in a multi-J-Link environment, older J-Links can be re-configured to use them in a multi-J-link environment.

5.6.1 How does it work?

USB devices are identified by the OS by their product id, vendor id and serial number. The serial number reported by current J-Links is a unique number which allows to have an almost unlimited number of J-Links connected to the same host at the same time.

The sketch below shows a host, running two application programs. Each applicationcommunicates with one ARM core via a separate J-Link.



Older J-Links / J-Traces all reported the same serial number which made it necessary to configure them for USB0-3 if multiple J-Link should be connected to the same PC in parallel.

For these J-Links, we recommend to re-configure them to use the new enumeration method (report real serial number).

Re-configuration can be done by using the J-Link Configurator, which is part of the J-Link software and documentation package.

Re-configuring J-Link to use the new method does not have any bad side-effects on the current debug environment. Usually the user does not see any difference as long as only one emulator is connected.

In order to re-configure a J-Link to use the new USB identification method use the J-Link Configurator which comes with the J-Link software and documentation package. For more information about the J-Link Configurator and how to use it, please refer to *J-Link Configurator* on page 91.

5.7 J-Link control panel

Since software version V3.86 J-Link the J-Link control panel window allows the user to monitor the J-Link status and the target status information in real-time. It also allows the user to configure the use of some J-Link features such as flash download, flash breakpoints and ARM instruction set simulation. The J-Link control panel window can be accessed via the J-Link tray icon in the tray icon list. This icon is available when the debug session is started.

🚅 🔜 13:35

To open the status window, simply click on the tray icon.



5.7.1 Tabs

The J-Link status window supports different features which are grouped in tabs. The organization of each tab and the functionality which is behind these groups will be explained in this section

5.7.1.1 General

In the **General** section, general information about J-Link and the target hardware are shown. Moreover the following general settings can be configured:

- **Show tray icon**: If this checkbox is disabled the tray icon will not show from the next time the DLL is loaded.
- **Start minimized**: If this checkbox is disabled the J-Link status window will show up automatically each time the DLL is loaded.
- **Always on top**: if this checkbox is enabled the J-Link status window is always visible even if other windows will be opened.

The general information about target hardware and J-Link which are shown in this section, are:

- **Process**: Shows the path of the file which loaded the DLL.
- **J-Link**: Shows OEM of the connected J-Link, the hardware version and the Serial number. If no J-Link is connected it shows "not connected" and the color indicator is red.
- **Target interface**: Shows the selected target interface (JTAG/SWD) and the current JTAG speed. The target current is also shown. (Only visible if J-Link is connected)
- Endian: Shows the target endianess (Only visible if J-Link is connected)
- **Device**: Shows the selected device for the current debug session.
- License: Opens the J-Link license manager.
- **About**: Opens the about dialog.

5.7.1.2 Settings

In the **Settings** section project- and debug-specific settings can be set. It allows the configuration of the use of flash download and flash breakpoints and some other target specific settings which will be explained in this topic. Settings are saved in the configuration file. This configuration file needs to be set by the debugger. If the debugger does not set it, settings can not be saved. All settings can only the changed by the user himself. All settings which are modified during the debug session have to be saved by pressing **Save settings**, otherwise they are lost when the debug session is closed.

Section: Flash download

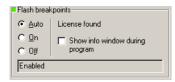
In this section, settings for the use of the J-Link ARM FlashDL feature and related settings can be configured. When a license for J-Link ARM FlashDL is found, the color indicator is green and "License found" appears right to the J-Link ARM FlashDL usage settings.

Flash download				
	License found			
<u>О D</u> n	Skip download on CRC match			
○ 0 <u>f</u> f	Verify download			
Enabled, 10272 bytes downloaded				

- Auto: This is the default setting of J-Link ARM FlashDL usage. If a license is found J-Link ARM FlashDL is enabled. Otherwise J-Link ARM FlashDL will be disabled internally.
- **On:** Enables the J-Link ARM FlashDL feature. If no license has been found an error message appears.
- Off: Disables the J-Link ARM FlashDL feature.
- Skip download on CRC match: J-Link checks the CRC of the flash content to determine if the current application has already been downloaded to the flash. If a CRC match occurs, the flash download is not necessary and skipped. (Only available if J-Link ARM FlashDL usage is configured as **Auto** or **On**)
- Verify download: If this checkbox is enabled J-Link verifies the flash content after the download. (Only available if J-Link ARM FlashDL usage is configured as Auto or On)

Section: Flash breakpoints:

In this section, settings for the use of the FlashBP feature and related settings can be configured. When a license for FlashBP is found, the color indicator is green and "License found" appears right to the FlashBP usage settings.



- Auto: This is the default setting of FlashBP usage. If a license has been found the FlashBP feature will be enabled. Otherwise FlashBP will be disabled internally.
- **On**: Enables the FlashBP feature. If no license has been found an error message appears.
- **Off:** Disables the FlashBP feature.
- Show window during program: When this checkbox is enabled the "Programming flash" window is shown when flash is re-programmed in order to set/clear flash breakpoints.

Flash download and flash breakpoints independent settings

These settings do not belong to the J-Link flash download and flash breakpoints settings section. They can be configured without any license needed.

🛃 SEGGER - Control panel 📃 🖂 🗙
General Settings Breakpoints Log CPU Regs Target Power SWV Device Emulator M
Log file Override C:VLink.log
Settings file Override
Flash download Flash breakpoints C Auto License found C On V Skip download on CRC match C Off Verify download Disabled Disabled
Override device selection Allow caching of flash contents (Dn) Allow instruction set simulation Override memory map Modify breakpoints during execution Allow
Ready JLINKARM_GetSpeed (Done) 1.208 sec. in 32 calls

- **Log file:** Shows the path where the J-Link log file is placed. It is possible to override the selection manually by enabling the Override checkbox. If the Override checkbox is enabled a button appears which let the user choose the new location of the log file.
- **Settings file**: Shows the path where the configuration file is placed. This configuration file contains all the settings which can be configured in the **Settings** tab.
- **Override device selection**: If this checkbox is enabled, a dropdown list appears, which allows the user to set a device manually. This especially makes sense when J-Link can not identify the device name given by the debugger or if a particular device is not yet known to the debugger, but to the J-Link software.
- Allow caching of flash contents: If this checkbox is enabled, the flash contents are cached by J-Link to avoid reading data twice. This speeds up the transfer between debugger and target.
- Allow instruction set simulation: If this checkbox is enabled, ARM instructions will be simulated as far as possible. This speeds up single stepping, especially when FlashBPs are used.
- **Save settings**: When this button is pushed, the current settings in the **Settings** tab will be saved in a configuration file. This file is created by J-Link and will be created for each project and each project configuration (e.g. Debug_RAM, Debug_Flash). If no settings file is given, this button is not visible.
- **Modify breakpoints during execution:** This dropdown box allows the user to change the behavior of the DLL when setting breakpoints if the CPU is running. The following options are available:

Allow: Allows settings breakpoints while the CPU is running. If the CPU needs to be halted in order to set the breakpoint, the DLL halts the CPU, sets the breakpoints and restarts the CPU.

Allow if CPU does not need to be halted: Allows setting breakpoints while the CPU is running, if it does not need to be halted in order to set the breakpoint. If the CPU has to be halted the breakpoint is not set.

Ask user if CPU needs to be halted: If the user tries to set a breakpoint while the CPU is running and the CPU needs to be halted in order to set the breakpoint, the user is asked if the breakpoint should be set. If the breakpoint can be set without halting the CPU, the breakpoint is set without explicitly confirmation by the user.

Do not allow: It is not allowed to set breakpoints while the CPU is running.

5.7.1.3 Break/Watch

In the Break/Watch section all breakpoints and watchpoints which are in the DLL internal breakpoint and watchpoint list are shown.

J	SEGGER J-Link ARM - Control panel						
G	ene	ral Settings	Break/Watch	log ∫CPU	Regs Target Power	SWV Device Emulator	
В	real	kpoints:					
	#	Handle	Address	Mode	Permission	Implementation	
	1		0x0800011C	Unknown	Any	Flash - TBC	
	2		0x08000128	Unknown	Any	Flash - TBC	
	3		0x08000124	Unknown	Any	Flash - TBC	
	4		0x0800013A	Unknown	Any	Flash - TBC	
	5		0x08000150	Unknown	Any	Flash - TBC	
	6		0x0800016A	Unknown	Any	Flash - TBC	
	Vato #	chpoints: Handle	Address		Data	Access	_
l h	+	0x8000000*	0x08000120		0x00001000		
	·	0x8000000"	0x08000120		0x00001000	Write, 16-bit	
V	'ecti	or catch:					
	#	Vector					
Leady JLINKARM_ReadMem (Done) 1.494 sec. in 219 calls							

Section: Code

Lists all breakpoints which are in the DLL internal breakpoint list are shown.

- Handle: Shows the handle of the breakpoint.
- **Address**: Shows the address where the breakpoint is set.
- **Mode**: Describes the breakpoint type (ARM/THUMB)
- **Permission**: Describes the breakpoint implementation flags.
- **Implementation**: Describes the breakpoint implementation type. The breakpoint types are: RAM, Flash, Hard. An additional TBC (to be cleared) or TBS (to be set) gives information about if the breakpoint is (still) written to the target or if it's just in the breakpoint list to be written/cleared.

Note: It is possible for the debugger to bypass the breakpoint functionality of the J-Link software by writing to the debug registers directly. This means for ARM7/ ARM9 cores write accesses to the ICE registers, for Cortex-M3 devices write accesses to the memory mapped flash breakpoint registers and in general simple write accesses for software breakpoints (if the program is located in RAM). In these cases, the J-Link software can not determine the breakpoints set and the list is empty.

Section: Data

In this section, all data breakpoints which are listed in the DLL internal breakpoint list are shown.

- **Handle**: Shows the handle of the data breakpoint.
- Address: Shows the address where the data breakpoint is set.
- AddrMask: Specifies which bits of Address are disregarded during the comparison for a data breakpoint match. (A 1 in the mask means: disregard this bit)
- **Data**: Shows on which data to be monitored at the address where the data breakpoint is set.
- Data Mask: Specifies which bits of Data are disregarded during the comparison
- for a data breakpoint match. (A 1 in the mask means: disregard this bit)
- **Ctrl**: Specifies the access type of the data breakpoint (read/write).
- **CtrlMask**: Specifies which bits of Ctrl are disregarded during the comparison for a data breakpoint match.

5.7.1.4 Log

In this section the log output of the DLL is shown. The user can determine which function calls should be shown in the log window.

Available function calls to log: Register read/write, Memory read/write, set/clear breakpoint, step, go, halt, is halted.

General Settings Break/Watch Log CPU Regs Target Power SWV	
Register write Memory write V BP set V Step V Halt	1
🔲 Register read 🔲 Memory read 🔽 BP clear 🔽 Go 📄 IsHalted	<u>C</u> lear log
J-Link ARM U3.85i (beta) DLL Log DLL Compiled: Jun 26 2008 17:06:33 Logging started @ 2008-06-27 15:00	A
Test 288 560 JLNKGR Halt() Test 288 560 JLNKGR Halt() Te	
<u>л</u>	T T
Ready	

5.7.1.5 CPU Regs

In this section the name and the value of the CPU registers are shown.

J	J-Link i	ARM			
	General	Settings Break/W	atch Log CPU Regs	Target Power SWV	
	Index	Name	Value	State	▲ ·
	0	RO	0x0010269C		
	1	R1	0x00000050		
	2	R2	0x00000010		
	3	R3	0x0000003		
	4	R4	0x00201100		
	5	R5	0x0000000		
	6	R6	0x0000000		
	7	R7	0x00000000		
	8	CPSR D15 (DC)	0x80000053		
	9 10	R15 (PC)	0x00100FB8 0x00000000		
	11	R8_USR R9_USR	0x00000000		
	12	R10_USR	0x00000000		
	13	R11_USR	0x00000000		
	14	R12_USR	0x00000002		
	15	R13_USR	0x00000000		
	16	R14_USR	0x00000000		-
	,				
Re	ady				

5.7.1.6 Target Power

In this section currently just the power consumption of the target hardware is shown.

J-Link ARM					_ 🗆 X
General Setting	gs Break/Watch	Log CPU Regs	Target Power	SWV Device	MemMap
Current statu: C Power gr C Power dis	abled sabled	Permanent status C Power enabled © Power <u>d</u> isabled			
 Power inform Consumption 					
Ready	JLINKARM_ExecC	ommand (Done)		0.008 sec. in 20 c	alls

5.7.1.7 SWV

In this section SWV information are shown.

J-Link ARM			_ 🗆 X
General Settings	Break/Watch Log CPU Reg	s Target Power SWV	
Status	UART encoding, 19200 bps	Host buffer 4 MB	
Bytes in buffer	0 bytes	Emulator buffer 4 KB	
Bytes transferred	23570 bytes		
Refresh counter	1522		
Ready			

- **Status**: Shows the encoding and the baudrate of the SWV data received by the target (Manchester/UART, currently J-Link only supports UART encoding).
- Bytes in buffer: Shows how many bytes are in the DLL SWV data buffer.
- **Bytes transferred**: Shows how many bytes have been transferred via SWV, since the debug session has been started.
- **Refresh counter**: Shows how often the SWV information in this section has been updated since the debug session has been started.
- **Host buffer**: Shows the reserved buffer size for SWV data, on the host side.
- **Emulator buffer**: Shows the reserved buffer size for SWV data, on the emulator side.

5.8 Reset strategies

J-Link / J-Trace 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. For example reset strategies which use the reset pin can not succeed on targets where the reset pin of the CPU is not connected to the reset pin of the JTAG connector. Reset strategy 0 is always the recommended one because it has been adapted to work on every target even if the reset pin (Pin 15) is not connected.

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

The instructions which are 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 components, 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.

5.8.1 Strategies for ARM 7/9 devices

5.8.1.1 Type 0: 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.

This reset strategy is typically used if nRESET and nTRST are coupled. If nRESET and nTRST are coupled, either on the board or the CPU itself, reset clears the breakpoint, which means that the CPU can not be stopped after reset with the BP@0 reset strategy.

5.8.1.2 Type 1: 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.

5.8.1.3 Type 2: Software, for Analog Devices ADuC7xxx MCUs

This reset strategy is a software strategy. The CPU is halted and performs a sequence which causes a peripheral reset. 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 the recommended reset sequence for Analog Devices ADuC7xxx MCUs and works with these chips only.

5.8.1.4 Type 3: No reset

No reset is performed. Nothing happens.

5.8.1.5 Type 4: Hardware, halt with WP

The hardware RESET pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU using a watchpoint. 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

5.8.1.6 Type 5: Hardware, halt with DBGRQ

The hardware RESET pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU using the DBGRQ. 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.

5.8.1.7 Type 6: Software

This reset strategy is only a software reset. "Software reset" means basically no reset, just changing the CPU registers such as PC and CPSR. This reset strategy sets the CPU registers to their after-Reset values:

- PC = 0
- CPSR = 0xD3 (Supervisor mode, ARM, IRQ / FIQ disabled)
- All SPSR registers = 0x10
- All other registers (which are unpredictable after reset) are set to 0.
- The hardware RESET pin is not affected.

5.8.1.8 Type 7: Reserved

Reserved reset type.

5.8.1.9 Type 8: Software, for ATMEL AT91SAM7 MCUs

The reset pin of the device is disabled by default. This means that the reset strategies which rely on the reset pin (low pulse on reset) do not work by default. For this reason a special reset strategy has been made available.

It is recommended to use this reset strategy. This special reset strategy resets the peripherals by writing to the RSTC_CR register. Resetting the peripherals puts all peripherals in the defined reset state. This includes memory mapping register, which means that after reset flash is mapped to address 0. It is also possible to achieve the same effect by writing 0x4 to the RSTC_CR register located at address 0xffffd00.

5.8.1.10 Type 9: Hardware, for NXP LPC MCUs

After reset a bootloader is mapped at address 0 on ARM 7 LPC devices. This reset strategy performs a reset via reset strategy Type 1 in order to reset the CPU. It also ensures that flash is mapped to address 0 by writing the MEMMAP register of the LPC. This reset strategy is the recommended one for all ARM 7 LPC devices.

5.8.2 Strategies for Cortex-M devices

J-Link supports different specific reset strategies for the Cortex-M cores. All of the following reset strategies are available in JTAG and in SWD mode. All of them halt the CPU after the reset.

5.8.2.1 Type 0: Normal

This is the default strategy. It works well for most Cortex-M devices. J-Link tries to reset both, core and peripherals by setting the SYSRESETREQ & VECTRESET bits in the AIRCR. The VC_CORERESET bit is used to halt the CPU before it executes a single instruction.

On devices that are known to have a bootloader, this bootloader is started after the core & peripherals have been reset and stopped before trying to start the application program, thus ensuring that the bootloader (which may perform important initialisations) has a chance to do so.

This type of RESET can fail:

One reason is that the CPU is in power down state. In this case, the reset pin is used to reset the device. If this fails as well, then Connect-under-Reset is executed.

Other reasons why the initial reset may not work are typically shortcomings in the silicon (sometimes only in Beta silicon). Some of these reasons are:

- Watchdog continues to run when CPU is halted
- SYSRESETREQ also reset debug unit

5.8.2.2 Type 1: Core

Only the core is reset via the VECTRESET bit. The peripherals are not affected. After setting the VECTRESET bit, J-Link waits for the S_RESET_ST bit in the Debug Halting Control and Status Register (DHCSR) to first become high and then low afterwards. The CPU does not start execution of the program because J-Link sets the VC_CORERESET bit before reset, which causes the CPU to halt before execution of the first instruction.

5.8.2.3 Type 2: ResetPin

J-Link pulls its RESET pin low to reset the core and the peripherals. This normally causes the CPU RESET pin of the target device to go low as well, resulting in a reset of both CPU and peripherals. This reset strategy will fail if the RESET pin of the target device is not pulled low. The CPU does not start execution of the program because J-Link sets the VC_CORERESET bit before reset, which causes the CPU to halt before execution of the first instruction.

5.8.2.4 Type 3: Connect under Reset

J-Link connects to the target while keeping Reset active (reset is pulled low and remains low while connecting to the target). This is the recommended reset strategy for STM32 devices. This reset strategy has been designed for the case that communication with the core is not possible in normal mode so the VC_CORERESET bit can not be set in order to guarantee that the core is halted immediately after reset.

5.8.2.5 Type 4: Reset core & peripherals, halt after bootloader

Same as type 0, but bootloader is always executed. This reset strategy has been designed for MCUs/CPUs which have a bootloader located in ROM which needs to run at first, after reset (since it might initialize some target settings to their reset state). When using this reset strategy, J-Link will let the bootloader run after reset and halts the target immediately after the bootloader and before the target application is started. This is the recommended reset strategy for LPC11xx and LPC13xx devices where a bootloader should execute after reset to put the chip into the "real" reset state.

5.8.2.6 Type 5: Reset core & peripherals, halt before bootloader

Same as Type 0, but bootloader is never executed. Not normally used, except in situations where the bootloader needs to be debugged.

5.8.2.7 Type 6: Reset for Freescale Kinetis devices

Performs a via reset strategy 0 (normal) first in order to reset the core & peripherals and halt the CPU immediately after reset. After the CPU is halted, the watchdog is disabled, since the watchdog is running after reset by default and if the target application does not feed the watchdog, J-Link loses connection to the device since it is reset permanently.

5.8.2.8 Type 7: Reset for Analog Devices CPUs (ADI Halt after kernel)

Performs a reset of the core and peripherals by setting the SYSRESETREQ bit in the AIRCR. The core is allowed to perform the ADI kernel (which enables the debug interface) but the core is halted before the first instruction after the kernel is executed in order to guarantee that no user application code is performed after reset.

Type 8: Reset core and peripherals

J-Link tries to reset both, core and peripherals by setting the SYSRESETREQ bit in the AIRCR. The VC_CORERESET bit is used to halt the CPU before it executes a single instruction.

5.8.2.9 Type 9: Reset for LPC1200 devices

On the NXP LPC1200 devices the watchdog is enabled after reset and not disabled by the bootloader, if a valid application is in the flash memory. Moreover, the watchdog keeps counting if the CPU is in debug mode. When using this reset strategy, J-Link performs a reset of the CPU and peripherals, using the SYSRESETREQ bit in the AIRCR and halts the CPU after the bootloader has been performed and before the first instruction of the user code is executed. Then the watchdog of the LPC1200 device is disabled. This reset strategy is only guaranteed to work on "modern" J-Links (J-Link V8, J-Link Pro, J-Link Ultra, J-Trace for Cortex-M, J-Link Lite) and if a SWD speed of min. 1 MHz is used. This reset strategy should also work for J-Links with hardware version 6, but it can not be guaranteed that these J-Links are always fast enough in disabling the watchdog.

5.8.2.10 Type 10: Reset for Samsung S3FN60D devices

On the Samsung S3FN60D devices the watchdog may be running after reset (if the watchdog is active after reset or not depends on content of the smart option bytes at addr 0xC0). The watchdog keeps counting even if the CPU is in debug mode (e.g. halted by a halt request or halted by vector catch). When using this reset strategy, J-Link performs a reset of the CPU and peripherals, using the SYSRESETREQ bit and sets VC_CORERESET in order to halt the CPU after reset, before it executes a single instruction. Then the watchdog of the S3FN60D device is disabled.

5.9 Using DCC for memory access

The ARM7/9 architecture requires cooperation of the CPU to access memory when the CPU is running (not in debug mode). This means that memory can not normally be accessed while the CPU is executing the application program. The normal way to read or write memory is to halt the CPU (put it into debug mode) before accessing memory. Even if the CPU is restarted after the memory access, the real time behavior is significantly affected; halting and restarting the CPU costs typically multiple milliseconds. For this reason, most debuggers do not even allow memory access if the CPU is running.

Fortunately, there is one other option: DCC (Direct communication channel) can be used to communicate with the CPU while it is executing the application program. All that is required is that the application program calls a DCC handler from time to time. This DCC handler typically requires less than 1 μ s per call.

The DCC handler, as well as the optional DCC abort handler, is part of the J-Link software package and can be found in the $Samples \ DCC \ IAR$ directory of the package.

5.9.1 What is required?

- An application program on the host (typically a debugger) that uses DCC
- A target application program that regularly calls the DCC handler
- The supplied abort handler should be installed (optional)

An application program that uses DCC is <code>JLink.exe</code>.

5.9.2 Target DCC handler

The target DCC handler is a simple C-file taking care of the communication. The function DCC_Process() needs to be called regularly from the application program or from an interrupt handler. If a RTOS is used, a good place to call the DCC handler is from the timer tick interrupt. In general, the more often the DCC handler is called, the faster memory can be accessed. On most devices, it is also possible to let the DCC generate an interrupt which can be used to call the DCC handler.

5.9.3 Target DCC abort handler

An optional DCC abort handler (a simple assembly file) can be included in the application. The DCC abort handler allows data aborts caused by memory reads/writes via DCC to be handled gracefully. If the data abort has been caused by the DCC communication, it returns to the instruction right after the one causing the abort, allowing the application program to continue to run. In addition to that, it allows the host to detect if a data abort occurred.

In order to use the DCC abort handler, 3 things need to be done:

- Place a branch to DCC_Abort at address 0x10 ("vector" used for data aborts)
- Initialize the Abort-mode stack pointer to an area of at least 8 bytes of stack memory required by the handler
- Add the DCC abort handler assembly file to the application

5.10 J-Link script files

In some situations it it necessary to customize some actions performed by J-Link. In most cases it is the connection sequence and/or the way in which a reset is performed by J-Link, since some custom hardware needs some special handling which can not be integrated into the generic part of the J-Link software. J-Link script files are written in C-like syntax in order to have an easy start to learning how to write J-Link script files. The script file syntax does support most statements (if-else, while, declaration of variables, ...) which are allowed in C, but not all of them. Moreover, there are some statements that are script file specific. The script file allows maximum flexibility, so almost any target initialization which is necessary, can be supported.

5.10.1 Actions that can be customized

The script file support allows customizing of different actions performed by J-Link. If an generic-implemented action is replaced by an action defined in a script file depends on if the corresponding function is present in the script file. In the following all J-Link actions which can be customized using a script file, are listed and explained.

5.10.1.1 ResetTarget()

Decsription

If present, it replaces the reset strategy performed by the DLL when issuing a reset.

Prototype

void ResetTarget(void);

5.10.1.2 InitEMU()

Decsription

If present, it allows configuration of the emulator prior to starting target communication. Currently this function is only used to configure if the target which is connected to J-Link has an ETB or not. For more information how to configure the existence of an ETB, please refer to *Global DLL variables* on page 128.

Prototype

void InitEMU(void);

5.10.1.3 InitTarget()

Decsription

If present, it can replace the auto-detection capability of J-Link. Some targets can not be auto-detected by J-Link since some special target initialization is necessary before communication with the core is possible. Moreover, J-Link uses a TAP reset to get the JTAG IDs of the devices in the JTAG chain. On some targets this disables access to the core.

Prototype

void InitTarget(void);

5.10.2 Script file API functions

In the following, the API functions which can be used in a script file to communicate with the DLL are explained.

5.10.2.1 MessageBox()

Description

Outputs a string in a message box.

Prototype

___api___ int MessageBox(const char * sMsg);

5.10.2.2 MessageBox1()

Description

Outputs a constant character string in a message box. In addition to that, a given value (can be a constant value, the return value of a function or a variable) is added, right behind the string.

Prototype

```
__api__ int MessageBox1(const char * sMsg, int v);
```

5.10.2.3 Report()

Description

Outputs a constant character string on stdio.

Prototype

__api__ int Report(const char * sMsg);

5.10.2.4 Report1()

Description

Outputs a constant character string on stdio. In addition to that, a given value (can be a constant value, the return value of a function or a variable) is added, right behind the string.

Prototype

__api__ int Report1(const char * sMsg, int v);

5.10.2.5 JTAG_SetDeviceId()

Description

Sets the JTAG Id of a specified device, in the JTAG chain. The index of the device depends on its position in the JTAG chain. The device closest to TDO has index 0. The Id is used by the DLL to recognize the device.

Before calling this function, please make sure that the JTAG chain has been configured correctly by setting the appropriate global DLL variables. For more information about the known global DLL variables, please refer to *Global DLL variables* on page 128.

Prototype

___api___ int JTAG_SetDeviceId(int DeviceIndex, unsigned int Id);

5.10.2.6 JTAG_GetDeviceId()

Description

Retrieves the JTAG Id of a specified device, in the JTAG chain. The index of the device depends on its position in the JTAG chain. The device closest to TDO has index 0.

Prototype

___api___ int JTAG_GetDeviceId(int DeviceIndex);

5.10.2.7 JTAG_WriteIR()

Description

Writes a JTAG instruction.

Before calling this function, please make sure that the JTAG chain has been configured correctly by setting the appropriate global DLL variables. For more information about the known global DLL variables, please refer to *Global DLL variables* on page 128.

Prototype

___api___ int JTAG_WriteIR(unsigned int Cmd);

5.10.2.8 JTAG_StorelR()

Description

Stores a JTAG instruction in the DLL JTAG buffer.

Before calling this function, please make sure that the JTAG chain has been configured correctly by setting the appropriate global DLL variables. For more information about the known global DLL variables, please refer to *Global DLL variables* on page 128.

Prototype

___api___ int JTAG_StoreIR(unsigned int Cmd);

5.10.2.9 JTAG_WriteDR()

Description

Writes JTAG data.

Before calling this function, please make sure that the JTAG chain has been configured correctly by setting the appropriate global DLL variables. For more information about the known global DLL variables, please refer to *Global DLL variables* on page 128.

Prototype

___api___ int JTAG_WriteDR(unsigned ___int64 tdi, int NumBits);

5.10.2.10JTAG_StoreDR()

Description

Stores JTAG data in the DLL JTAG buffer.

Before calling this function, please make sure that the JTAG chain has been configured correctly by setting the appropriate global DLL variables. For more information about the known global DLL variables, please refer to *Global DLL variables* on page 128.

Prototype

___api___ int JTAG_StoreDR(unsigned ___int64 tdi, int NumBits);

5.10.2.11JTAG_Write()

Description

Writes a JTAG sequence (max. 64 bits per pin).

Prototype

```
__api__ int JTAG_Write(unsigned __int64 tms, unsigned __int64 tdi, int NumBits);
```

5.10.2.12JTAG_Store()

Description

Stores a JTAG seuqnece (max. 64 bits per pin) in the DLL JTAG buffer.

Prototype

```
__api__ int JTAG_Store(unsigned __int64 tms, unsigned __int64 tdi, int NumBits);
```

5.10.2.13JTAG_GetU32()

Description

Gets 32 bits JTAG data, starting at given bit position.

Prototype

___api___ int JTAG_GetU32(int BitPos);

5.10.2.14JTAG_WriteClocks()

Description

Writes a given number of clocks.

Prototype

__api__ int JTAG_WriteClocks(int NumClocks);

5.10.2.15JTAG_StoreClocks()

Description

Stores a given number of clocks in the DLL JTAG buffer.

Prototype

___api___ int JTAG_StoreClocks(int NumClocks);

5.10.2.16JTAG_Reset()

Description

Performs a TAP reset and tries to auto-detect the JTAG chain (Total IRLen, Number of devices). If auto-detection was successful, the global DLL variables which determine the JTAG chain configuration, are set to the correct values. For more information about the known global DLL variables, please refer to *Global DLL variables* on page 128.

Note: This will not work for devices which need some special init (for example to add the core to the JTAG chain), which is lost at a TAP reset.

Prototype

__api__ int JTAG_Reset(void);

5.10.2.17SYS_Sleep()

Description

Waits for a given number of miliseconds. During this time, J-Link does not communicate with the target.

Prototype

__api__ int SYS_Sleep(int Delayms);

5.10.2.18CORESIGHT_AddAP()

Description

Allows the user to manually configure the AP-layout of the device J-Link is connected to. This makes sense on targets where J-Link can not perform a auto-detection of the APs which are present of the target system. Type can only be a known global J-Link DLL AP constant. For a list of all available constants, please refer to *Global DLL constants* on page 131.

Prototype

___api___ int CORESIGHT_AddAP(int Index, unsigned int Type);

Example

```
CORESIGHT_AddAP(0, CORESIGHT_AHB_AP); // First AP is a AHB-AP
CORESIGHT_AddAP(1, CORESIGHT_APB_AP); // Second AP is a APB-AP
CORESIGHT_AddAP(2, CORESIGHT_JTAG_AP); // Third AP is a JTAG-AP
```

5.10.3 Global DLL variables

The script file feature also provides some global variables which are used for DLL configuration. Some of these variables can only be set to some specifc values, other ones can be set to the whole datatype with. In the following all global variables and their value ranges are listed and described.

Note: All global variables are treated as unsigned 32-bit values and are zero-initialized.

Variable	Description	R/W
СРИ	Pre-selct target CPU J-Link is communicating with. Used in InitTarget() to skip the core auto- detection of J-Link. This variable can only be set to a known global J-Link DLL constant. For a list of all valid values, please refer to <i>Global DLL con-</i> <i>stants</i> on page 131. Example CPU = ARM926EJS;	W
JTAG_IRPre	Used for JTAG chain configuration. Sets the num- ber of IR-bits of all devices which are closer to TDO than the one we want to communicate with. Example JTAG_IRPre = 6;	R/W
JTAG_DRPre	Used for JTAG chain configuration. Sets the num- ber of devices which are closer to TDO than the one we want to communicate with. Example JTAG_DRPre = 2;	R
JTAG_IRPost	Used for JTAG chain configuration. Sets the num- ber of IR-bits of all devices which are closer to TDI than the one we want to communicate with. Example JTAG_IRPost = 6;	R
JTAG_DRPost	Used for JTAG chain configuration. Sets the num- ber of devices which are closer to TDI than the one we want to "communicate with. Example JTAG_DRPost = 0;	R

Table 5.10: Global DLL variables

Variable	Description	R/W
JTAG_IRLen	IR-Len (in bits) of the device we want to commu- nicate with. Example JTAG_IRLen = 4;	R
JTAG_TotalIRLen	<pre>Computed automatically, based on the values of JTAG_IRPre, JTAG_DRPre, JTAG_IRPost and JTAG_DRPost. Example v = JTAG_TotalIRLen;</pre>	R
JTAG_AllowTAPReset	En-/Disables auto-JTAG-detection of J-Link. Has to be disabled for devices which need some spe- cial init (for example to add the core to the JTAG chain), which is lost at a TAP reset. Allowed values 0 Auto-detection is enabled. 1 Auto-detection is disabled.	w
JTAG_Speed	Sets the JTAG interface speed. Speed is given in kHz. Example JTAG_Speed = 2000; // 2MHz JTAG speed	W
JTAG_ResetPin	<pre>Pulls reset pin low / Releases nRST pin. Used to issue a reset of the CPU. Value assigned to reset pin reflects the state. 0 = Low, 1 = high. Example JTAG_ResetPin = 0; SYS_Sleep(5); // Give pin some time to get low JTAG_ResetPin = 1;</pre>	W
JTAG_TRSTPin	<pre>Pulls reset pin low / Releases nTRST pin. Used to issue a reset of the debug logic of the CPU. Value assigned to reset pin reflects the state. 0 = Low, 1 = high. Example JTAG_TRSTPin = 0; SYS_Sleep(5); // Give pin some time to get low JTAG_TRSTPin = 1;</pre>	w
JTAG_TCKPin	Pulls TCK pin LOW / HIGH. Value assigned to reset pin reflects the state. 0 = LOW, 1 = HIGH. Example JTAG_TCKPin = 0;	R/W
JTAG_TDIPin	Pulls TDI pin LOW / HIGH. Value assigned to reset pin reflects the state. 0 = LOW, 1 = HIGH. Example JTAG_TDIPin = 0;	R/W
JTAG_TMSPin	Pulls TMS pin LOW / HIGH. Value assigned to reset pin reflects the state. 0 = LOW, 1 = HIGH. Example JTAG_TMSPin = 0;	R/W
EMU_ETB_IsPresent Table 5.10: Global DLL variable	<pre>If the connected device has an ETB and you want to use it with J-link, this variable should be set to 1. Setting this variable in another function as InitEmu() does not have any effect. Example void InitEmu(void) { EMU_ETB_IsPresent = 1; }</pre>	W

Table 5.10: Global DLL variables

Variable	Description	R/W
EMU_ETB_USeETB	Use ETB instead of RAWTRACE capability of the emulator. Setting this variable in another func- tion as InitEmu() does not have any effect. Example EMU_ETB_USETB = 0;	R
EMU_ETM_IsPresent	<pre>Selects whether an ETM is present on the target or not. Setting this variable in another function as InitEmu() does not have any effect. Example EMU_ETM_IsPresent= 0;</pre>	R/W
EMU_ETM_USeETM	<pre>Use ETM as trace source. Setting this variable in another function as InitEmu() does not have any effect. Example EMU_ETM_USETM = 1;</pre>	w
EMU_JTAG_ DisableHWTransmissions	Disable use of hardware units for JTAG transmis- sions since this can cause problems on some hardware designs. Example EMU_JTAG_DisableHWTransmissions = 1;	w
CORESIGHT_CoreBaseAddr	Set base address of core debug component for CoreSight compliant devices. Setting this vari- able disables the J-Link auto-detection of the core debug component base address. Used on devices where auto-detection of the core debug component base address is not possible due to incorrect CoreSight information. Example CORESIGHT_CoreBaseAddr = 0x80030000;	R/W
CORESIGHT_ IndexAHBAPToUse	<pre>Pre-select an AP as an AHB-AP that J-Link uses for debug communication (Cortex-M). Setting this variable is necessary for example when debugging multi-core devices where multiple AHB-APs are present (one for each device). This function can only be used if a AP-layout has been configured via CORESIGHT_AddAP(). Example CORESIGHT_AddAP(0, CORESIGHT_AHB_AP); CORESIGHT_AddAP(1, CORESIGHT_AHB_AP); CORESIGHT_AddAP(2, CORESIGHT_AHB_AP); CORESIGHT_AddAP(2, CORESIGHT_APB_AP); // // Use second AP as AHB-AP // for target communication // CORESIGHT_IndexAHBAPTOUSe = 1;</pre>	W

Table 5.10: Global DLL variables

Variable	Description	R/W
CORESIGHT_ IndexAPBAPToUse	<pre>Pre-select an AP as an APB-AP that J-Link uses for debug communication (Cortex-A/R). Setting this variable is necessary for example when debugging multi-core devices where multiple APB-APs are present (one for each device). This function can only be used if a AP-layout has been configured via CORESIGHT_AddAP(). Example CORESIGHT_AddAP(0, CORESIGHT_AHB_AP); CORESIGHT_AddAP(1, CORESIGHT_APB_AP); CORESIGHT_AddAP(2, CORESIGHT_APB_AP); // // Use third AP as APB-AP // for target communication // CORESIGHT_IndexAPBAPToUse = 2;</pre>	W
MAIN_ResetType	<pre>Used to determine what reset type is currently selected by the debugger. This is useful, if the script has to behave differently if a specific reset type is selected by the debugger and the script file has a ResetTarget() function which over- rides the J-Link reset strategies. Example if (MAIN_ResetType == 2) { [] } else { [] }</pre>	R
MAIN_IsFirstIdentify Table 5.10: Global DLL variable	<pre>Used to check if this is the first time we are run- ning into InitTarget(). Useful if some init steps only need to be executed once per debug ses- sion.Example if (MAIN_IsFirstIdentify == 1) { [] } else { [] }</pre>	R

Table 5.10: Global DLL variables

5.10.4 Global DLL constants

Currently there are only global DLL constants to set the global DLL variable $_{\tt CPU}.$ If necessary, more constants will be implemented in the future.

5.10.4.1 Constants for global variable: CPU

The following constants can be used to set the global DLL variable CPU:

- ARM7
- ARM7TDMI
- ARM7TDMIR3
- ARM7TDMIR4
- ARM7TDMIS
- ARM7TDMISR3
- ARM7TDMISR4
- ARM9
- ARM9TDMIS
- ARM920T
- ARM922T
- ARM926EJS
- ARM946EJS

- ARM966ES
- ARM968ES
- ARM11
- ARM1136
- ARM1136J
- ARM1136JS
- ARM1136JF
- ARM1136JFS
- ARM1156
- ARM1176
- ARM1176J
 ARM1176JS
- ARM1176JS
 ARM1176IF
- ARM1176IF
- ARM1176JFS
 CORTEX_M0
- CORTEX_M0
 CORTEX_M1
- CORTEX_M1
 CORTEX_M3
- CORTEX_M3R1P0
- CORTEX_M3R1P1
- CORTEX_M3R2P0
- CORTEX_M4
- CORTEX_A5
- CORTEX_A8
- CORTEX_A9
- CORTEX_R4
- CORESIGHT_AHB_AP
- CORESIGHT_APB_AP
- CORESIGHT_JTAG_AP
- CORESIGHT_CUSTOM_AP

5.10.5 Script file language

The syntax of the J-Link script file language follows the conventions of the C-language, but it does not support all expressions and operators which are supported by the C-language. In the following, the supported operators and expressions are listed.

5.10.5.1 Supported Operators

The following operators are supported by the J-Link script file language:

- Multiplicative operators: *, /, %
- Additive operators: +, -
- Bitwise shift operators: <<, >>)
- Relational operators: <, >, <=, >=
- Equality operators: ==, !=
- Bitwise operators: &, |, ^
- Logical operators: &&, ||
- Assignment operators: =, *=, /=, +=, -=, <<=, >>=, &=, ^=, |=

5.10.5.2 Supported type specifiers

The following type specifiers are supported by the J-Link script file language:

- void
- char
- int (32-bit)
- ___int64

5.10.5.3 Supported type qualifiers

The following type qualifiers are supported by the J-Link script file language:

- const
- signed

unsigned

5.10.5.4 Supported declarators

The following type qualifiers are supported by the J-Link script file language:

• Array declarators

5.10.5.5 Supported selection statements

The following selection statements are supported by the J-Link script file language:

- if-statements
- if-else-statements

5.10.5.6 Supported iteration statements

The following iteration statements are supported by the J-Link script file language:

- while
- do-while

5.10.5.7 Jump statements

The following jump statements are supported by the J-Link script file language:

• return

5.10.5.8 Sample script files

The J-Link software and documentation package comes with sample script files for different devices. The sample script files can be found at *\$JLINK_INST_DIR\$\Samples\JLink\Scripts*.

5.10.6 Script file writing example

In the following, a short example how a J-Link script file could look like. In this example we assume a JTAG chain with two devices on it (Cortex-A8 4 bits IRLen, custom device 5-bits IRLen).

```
void InitTarget(void) {
 Report("J-Link script example.");
 MessageBox("Can not find xxx device");
   return 1;
 JTAG_DRPre
                       = 0; // Cortex-A8 is closest to TDO, no no pre devices
 JTAG_DRPost
                       = 1; // 1 device (custom device) comes after the Cortex-A8
                       = 0; // Cortex-A8 is closest to TDO, no no pre IR bits
 JTAG_IRPre
                       = 5; // custom device after Cortex-A8 has 5 bits IR len
 JTAG IRPost
 JTAG_IRLen
                       = 4; // We selected the Cortex-A8, it has 4 bits IRLen
                       = CORTEX_A8; // We are connected to a Cortex-A8
= 1; // We are allowed to enter JTAG TAP reset
 CPU
 JTAG_AllowTAPReset
  11
 // We have a non-CoreSight compliant Cortex-A8 here
  // which does not allow auto-detection of the Core debug components base address.
 // so set it manually to overwrite the DLL auto-detection
  11
 CORESIGHT_CoreBaseAddr = 0x80030000;
3
```

5.10.7 Executing J-Link script files

5.10.7.1 In J-Link commander

When J-Link commander is started it searches for a script file called Default.JLinkScript. If this file is found, it is executed instead of the standard auto detection of J-Link. If this file is not present, J-Link commander behaves as before and the normal auto-detection is performed.

5.10.7.2 In debugger IDE environment

To execute a script file out of your debugger IDE, simply select the script file to execute in the Settings tab of the J-Link control panel and click the save button (after the debug session has been started). Usually a project file for J-Link is set by the debugger, which allows the J-Link DLL to save the settings of the control panel in this project file. After selecting the script file restart your debug session. From now on, the script file will be executed when starting the debug session.

5.10.7.3 In GDB Server

In order to execute a script file when using J-Link GDB Server, simply start the GDB Server, using the following command line paramter:

-scriptfile <file>

For more information about the <code>-scriptfile</code> command line parameter, please refer to UM08005, chapter "command line options".

5.11 Command strings

The behavior of the J-Link can be customized via command strings passed to the JLinkARM.dll which controls J-Link. Applications such as the J-Link Commander, but also the C-SPY debugger which is part of the IAR Embedded Workbench, allow passing one or more command strings. Command line strings can be used for passing commands to J-Link (such as switching on target power supply), as well as customize the behavior (by defining memory regions and other things) of J-Link. The use of command strings enables options which can not be set with the configuration dialog box provided by C-SPY.

5.11.1 List of available commands

The table below lists and describes the available command strings.

Command	Description
device	Selects the target device.
DisableFlashBPs	Disables the FlashPB feature.
DisableFlashDL	Disables the J-Link ARM FlashDL feature.
EnableFlashBPs	Enables the FlashPB feature.
EnableFlashDL	Enables the J-Link ARM FlashDL feature.
map exclude	Ignore all memory accesses to specified area.
map indirectread	Specifies an area which should be read indirect.
map ram	Specifies location of target RAM.
map reset	Restores the default mapping, which means all mem-
	ory accesses are permitted.
SetAllowSimulation	Enable/Disable instruction set simulation.
SetCheckModeAfterRead	Enable/Disable CPSR check after read operations.
SetResetPulseLen	Defines the length of the RESET pulse in milliseconds.
SetResetType	Selects the reset strategy
SetRestartOnClose	Specifies restart behavior on close.
SetDbgPowerDownOnClose	Used to power-down the debug unit of the target CPU when the debug session is closed.
SetSysPowerDownOnIdle	Used to power-down the target CPU, when there are no transmissions between J-Link and target CPU, for a specified timeframe.
SupplyPower	Activates/Deactivates power supply over pin 19 of the JTAG connector.
SupplyPowerDefault	Activates/Deactivates power supply over pin 19 of the JTAG connector permanently.

5.11.1.1 device

This command selects the target device.

Syntax

device = <DeviceID>

DeviceID has to be a valid device identifier. For a list of all available device identifiers please refer to chapter *Supported devices* on page 148.

Example

device = AT91SAM7S256

5.11.1.2 DisableFlashBPs

This command disables the FlashBP feature.

Syntax

DisableFlashBPs

5.11.1.3 DisableFlashDL

This command disables the J-Link ARM FlashDL feature.

Syntax

DisableFlashDL

5.11.1.4 EnableFlashBPs

This command enables the FlashBP feature.

Syntax

EnableFlashBPs

5.11.1.5 EnableFlashDL

This command enables the J-Link ARM FlashDL feature.

Syntax

EnableFlashDL

5.11.1.6 map exclude

This command excludes a specified memory region from all memory accesses. All subsequent memory accesses to this memory region are ignored.

Memory mapping

Some devices do not allow access of the entire 4GB memory area. Ideally, the entire memory can be accessed; if a memory access fails, the CPU reports this by switching to abort mode. The CPU memory interface allows halting the CPU via a WAIT signal. On some devices, the WAIT signal stays active when accessing certain unused memory areas. This halts the CPU indefinitely (until RESET) and will therefore end the debug session. This is exactly what happens when accessing critical memory areas. Critical memory areas should not be present in a device; they are typically a hardware design problem. Nevertheless, critical memory areas exist on some devices.

To avoid stalling the debug session, a critical memory area can be excluded from access: J-Link will not try to read or write to critical memory areas and instead ignore the access silently. Some debuggers (such as IAR C-SPY) can try to access memory in such areas by dereferencing non-initialized pointers even if the debugged program (the debuggee) is working perfectly. In situations like this, defining critical memory areas is a good solution.

Syntax

map exclude <SAddr>-<EAddr>

Example

This is an example for the ${\tt map}$ ${\tt exclude}$ command in combination with an NXP LPC2148 MCU.

Memory map

0x00000000-0x0007FFFF	On-chip flash memory
0x00080000-0x3FFFFFFF	Reserved
0x40000000-0x40007FFF	On-chip SRAM
0x40008000-0x7FCFFFFF	Reserved
0x7FD00000-0x7FD01FFF	On-chip USB DMA RAM
0x7FD02000-0x7FD02000	Reserved
0x7FFFD000-0x7FFFFFFF	Boot block (remapped from on-chip flash memory)
0x80000000-0xDFFFFFFF	Reserved
0xE0000000-0xEFFFFFFF	VPB peripherals
0xF0000000-0xFFFFFFFF	AHB peripherals

The "problematic" memory areas are:

0x00080000-0x3FFFFFFF	Reserved
0x40008000-0x7FCFFFFF	Reserved
0x7FD02000-0x7FD02000	Reserved
0x80000000-0xDFFFFFFF	Reserved

To exclude these areas from being accessed through J-Link the $\tt map \ exclude \ command \ should \ be used as follows:$

map exclude 0x00080000-0x3FFFFFFF
map exclude 0x40008000-0x7FCFFFFF
map exclude 0x7FD02000-0x7FD02000
map exclude 0x8000000-0xDFFFFFFF

5.11.1.7 map indirectread

This command can be used to read a memory area indirectly. Indirectly reading means that a small code snippet is downloaded into RAM of the target device, which reads and transfers the data of the specified memory area to the host. Before map indirectread can be called a RAM area for the indirectly read code snippet has to be defined. Use therefor the map ram command and define a RAM area with a size of >= 256 byte.

Typical applications

Refer to chapter Fast GPIO bug on page 212 for an example.

Syntax

map indirectread <StartAddressOfArea>-<EndAddress>

Example

map indirectread 0x3fffc000-0x3fffcfff

5.11.1.8 map ram

This command should be used to define an area in RAM of the target device. The area must be 256-byte aligned. The data which was located in the defined area will not be corrupted. Data which resides in the defined RAM area is saved and will be restored if necessary. This command has to be executed before map indirectread will be called.

Typical applications

Refer to chapter Fast GPIO bug on page 212 for an example.

Syntax

map ram <StartAddressOfArea>-<EndAddressOfArea>

Example

map ram 0x4000000-0x40003fff;

5.11.1.9 map reset

This command restores the default memory mapping, which means all memory accesses are permitted.

Typical applications

Used with other "map" commands to return to the default values. The map reset command should be called before any other "map" command is called.

Syntax

map reset

Example

map reset

5.11.1.10 SetAllowSimulation

This command can be used to enable or disable the instruction set simulation. By default the instruction set simulation is enabled.

Syntax

```
SetAllowSimulation = 0 | 1
```

Example

SetAllowSimulation 1 // Enables instruction set simulation

5.11.1.11 SetCheckModeAfterRead

This command is used to enable or disable the verification of the CPSR (current processor status register) after each read operation. By default this check is enabled. However this can cause problems with some CPUs (e.g. if invalid CPSR values are returned). Please note that if this check is turned off (SetCheckModeAfterRead = 0), the success of read operations cannot be verified anymore and possible data aborts are not recognized.

Typical applications

This verification of the CPSR can cause problems with some CPUs (e.g. if invalid CPSR values are returned). Note that if this check is turned off (SetCheckModeAfterRead = 0), the success of read operations cannot be verified anymore and possible data aborts are not recognized.

Syntax

```
SetCheckModeAfterRead = 0 | 1
```

Example

```
SetCheckModeAfterRead = 0
```

5.11.1.12 SetResetPulseLen

This command defines the length of the RESET pulse in milliseconds. The default for the RESET pulse length is 20 milliseconds.

Syntax

```
SetResetPulseLen = <value>
```

Example

```
SetResetPulseLen = 50
```

5.11.1.13 SetResetType

This command selects the reset startegy which shall be used by J-Link, to reset the device. The value which is used for this command is analog to the reset type which shall be selected. For a list of all reset types which are available, please refer to *Reset strategies* on page 119. Please note that there different reset strategies for ARM 7/9 and Cortex-M devices.

Syntax

SetResetType = <value>

Example

SetResetType = 0 // Selects reset strategy type 0: normal

5.11.1.14 SetRestartOnClose

This command specifies whether the J-Link restarts target execution on close. The default is to restart target execution. This can be disabled by using this command.

Syntax

```
SetRestartOnClose = 0 | 1
```

Example

SetRestartOnClose = 1

5.11.1.15 SetDbgPowerDownOnClose

When using this command, the debug unit of the target CPU is powered-down when the debug session is closed.

Note: This command works only for Cortex-M3 devices

Typical applications

This feature is useful to reduce the power consumption of the CPU when no debug session is active.

Syntax

SetDbgPowerDownOnClose = <value>

Example

SetDbgPowerDownOnClose = 1 // Enables debug power-down on close. SetDbgPowerDownOnClose = 0 // Disables debug power-down on close.

5.11.1.16 SetSysPowerDownOnIdle

When using this command, the target CPU is powered-down when no transmission between J-Link and the target CPU was performed for a specific time. When the next command is given, the CPU is powered-up.

Note: This command works only for Cortex-M3 devices.

Typical applications

This feature is useful to reduce the power consumption of the CPU.

Syntax

```
SetSysPowerDownOnIdle = <value>
```

Note: A 0 for <value> disables the power-down on idle functionality.

Example

5.11.1.17 SupplyPower

This command activates power supply over pin 19 of the JTAG connector. The KS (Kickstart) versions of J-Link have the V5 supply over pin 19 activated by default.

Typical applications

This feature is useful for some eval boards that can be powered over the JTAG connector.

Syntax

SupplyPower = $0 \mid 1$

Example

```
SupplyPower = 1
```

5.11.1.18 SupplyPowerDefault

This command activates power supply over pin 19 of the JTAG connector permanently. The KS (Kickstart) versions of J-Link have the V5 supply over pin 19 activated by default.

Typical applications

This feature is useful for some eval boards that can be powered over the JTAG connector.

Syntax

SupplyPowerDefault = 0 | 1

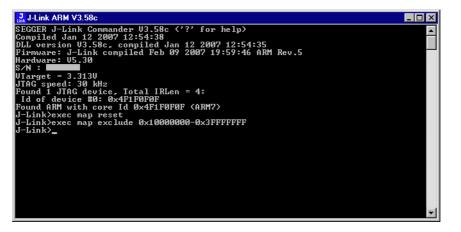
Example

SupplyPowerDefault = 1

5.11.2 Using command strings

5.11.2.1 J-Link Commander

The J-Link command strings can be tested with the J-Link Commander. Use the command $_{\rm exec}$ supplemented by one of the command strings.



Example

```
exec SupplyPower = 1
exec map reset
exec map exclude 0x1000000-0x3FFFFFFF
```

5.11.2.2 IAR Embedded Workbench

The J-Link command strings can be supplied using the C-SPY debugger of the IAR Embedded Workbench. Open the **Project options** dialog box and select **Debugger**.

Options for node "Proj	ect"	×
Options for node "Proj Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link/J-Trace LMI FTDI Macraigor RDI Third-Party Driver		Factory Settings
	OK	Cancel

On the Extra Options page, select Use command line options.

Enter --jlink_exec_command "<CommandLineOption>" in the textfield, as shown in the screenshot below. If more than one command should be used separate the commands with semicolon.

Options for node "Proje	ect"
Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link/J-Trace LMI FTDI Macraigor RDI Third-Party Driver	Factory Settings Setup Download Extra Options Plugins Use command line options; Command line options; Command line options; (one per line) jlink_exec_command 'map ram 0x40000000-0x40003fff; map indire Image: set the set of the
	OK Cancel

5.12 Switching off CPU clock during debug

We recommend not to switch off CPU clock during debug. However, if you do, you should consider the following:

Non-synthesizable cores (ARM7TDMI, ARM9TDMI, ARM920, etc.)

With these cores, the TAP controller uses the clock signal provided by the emulator, which means the TAP controller and ICE-Breaker continue to be accessible even if the CPU has no clock.

Therefore, switching off CPU clock during debug is normally possible if the CPU clock is periodically (typically using a regular timer interrupt) switched on every few ms for at least a few us. In this case, the CPU will stop at the first instruction in the ISR (typically at address 0x18).

Synthesizable cores (ARM7TDMI-S, ARM9E-S, etc.)

With these cores, the clock input of the TAP controller is connected to the output of a three-stage synchronizer, which is fed by clock signal provided by the emulator, which means that the TAP controller and ICE-Breaker are not accessible if the CPU has no clock.

If the RTCK signal is provided, adaptive clocking function can be used to synchronize the JTAG clock (provided by the emulator) to the processor clock. This way, the JTAG clock is stopped if the CPU clock is switched off.

If adaptive clocking is used, switching off CPU clock during debug is normally possible if the CPU clock is periodically (typically using a regular timer interrupt) switched on every few ms for at least a few us. In this case, the CPU will stop at the first instruction in the ISR (typically at address 0x18).

5.13 Cache handling

Most ARM systems with external memory have at least one cache. Typically, ARM7 systems with external memory come with a unified cache, which is used for both code and data. Most ARM9 systems with external memory come with separate caches for the instruction bus (I-Cache) and data bus (D-Cache) due to the hardware architecture.

5.13.1 Cache coherency

When debugging or otherwise working with a system with processor with cache, it is important to maintain the cache(s) and main memory coherent. This is easy in systems with a unified cache and becomes increasingly difficult in systems with hardware architecture. A write buffer and a D-Cache configured in write-back mode can further complicate the problem.

ARM9 chips have no hardware to keep the caches coherent, so that this is the responsibility of the software.

5.13.2 Cache clean area

J-Link / J-Trace handles cache cleaning directly through JTAG commands. Unlike other emulators, it does not have to download code to the target system. This makes setting up J-Link / J-Trace easier. Therefore, a cache clean area is not required.

5.13.3 Cache handling of ARM7 cores

Because ARM7 cores have a unified cache, there is no need to handle the caches during debug.

5.13.4 Cache handling of ARM9 cores

ARM9 cores with cache require J-Link / J-Trace to handle the caches during debug. If the processor enters debug state with caches enabled, J-Link / J-Trace does the following:

When entering debug state

J-Link / J-Trace performs the following:

- it stores the current write behavior for the D-Cache
- it selects write-through behavior for the D-Cache.

When leaving debug state

J-Link / J-Trace performs the following:

- it restores the stored write behavior for the D-Cache
- it invalidates the D-Cache.

Note: The implementation of the cache handling is different for different cores. However, the cache is handled correctly for all supported ARM9 cores.

Chapter 6 Flash download

This chapter describes how the flash download feature of the DLL can be used in different debugger environments.

6.1 Introduction

The J-Link DLL comes with a lot of flash loaders that allow direct programming of internal flash memory for popular microcontrollers. Moreover, the J-Link DLL also allows programming of CFI-compliant external NOR flash memory. The flash download feature of the J-Link DLL does not require an extra license and can be used free of charge.

Why should I use the J-Link flash download feature?

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 download feature of J-Link is very efficient and allows fast flash programming. For example, if a debugger splits the download image into several pieces, the flash download software will collect the individual parts and perform the actual flash programming right before program execution. This avoids repeated flash programming. Once the setup of flash download is completed. Moreover, the J-Link flash loaders make flash behave as RAM. This means that the debugger only needs to select the correct device which enables the J-Link DLL to automatically activate the correct flash loader if the debugger writes to a specific memory address.

This also makes it very easy for debugger vendors to make use of the flash download feature because almost no extra work is necessary on the debugger side since the debugger has not to differ between memory writes to RAM and memory writes to flash.

6.2 Licensing

No extra license required. The flash download feature can be used free of charge.

6.3 Supported devices

J-Link supports download into the internal flash of a large number of microcontrollers. You can always find the latest list of supported devices on our website:

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

In general, J-Link can be used with any ARM7/9/11, Cortex-M0/M1/M3/M4 and Cortex-A5/A8/R4 core even if it does not provide internal flash.

Furthermore, flash download is also available for all CFI-compliant external NOR-flash devices.

6.4 Setup for various debuggers (internal flash)

The J-Link flash download feature can be used by different debuggers, such as IAR Embedded Workbench, Keil MDK, GDB based IDEs, ... For different debuggers there are different steps required to enable J-Link flash download. In this section, the setup for different debuggers is explained.

6.4.1 IAR Embedded Workbench

Using the J-Link flash download feature in IAR EWARM is quite simple:

First, choose the right device in the project settings if not already done. The device settings can be found at **Project->Options->General Options->Target**.

Options for node "at91s	sam7s-ek"	×
Category: General Options C/C++ Compiler Assembler Output Converter Custom Build Build Actions Linker Debugger Simulator Angel GDB Server IAR ROM-monitor J-Link/J-Trace LMI FTDI Macraigor RDI Third-Party Driver	Target Output Library Configuration Library Options MISRA-C Processor variant C Ogre ARM7TDMI Device Atmel at91sam7s256 Little Endian mode EPU Little Big BE32 BE32 	
	Cance	

To use the J-Link flash loaders, the IAR flash loader has to be disabled. To disable the IAR flash loader, the checkbox **Use flash loader(s)** at **Project->Options->Debug-ger->Download** has to be disabled, as shown below.

General Options C/C++ Compiler		Factory Settings
Assembler Output Converter Custom Build Build Actions Linker Debugger Simulator Angel GDB Server IAR ROM-monitor J-Link/J-Trace LMI FTDI Macraigor RDI Third-Party Driver	Setup Download Extra Options Plugins Attach to program Verify download Suppress download Use flash loader(s) 0x100000.(default).	Edit

6.4.2 Keil MDK

To use the J-Link flash download feature in Keil MDK, the following steps need to be performed:

First, choose the device in the project settings if not already done. The device settings can be found at **Project->Options for Target->Device**.

🖁 Options for Target 'MCBSTM32E Flash' 🛛 🔍 🕺
Device Target Output Listing User C/C++ Asm Linker Debug Utilities Database: Generic CPU Data Base
Device: STM32F103ZE Toolset: ARM
ARM 32-bit Cortex-M3 Microcontroller, 72MHz, 512kB Riash, 64kB SRAM, STM32F103VG STM32F1032C STM32F1032D STM32F1032D STM32F1032D STM32F1032E STM32F1032E STM32F1032E STM32F1032F STM32F1032F STM32F1032F STM32F105R8
OK Cancel Defaults Help

To enable the J-Link flash loader **J-Link / J-Trace** at **Project->Options for Target->Utilities** has to be selected. It is important that "Update Target before Debugging" is unchecked since otherwise uVision tries to use its own flashloader.

🕅 Options for Target 'MCBSTM32E Flash'	×
Device Target Output Listing User C/C++ Asm Linker Debug Utilities	
Configure Flash Menu Command	
O Use Target Driver for Flash Programming	
Cortex-M/R J-LINK/J-Trace Settings Update Target before Debugging	
Init File: Edit	
C Use External Tool for Rash Programming	
Command:	
Arguments:	
E Run Independent	
OK Cancel Defaults Help	

Then J-Link has to be selected as debugger. To select J-Link as debugger simply choose J-Link / J-Trace from the list box which can be found at **Project-**>**Options for Target->Debug**.



Now setup the Download Options at Project->Options for Target->Debug -> Settings. Check Verify Code Download and Download to Flash as shown in the screenshot below.

Cortex JLink/JTrace Target Driver Setup	<u>×</u>
Debug Trace Flash Download	
J-Link / J-Trace Adapter JTAG Device Chain SN: 173000305 ▼ USB#;0 ▼ Device: J-Link ARM-Pro HW: V3.00 dll: V4.35c TD0 FW: J-Link ARM-Pro V3.x compilec Port: Max Clock: JTAG ▼ 2MHz<	eSight JTAG-DP 4 Up I JTAG device 5 Down
Debug Connect & Reset Options Reset: Nomal ✓ Cache Options ✓ Cache Options	Download Options ✓ Verify Code Download ✓ Download to Eashi
Interface TCP/IP © USB © TCP/IP Network Settings Scan IP-Address State: ready 127 . 0 . 0 . 1	Autodetect JLink Info Ping JLink Cmd
OK Cancel	Help

6.4.3 J-Link GDB Server

The configuration for the J-Link GDB Server is done by the <code>.gdbinit</code> file. The following command has to be added to the <code>.gdbinit</code> file to enable the J-Link flash download feature:

monitor flash device <DeviceName>

<DeviceName> is the name of the device for which download into internal flash memory shall be enabled. For a list of supported devices, please refer to Supported devices on page 148. For more information about the GDB monitor commands please refer to UM08005, J-Link GDB Server User Guide chapter Supported remote commands.

6.4.4 J-Link Commander

To configure J-Link Commander for flash download simply select the connected device by typing in the following command:

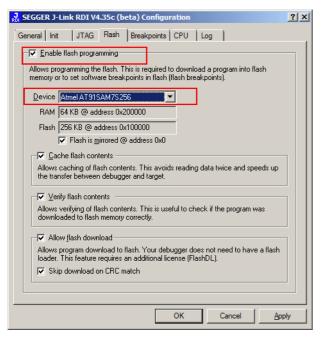
exec device = <DeviceName>

<DeviceName> is the name of the device for which download into internal flash memory shall be enabled. For a list of supported devices, please refer to *Supported devices* on page 148.

🔜 J-Link Commander	
JTAG speed: 100 kHz	
J-Link>speed 4000	
JTAG speed: 4000 kHz	
J-Link)h	
PC: (R15) = 0010079A, CPSR = 2000007F (System mode, THUMB FIQ dis.)	
R0 = 00000001, $R1 = 00202060$, $R2 = 00000001$, $R3 = 0010198F$	
R4 = 000001F4, R5 = 00000000, R6 = 00025992, R7 = 00202CE0 USR: R8 =00000000, R9 =00000000, R10=00000000, R11 =00000000, R12 =0000005F	
R13=00201FD8. R14=00102495	
FIQ: R8 =00000000, R9 =00000000, R10=00000000, R11 =00000000, R12 =00000000	
R13=002020400, R14=00000000, SPSR=F0000036	
SUC: R13=000000000, R14=001007A0, SPSR=2000007F	
ABT: R13=00000000, R14=00000000, SPSR=F00000F9	
IRQ: R13=00202840, R14=001006DD, SPSR=8000007F	
UND: R13=00000000. R14=00000000. SPSR=E0000092	
J-Link>exec device = AT91SAM7S256	
Info: Device "AT91SAM7S256" selected (256 KB flash, 64 KB RAM).	
J-Link>loadbin C:\Temp\test.bin,0x100000	
Loading binary file LUNlemp/test.binl	
Writing bin data into target memory @ 0x00100000.	
Info: J-Link: Flash download: Flash programming performed for 1 range <16384	лус
Info: J-Link: Flash download: Total time needed: 0.844s (Prepare: 0.116s, Co e: 0.020s, Program: 0.654s, Verify: 0.015s, Restore: 0.037s)	mhan.
e. 0.0208, frogram. 0.0348, verify. 0.0138, nestore. 0.03787 J-Link>	-

6.4.5 J-Link RDI

The configuration for J-Link RDI is done via the J-Link RDI configuration dialog.



For more information about the J-Link RDI configuration dialog please refer to UM08004, J-Link RDI User Guide, chapter Configuration dialog.

6.5 Setup for various debuggers (CFI flash)

The setup for download into CFI-compliant memory is different from the one for internal flash. In this section, the setup for different debuggers is explained.

6.5.1 IAR Embedded Workbench / Keil MDK

Using the J-Link flash download feature with IAR Embedded Workbench / Keil MDK is quite simple:

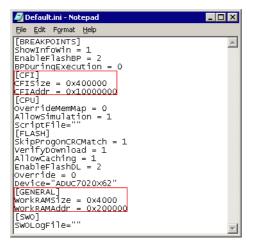
First, start the debug session and open the J-Link Control Panel. In the tab "Settings" you will find the location of the settings file.

🛃 SEGGER J-Link ¥4.15r (beta) - Control panel	_ 🗆 🗙
General Settings Breakpoints Log NET C	PU Regs Target Power SWV Device Emu
C: VLink.log	verride … Clear
Settings file C:\Program Files\SEGGER\JLinkARM_V415r\Defau	It.ini Override
Script file	
Flash download On Verify download on CRC match Verify download	■Flash breakpoints C Auto License found C On V Show info window during C Off program
Enabled, download pending: 0 bytes	Enabled
Override device selection Allow caching of flash contents (On) Allow instruction set simulation Override memory map	Modify breakpoints during execution
Ready JLINK_GetSpeed (Done)	0.243 sec. in 36 calls //.

Close the debug session and open the settings file with a text editor. Add the following lines to the file:

[CFI] CFISize = <FlashSize> CFIAddr = <FlashAddr> [GENERAL] WorkRAMSize = <RAMSize> WorkRAMAddr = <RAMAddr>

After this the file should look similar to the sample in the following screenshot.



Save the settings file and restart the debug session. Open the J-Link Control Panel and verify that the "MemMap" tab shows the new settings for CFI flash and work RAM area.

SEGGER J-Link V4.			1 I	1			
Log NET CPU CFI Flash Config 0x10000000 Info n.a.	Regs Target F	ower SWV		Work R/	\M	Performance	
Range	Siz		Гуре І	Explanation			
0x0000000 - 0xFFF	FFFFF 4 G	B	N 1	Normal			
Ready JLIN	K_ETM_IsPresen	: (Done)			0.603 sec. i	n 37 calls	

6.5.2 J-Link GDB Server

The configuration for the J-Link GDB Server is done by the .gdbinit file. The following commands have to be added to the .gdbinit file to enable the flash download feature:

```
monitor WorkRAM = <SAddr>-<EAddr>
monitor flash CFI = <SAddr>-<EAddr>
```

For more information about the GDB monitor commands please refer to UM08005, J-Link GDB Server User Guide chapter Supported remote commands.

6.5.3 J-Link commander

r

The following command sequence shows how to perform a download into external, CFI-compliant, parallel NOR-Flash on a ST STM32F103ZE using J-Link commander:

```
speed 1000
exec setcfiflash 0x64000000 - 0x64FFFFF
exec setworkram 0x2000000 - 0x2000FFFF
w4 0x40021014, 0x00000114 // RCC_AHBENR, FSMC clock enable
w4 0x40021018, 0x000001FD // GPIOD~G clock enable
w4 0x40011400, 0xB4BE44BB // GPIOD low config, NOE, NWE => Output, NWAIT => Input
w4 0x40011404, 0xBBBBBBBB // GPIOD high config, A16-A18
w4 0x40011800, 0xBBBBBBBB // GPIOE low config, A19-A23
w4 0x40011804, 0xBBBBBBBB // GPIOE high config, D5-D12
w4 0x40011C00, 0x44BBBBBB // GPIOF low config, A0-A5
w4 0x40011C04, 0xBBBB4444 // GPIOF high config, A6-A9
w4 0x40012004, 0x44BBBBB // GPIOG low config, A10-A15
w4 0x40012004, 0x44BBBBB // GPIOG high config, NE2 => output
w4 0xA000008, 0x00001059 // CS control reg 2, 16-bit, write enable, Type: NOR flash
w4 0xA000000C, 0x10000505 // CS2 timing reg (read access)
w4 0xA00010C, 0x1000505 // CS2 timing reg (write access)
speed 4000
mem 0x64000000,100
loadbin C:\STMB672_STM32F103ZE_TestBlinky.bin,0x64000000
mem 0x64000000,100
```

6.6 Using the DLL flash loaders in custom applications

The J-Link DLL flash loaders make flash behave as RAM from a user perspective, since flash programming is triggered by simply calling the J-Link API functions for memory reading / writing. For more information about how to setup the J-Link API for flash programming please refer to *UM08002 J-Link SDK* documentation (available for SDK customers only).

CHAPTER 6

Chapter 7 Flash breakpoints

This chapter describes how the flash breakpoints feature of the DLL can be used in different debugger environments.

7.1 Introduction

The J-Link DLL supports a feature called flash breakpoints which allows the user to set an unlimited number of breakpoints in flash memory rather than only being able to use the hardware breakpoints of the device. Usually when using hardware breakpoints only, a maximum of 2 (ARM 7/9/11) to 8 (Cortex-A/R) breakpoints can be set. The flash memory can be the internal flash memory of a supported microcontroller or external CFI-compliant flash memory. In the following sections the setup for different debuggers to use the flash breakpoints feature is explained.

How do breakpoints work?

There are basically 2 types of breakpoints in a computer system: hardware breakpoints and software breakpoints. Hardware breakpoints require a dedicate hardware unit for every breakpoint. In other words, the hardware dictates how many hardware breakpoints can be set simultaneously. ARM 7/9 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, this usually requires the program to be located in RAM.

What is special about software breakpoints in flash?

Flash breakpoints allows setting of an unlimited number of breakpoints even if the user application is not located in RAM. On modern microcontrollers this is the standard scenario because on most microcontrollers the internal RAM is not big enough to hold the complete application. When replacing instructions in flash memory this requires re-programming of the flash which takes much more time than simply replacing a instruction when debugging in RAM. The J-Link flash breakpoints feature is highly optimized for fast flash programming speed and in combination with the instruction set simulation only re-programs flash is absolutely necessary which makes debugging in flash using flash breakpoints almost as flawless as debugging in RAM.

What performance can I expect?

Flash algorithm, specially designed for this purpose, sets and clears flash breakpoints extremely fast; on microcontrollers with fast flash the difference between software breakpoints in RAM and flash is hardly noticeable.

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 soft ware and hardware breakpoints allows us to use hardware breakpoints a lot of times, especially when the debugger is source level-stepping, avoiding re-programming the 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, while maximizing the life time of the flash. All resources of the ARM microcontroller are available to the application program, no memory is lost for debugging.

7.2 Licensing

In order to use the flash breakpoints feature a separate license is necessary for each J-Link. For some devices J-Link comes with a device-based license and some J-Link models also come with a full license for flash breakpoints but the normal J-Link comes without any licenses. For more information about licensing itself and which devices have a device-based license, please refer to *Licensing* on page 47.

7.2.1 24h flash breakpoint trial license

In general, SEGGER offers free 30-days trial licenses for flash breakpoints upon request. The J-Link DLL also comes with a special feature that allows the user to test the flash breakpoints feature for 24 hours without the need to request a trial license explicitly from SEGGER via E-Mail. This especially is useful for users who simply want to do some short term testing with the flash breakpoints feature without needing to wait for a requested trial license key. This special trial license can only activated once per emulator. If the user sets breakpoints during the debug session which would require a flash breakpoint license and no license is found, the DLL offers the user to activate the 24 hour trial license for the connected emulator.

J-Link ¥x.	xxx Out of breakpoints 🛛 🔀
⚠	The debugger is trying to set a breakpoint point at address 0x00001F9C, but neither hardware nor software breakpoints are available.
	J-Link supports an unlimited number of breakpoints in flash memory. This feature requires an additional license from SEGGER, www.segger.com. It allows setting this and other breakpoints in flash memory and provides a significantly improved debugging experience.
	Free trial licenses are available. For a license, please contact info@segger.com. For more information, visit http://www.segger.com/cms/link-flash-breakpoints.html
	You can activate a free one-time 24h trial license for this emulator. This will enable an unlimited number of breakpoints in flash memory for the next 24 hours.
	Do you want to activate the free trial license now ?
	Do not show this message again for today
	<u>Yes</u> <u>N</u> o

7.3 Supported devices

J-Link supports flash breakpoints for a large number of microcontrollers. You can always find the latest list of supported devices on our website:

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

In general, J-Link can be used with any ARM7/9/11, Cortex-M0/M1/M3/M4 and Cortex-A5/A8/R4 core even if it does not provide internal flash.

Furthermore, flash breakpoints are also available for all CFI compliant external NOR-flash devices.

7.4 Setup & compatibility with various debuggers

7.4.1 Setup

In compatible debuggers, flash breakpoints work if the J-Link flash loader works and a license for flash breakpoints is present. No additional setup is required. The flash breakpoint feature is available for internal flashes and for external CFI-flash. For more information about how to setup various debuggers for flash download, please refer to *Setup for various debuggers (internal flash)* on page 149. If flash breakpoints are available can be verified using the J-Link control panel:

🌄 SEGGER J-Link ¥4.35g (beta) - Control panel	
General Settings Breakpoints Log NET CPU Reg	
C:VLink.log	Clear
Settings file C:\Program Files\SEGGER\JLinkARM_V435g\Default.ini	Cverride
Script file Not specified	
Flash download	Flash breakpoints Auto License found
Compare Using fastest method Off Verify Programmed sectors, fastest method	Auto License round On Show info window during Off program
Enabled, download pending: 0 bytes	Enabled
Allow caching of flash contents (On) Allow instruction set simulation Override memory map	Modify breakpoints during execution
Ready JLINK_HasError (Done)	1.110 sec. in 14 calls

7.4.2 Compatibility with various debuggers

Flash breakpoints can be used in all debugger which use the proper J-Link API to set breakpoints. Compatible debuggers/ debug interfaces are:

- IAR Embedded Workbench
- Keil MDK
- GDB-based debuggers
- Codewarrior
- RDI-compliant debuggers

Incompatible debuggers / debug interfaces

Rowley Crossworks

7.5 FAQ

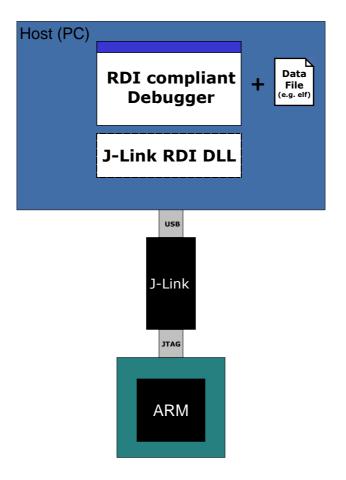
- Q: Why can flash breakpoints not be used with Rowley Crossworks?
- A: Because Rowley Crossworks does not use the proper J-Link API to set breakpoints. Instead of using the breakpoint-API, Crossworks programs the debug hardware directly, leaving J-Link no choice to use its flash breakpoints.

Chapter 8 RDI

RDI (Remote Debug Interface) is a standard defined by ARM, trying to standardize a debugger / debug probe interface. It is defined for cores only that have the same CPU register set as ARM7 CPUs. This chapter describes how to use the RDI DLL which comes with the J-Link software and documentation package. The J-Link RDI DLL allows the user to use J-Link with any RDI-compliant debugger and IDE.

8.1 Introduction

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. The J-Link DLL feature flash download and flash breakpoints can also be used with J-Link RDI.



8.1.1 Features

- Can be used with every RDI compliant debugger
- Easy to use
- Flash download feature of J-Link DLL can be used
- Flash breakpoints feature of J-Link DLL can be used.
- Instruction set simulation (improves debugging performance)

8.2 Licensing

In order to use the J-Link RDI software a separate license is necessary for each J-Link. For some devices J-Link comes with a device-based license and some J-Link models also come with a full license for J-Link RDI but the normal J-Link comes without any licenses. For more information about licensing itself and which devices have a device-based license, please refer to *Licensing* on page 47.

8.3 Setup for various debuggers

The J-Link RDI software is an ARM Remote Debug Interface (RDI) for J-Link. It makes it possible to use J-Link with any RDI compliant debugger. Basically, J-Link RDI consists of a additional DLL (JLinkRDI.dll) which builds the interface between the RDI API and the normal J-Link DLL. The JLinkRDI.dll itself is part of the J-Link software and documentation package.

8.3.1 IAR Embedded Workbench IDE

J-Link RDI can be used with IAR Embedded Workbench for ARM.

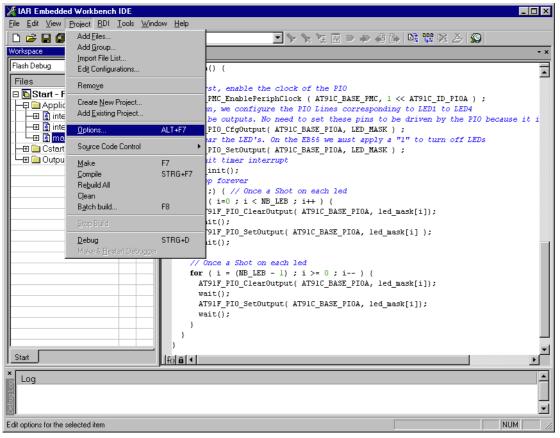
8.3.1.1 Supported software versions

J-Link RDI 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.

Note: Since IAR EWARM V5.30 J-Link is fully and natively supported by EWARM, so RDI is no longer needed.

8.3.1.2 Configuring to use J-Link RDI

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



2. Choose Project | Options and select the Debugger category. Change the

Category: General Options Factory Settings General Options C/C++ Compiler Assembler Download Extra Options Plugins Custom Build Driver Eun to Build Actions Linker Setup macros main Simulator Angel Lase macro file Juinker J-Link Use macro file Use macro file Use macro file Devige description file Diveride defauli StoOLKIT DIRS/CONFIG/Voot91sam7s64.ddf Image: Constant Constan	Driver	option to RDI.
	General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI	Setup Download Extra Options Plugins Driver Plugins Extra Options Plugins Setup macros

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

	Factory Settings
RDI	
Manufacturer RDI driver C:\seggerWLinkRDI/ULinkRDI.dll	
C Allow hardware reset	Note Use the RDI menu to specify additional driver settings. (This menu is available after the RDI
ETM trace	driver has been located) Catch exceptions Beset Data EIQ
Log RDI <u>communication</u> \$TOOLKIT_DIR\$\cspycomm.log	Undef Erefetch
	Manufacturer RDI driver C:\segger\JLinkRDI\JLinkRDI.dll Allow hardware reset ETM trace Log RDI communication

4. Now an extra menu, RDI, has been added to the menu bar. Choose **RDI** | **Configure** to configure the J-Link. For more information about the generic setup of J-Link RDI, please refer to *Configuration* on page 182.

X 1	💥 IAR Embedded Workbench IDE								
Eile	<u>E</u> dit	<u>V</u> iew	Project	<u>R</u> DI	<u>T</u> ools	<u>W</u> indow	Hel	p	
D	2	86	I 😂	<u>_</u> 0	onfigure	·		•	

8.3.1.3 Debugging on Cortex-M3 devices

The RDI protocol has only been specified by ARM for ARM 7/9 cores. For Cortex-M there is no official extension of the RDI protocol regarding the register assignement, that has been approved by ARM. Since IAR EWARM version 5.11 it is possible to use J-Link RDI for Cortex-M devices because SEGGER and IAR have been come to an agreement regarding the RDI register assignment for Cortex-M. The following table lists the register assignment for RDI and Cortex-M:

1 R1 2 R2 3 R3 4 R4 5 R5 6 R6 7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK	Register Index	Assigned register
2 R2 3 R3 4 R4 5 R5 6 R6 7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK	0	RO
3 R3 4 R4 5 R5 6 R6 7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	1	R1
4 R4 5 R5 6 R6 7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK	2	R2
5 R5 6 R6 7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	3	R3
6 R6 7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK	4	R4
7 R7 8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	5	R5
8 R8 9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	6	R6
9 R9 10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	7	R7
10 R10 11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	8	R8
11 R11 12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	9	R9
12 R12 13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	10	R10
13 MSP / PSP (depending on mode) 14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	11	R11
14 R14 (LR) 16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	12	R12
16 R15 (PC) 17 XPSR 18 APSR 19 IPSR 20 EPSR 21 IAPSR 22 EAPSR 23 IEPSR 24 PRIMASK 25 FAULTMASK	13	MSP / PSP (depending on mode)
17XPSR18APSR19IPSR20EPSR21IAPSR22EAPSR23IEPSR24PRIMASK25FAULTMASK	14	R14 (LR)
18APSR19IPSR20EPSR21IAPSR22EAPSR23IEPSR24PRIMASK25FAULTMASK	16	R15 (PC)
19IPSR20EPSR21IAPSR22EAPSR23IEPSR24PRIMASK25FAULTMASK	17	XPSR
20EPSR21IAPSR22EAPSR23IEPSR24PRIMASK25FAULTMASK	18	APSR
21IAPSR22EAPSR23IEPSR24PRIMASK25FAULTMASK	19	IPSR
22EAPSR23IEPSR24PRIMASK25FAULTMASK	20	EPSR
23IEPSR24PRIMASK25FAULTMASK	21	IAPSR
24PRIMASK25FAULTMASK	22	EAPSR
25 FAULTMASK	23	IEPSR
	24	PRIMASK
26 BASEPRI	25	FAULTMASK
	26	BASEPRI
27 BASEPRI_MAX	27	BASEPRI_MAX
28 CFBP (CONTROL/FAULT/BASEPRI/PRIMASK) Table 8.1: Cortex-M register mapping for IAB + 1-Link RDI	28	

Table 8.1: Cortex-M register mapping for IAR + J-Link RDI

8.3.2 ARM AXD (ARM Developer Suite, ADS)

8.3.2.1 Software version

The JLinkRDI.dll has been tested with ARM'S AXD version 1.2.0 and 1.2.1. There should be no problems with other versions of ARM'S AXD. All screenshots are taken from ARM'S AXD version 1.2.0.

8.3.2.2 Configuring to use J-Link RDI

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

Choose Target					? ×
- Target Environ	ments —				
Target	RDI	File		Version	Add
ARM TPA	1.5.1	C:\Tool\C\\RVT.DLL		1.0.0.19	
ARMUL	1.5.1	C:\Tool\C\\armulate.dll		1.4.0.89	<u>R</u> emove
					Re <u>n</u> ame
					<u>S</u> ave As
					Carlound
					<u>C</u> onfigure
	rograms with	h the 'ARMulator' Instruction : nout physical ARM hardware,			
			OK	Cancel	Help

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

Open				? ×
Look in: 🔂	JLinkRDI	 •	(÷ 🖻 (* ⊞•
JLinkArm.c				
JLinkRDI.				
File <u>n</u> ame:				Onen
rile <u>n</u> ame.	JLinkRDI.dll			<u>O</u> pen
Files of type:	DLLs (*.dll)		•	Cancel

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

Cł	oose Target					? ×
Г	Target Environme	ents —				
	Target	RDI	File		Version	Add
	ARM TPA	1.5.1	C:\Tool\C\\RVT.DLL		1.0.0.19	
	ARMUL	1.5.1	C:\Tool\C\\armulate.dll		1.4.0.89	<u>R</u> emove
	J-Link	1.5.1	C:WLinkRDI/WLinkRDI.dll		2.40a	
						Re <u>n</u> ame
						<u>S</u> ave As
						C6
						<u>C</u> onfigure
	Segger JLink ARN	/ JTAG				
]	014		
				OK	Cancel	Help

4. Select J-Link and press **OK** to connect to the target via J-Link ARM. For more information about the generic setup of J-Link RDI, please refer to *Configuration* on page 182. After downloading an image to the target board, the debugger window looks as follows:

(3) AXD - [ARM_1 - C:\w	vork\embOS\embOS_ARM_	RVDS21\start\CPU_STR71X\SAMPLE\Main_LED.c]
孩 File Search Process	or Views System Views Exec	sute Options <u>W</u> indow <u>H</u> elp
n n o 🖌 🕯	1 i i i i i i i i i i i i i i i i i i i	
ARM_1 - Registers		21 22 void Taskl(void) {
Register	Value 🔺	23 while (1) {
B-Current	{}	24 LED_ToggleLED1();
-r0	0x20001580	25 OS_Delay (200);
-rl	0x20001588	
-r2	0x00000150	27 }
-r3	0x00001E80	29 /************************************
-r4	0x2000063C	30 *
r5	0x00001E70	31 * main
r6	0x00000000	32 *
r7	0x00000000	33 ************************************
-r8	0x00000000	34
r9	0x00000000	35 int main(void) { 36 OS IncDI(); /* Initially disable interrupts */
-r10	0x000027B0	37 OS InitKern(); /* initialize OS */
r11	0x0000000	38 OS InitHW(); /* initialize Hardware for OS */
-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 */ 44 return 0;
spsr	nzcvqift_User	45 3
⊞ User/System	{}	46
⊞-FIQ	{}	
Target Image Files	Class Breakp	Dints ARM_1 - Memory Start address 0x0
B ⇒ C:\temp\emb0S		Processor Position Tab1 - Hex - No prefix Tab2 - Hex - No prefix Tab
- ⇒ ₩ ARM_1		ARM_1 Stat_STR71x.axf: Main_LED.c Address 0 4 8 c
		Address 0 4 0 C
		0x0000000 E59FF018 E1400000 E59FF018 E59FF018 E59FF014
		0x00000020 000024A4 0000003C 00000040 00000044
		0x00000030 0000048 000022F0 0000004C EAFFFFE
		0x00000040 EAFFFFE EAFFFFE EAFFFFE EAFFFFFE
For Help, press F1		<pre><no pos=""> J-Link ARM_1 Start_STR71x.axf</no></pre>

8.3.3 ARM RVDS (RealView developer suite)

8.3.3.1 Software version

J-Link RDI has been tested with ARM RVDS version 2.1 and 3.0. There should be no problems with earlier versions of RVDS (up to version v3.0.1). All screenshots are taken from ARM's RVDS version 2.1.

Note: RVDS version 3.1 does not longer support RDI protocol to communicate with the debugger.

8.3.3.2 Configuring to use J-Link RDI

1. Start the Real View debugger:

<mark>₩</mark> RVDEBUG <start_str71x></start_str71x>			_ 🗆 🗵
File Edit Find View Project Tools Debug Help			
🗅 😅 🖬 🌡 🍽 砲 🔜 み み み み み か 🛤 수 🔸 🗶 🗈 🖭 🛱	と認識・	🗖 🖬 📰 🕮 🌮 👗	J State: Unknown
File: \$NO_SOURCE Find:			
Not connected - no PC or scope		🔼 Register	▼
Click to Connect to a Target		<no register<="" td=""><td>Context></td></no>	Context>
▲ ► \Dsm \Src \rtosinit_str71x.c+(vectors.s+/		F T Core	T F
☑ Name Value ☑ Name Value			
Call Stack (Locals (Statics) Watch1 (Watch2 (Watch3)	Memory 4		_
None>		4	Þ
For more information, select Help from Menu	Ln 1, Col 1		NUM

2. Select File | Connection | Connect to Target.

1000	VDEBUG <start_str7< th=""><th></th><th></th><th></th><th></th><th></th><th></th></start_str7<>						
	<u>File E</u> dit Fi <u>n</u> d <u>V</u> iew <u>F</u> New				D 60 92		
	<u>N</u> ew Open	Ctrl+O	ካውው ግ 🛋 🕇 ↓		¥ 結 融 →		a State: Unknown
File	Close	Ctrl+W	Line:	i) •		Register	▼ _ ⊽
No1 C1:-	Close Logs/Journals					<no register<="" th=""><th></th></no>	
	<u>S</u> ave Save As	Ctrl+S					
	Save/Close Multiple						
	<u>W</u> orkspace	•					
	Conn <u>e</u> ction Thread	+	Connect to <u>T</u> arget	Alt+0			
		01.01%.0	Disconnect (Defining Mode) Disconnect	Alt+Ctrl+0			
	Load Image Reload Image to Target	Ctrl+Shift+D Ctrl+F5	Connection Properties	Alt+Shift+0			
I	Refresh Symbols		Synchronization Control				
⊒[-	Set PC to Entry Point	Ctrl+Shift+F5	Attach Window to a Connect	on			_
• •	<u>P</u> rint	Ctrl+P			\		
	Recent <u>F</u> iles Recent Wor <u>k</u> spaces	· · · ·					
ack	Recent Images				2		
	Close <u>W</u> indow			atch2 🖌 Watch3 🖌 🔳			T
	E <u>x</u> it			^ ^_			
ŧ							
н.							
E No)ne>						
<u></u>	Crnd StdlO Build	I / FileFind / Src	Ctrl /*Log /			4	Þ
Select	t target(s) to connect to				Ln 1, Col 1		NUM //

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						
	Ilapse All t simulator pand Vehicles erface (parallel port) ol (serial port) d/Remove/Edit Devices hest Device File						
ARMOAK_M ARM-VIA-LP ARM-VIA-LP ARM-ARM-DIR ARM-ARM-DIR CPB926EJ-S_U.	ARM Vehicle Macraigor Wiggler ARM Ltd. Direct Connection						

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	IDI Target List			×
	Use the check boxes t	o add or remove	e RDI targets from the connection manager:	
	Name	Version	Description	
	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	
	<u> </u>			
		Add DLL	Reset list Configure Remove Duplicate.	
			Close	

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 more information about the generic setup of J-Link RDI, please refer to *Configu*-



ration on page 102	ration	on	page	182.
--------------------	--------	----	------	------

R	DI Target List		×						
Use the check boxes to add or remove RDI targets from the connection manager:									
	Name	Version	Description						
	🗹 🏶 JLinkRDI		J-Link ARM RDI Interface						
	🗹 🌮 Remote_A	v1.2	Angel debug protocol (serial port)						
	🗹 愛 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. Make sure your target hardware is already connected to J-Link.
 8. In the Connection control dialog, expand the JLink ARM RDI Interface and select the ARM_0 Processor. Close the Connection Control Window.

Name	Description
- 🗛 ARM-A-RR	ARM Ltd. RDI targets
🕂 🥵 ARMulator	ARM instruction set simulator
🗄 🕵 JLinkRDI.dll	J-Link ARM RDI Interface
[⊥]	ARM on localhost
- 🗛 Server	Connection Broker
🕂 🔁 localhost	Simulator Broker
- 🤬 ARM-VIA-LP	Motorola/Macraigor Wiggler emulator
🗄 🕵 MOT_WIGGLER	Macraigor Wiggler
- 🤬 ARM-ARM-DIR	ARM Ltd. Direct Connection
In Stranger Here II - Stranger	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>						
File Edit Find View Project Tools Debug Help						
	0 00 12	- 100 000		🖼 🌮 🗶	State:	Stopped
File: \$NO_SOURCE Find:	+++ 201					
		A	Regist	٥r		▼
No source for context: _ENTRY_< <entry point=""> Click to Load 'C:\temp\embOS Start STR71x\RAM\Start STR71x.axf'</entry>			RO	0000000	R1	000000
			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	00000000	PC	000000
				00000D3		
			MZC3	Core Deb	UCI /	
Dsm Src (rtosinit_str71x.c+(vectors.s+)			<u>نات</u>](<u></u> /		الشارك الكر
☑ Name Value ☑ Name Value		<noaddr></noaddr>				
<pre><0x0000000 <unknown location=""></unknown></pre>	Ē	<noaddr></noaddr>				
		<noaddr></noaddr>				
		<noaddr) <noaddr)< th=""><th></th><th></th><th></th><th></th></noaddr)<></noaddr) 				
		<noaddr) <noaddr)< td=""><td></td><td></td><td></td><td></td></noaddr)<></noaddr) 				
	2	<noaddr)< th=""><th></th><th></th><th></th><th></th></noaddr)<>				
	i i i i	<noaddr></noaddr>				
Call Stack (Locals (Statics)		Molddro				<u> </u>
connect,route 2						
<pre>> connect 10</pre>						
Advanced_info searched in: Local Advanced_info						
Using Advanced info based on 'Default' or 'All'						
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						
3 Stop>			_			
Crnd Stallo / Build / FileFind / SrcCtrl / Log /			4			-
— Currently opened file	Ln 1, Col	1			N	

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

BVDEBUG(Start_STR71x) = @ARM_0:ARM-A-RR File Edit Find View Project Tools Debug Help						l	. 🗆 ×
	(🗳 🏝 🗈	36 36 R	-		🌮 📥 s	tate: Stop	ped
File: main_led.c Find:	-						
<pre>while (1) { LED_ToggleLED1(); OS_Delay (200); OS_Delay (200); /* /** main * int main(void) { OS_IncD1(); /* Initially disable interrupts OS_InitHW(); /* initialize US OS_InitHW(); /* initialize LED ports /* You need to create at least one task here ! OS_CREATETASK(aTCB0, "HP Task", Task1, 50, Stack1); OS_CREATETASK(aTCB1, "LP Task", Task1, 50, Stack1</pre>	***/ */ */ */ */ */ */ */ */ */ */ */ */	NZ(000 1 T 1 T 1 T	0000058 0000063 0000000 0000000 00000289 20000051 00001F8 SR 000000F CV FIQ IRQ 00 DIS DIS JSR IRQ FIQ SVC SVC SVC SVC SVC SVC SVC SVC SVC SVC	0 R3 C R5 0 R7 0 R9 0 R11 5 PC 3 STATE 1 Thumb : 0xE59F1 0xE59F1 0x00000 0x00000	F018 0xE5 F014 0xE5 0040 0x00 004C 0xEA	0 0 0 0 4 9 9 9 7 7 0 4 9 9 7 7 0 4 0 0 0 0 4	
		xEA000000	ALROOU / DD	OABZOI	COLO OXEO	500000	
<pre>> bi \MAIN_LED\#35:0 > go Stopped at 0x00000514 due to SW Instruction Breakpoint Stopped at 0x00000514: MAIN_LED\main Line 35</pre>							
Stop> Cmd StallO / Build / FileFind / SrcCtrl /*Log /			4				▶▼
For more information, select Help from Menu		Ln 35, C	ol 9			NUM	

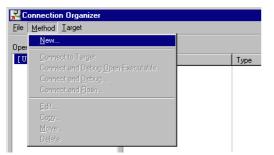
8.3.4 GHS MULTI

8.3.4.1 Software version

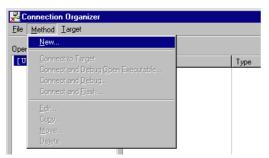
J-Link RDI has been tested with GHS MULTI version 4.07. There should be no problems with other versions of GHS MULTI. All screenshots are taken from GHS MULTI version 4.07.

8.3.4.2 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	e w Connection Method
Name:	J-Link
Type:	Custom
	Create Cancel

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 the standard J-Link installation path or another path that includes spaces has been used, enclose the path in quotation marks.

Example:

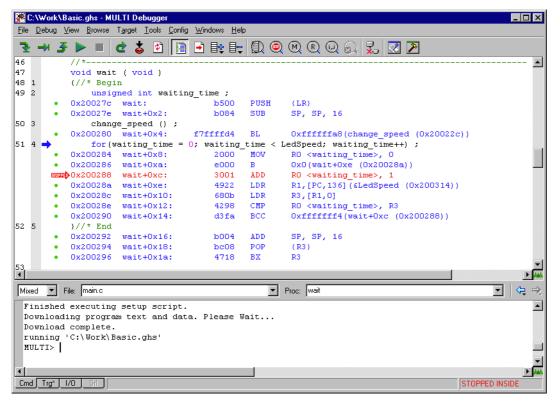
-config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll" Refer to GHS manual "MULTI: Configuring Connections for ARM Targets", chapter "ARM Remote Debug Interface (rdiserv) Connections" for a complete list of possible arguments.

Connection Editor						
Name:	J-Link					
Туре:	Custom					
🗖 Log Conr	nection to file:					
MULTI Targe	et Setup Script: 🛛 🖉					
Connect for:	 Download (Download and debug application) Attach (Debug application already on target) 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 the choices by clicking the **Apply** button afterwards the **Connect** button.

Connection Editor						
Name:	J-Link					
Туре:	Custom					
🔲 Log Conr	nection to file:					
MULTI Targe	et Setup Script: 🛛 🖉					
Connect for:	onnect for: © Download (Download and debug application) © Attach (Debug application already on target) © Board Setup (Debug board initialization sequence)					
Server:	rdiserv					
Arguments:	-config -dll ''C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll''					
mode=download rdiserv -config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll"						
Connect	OK Cancel Revert Apply					

- 6. The **J-Link RDI Configuration** dialog will be opened. For more information about the generic setup of J-Link RDI, please refer to *Configuration* on page 182.
- 7. Click the **OK** button to connect to the target. Build the project and start the debugger. Note that at least one action (for example **step** or **run**) has to be performed in order to initiate the download of the application.



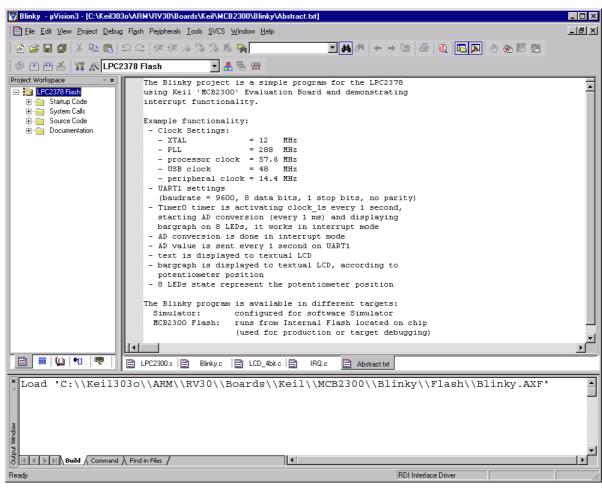
8.3.5 KEIL MDK (µVision IDE)

8.3.5.1 Software version

J-Link has been tested with KEIL MDK 3.34. There should be no problems with other versions of KEIL μ Vision. All screenshots are taken from MDK 3.34.

8.3.5.2 Configuring to use J-Link RDI

Start KEIL uVision and open the project.



Select **Project** | **Options for Target** '<NameOfTarget>' 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	Elinerface Driver VLINK ARM Debugger ULINK Contex-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 Toolbox Watchpoints & PA Memory Display	Restore Debug Session Settings Gradpoints Watchpoints Memory Display
CPU DLL: Parameter: SARM.DLL -cLPC2100	Driver DLL: Parameter:
Dialog DLL: Parameter: DARMP.DLL -pLPC2378	Dialog DLL: Parameter: TARMP.DLL
ОК Са	ncel 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\JLinkAR	M_V359a\JLinkRDI.dll	
Browse for ToolConf File		
Debug		
Cache Options Cache <u>C</u> ode Cache <u>M</u> emory	Configure <u>R</u> DI Driver	
	OK Cancel	<u>H</u> elp

The J-Link RDI Configuration dialog will be opened.For more information about the generic setup of J-Link RDI, please refer to *Configuration* on page 182.

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

prisio	n3 - [Disassembly]			-	
<u> </u>	<u>Project D</u> ebug Fl <u>a</u> sh Pe <u>r</u> iph	erals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp		-	Ð×
12 🚘 🖬 🎒	※ 階 🛍 ユ ユ 律	∉ ∧ % % % % %	r		
RST 📃 🔕 🖓	⊕ ()+ *() ⇔ ∺* ()*	0. 💭 🖤 😹 🔲 🗄 🚾 🤰	•		
Project Workspace	⊤ x	002. Veccors	LDR	PC, Reset_Addr	
Register	Value	⇒ 0x 00000000 E59FF 018 333:		PC,[PC,#0x0018] PC, Undef Addr	
Current	0.00000000	0x0000004 E59FF018		PC,[PC,#0x0018]	
B1	0x00000000 0x00000000	334:	LDR	PC, SWI_Addr	
	0x00000000	0x0000008 E59FF018 335:		PC,[PC,#0x0018]	
R3	0x00000000	0x000000C E59FF018		PC, PAbt_Addr PC, [PC,#0x0018]	
R4	0x00000000	336:	LDR	PC, DAbt Addr	
	0x00000000	0x00000010 E59FF018		PC,[PC,#0x0018]	
	0x00000000 0x00000000	337: 338: ;	NOP LDR	; Reserved Vector	
	0x00000000	0x00000014 B9206E50		PC, IRQ_Addr R0!,{R4,R6,R9-R11,R13-R14}	
	0x00000000	339:	LDR	PC, [PC, #-0x0120] ; Vector from VicVectAddr	
R10	0x00000000	0x00000018 E51FF120		PC,[PC,#-0x0120]	
	0x00000000	340: 341:	LDR	PC, FIQ_Addr	
R12	0x0000000	341: 342: Reset Addr	DCD	Reset Handler	
R13 (SP) R14 (LR)	0x00000000 0x00000000	343: Undef_Addr	DCD	Undef_Handler	
R15 (PC)	0x00000000	344: SWI_Addr	DCD	SWI_Handler	
	0x000000d3	345: PAbt_Addr	DCD	PAbt_Handler	
	0x00000010	346: DAbt_Addr 347:	DCD DCD	DAbt_Handler 0 : Reserved Address	
⊞ User/System		348: IRQ Addr	DCD	IRQ Handler	
+ Fast Interrupt		349: FIQ Addr	DCD	FIQHandler	
				· · · · · · · · · · · · · · · · · · ·	-
Interrupt Curporuiser	•				• •
F	▼ ∭ B ♥ F ♥ Te		🖹 LCD_4bit.c		•
Euroruisor	- 1 1 1	LPC2300.s Blinky.c		c E IRQ.c E Abstract.txt Q Disassembly	•
Euroruisor	- 1 1 1			CB230	
Euroruisor	- 1 1 1	LPC2300.s Blinky.c		IRQ.c Abstract.bst Q Disassembly CB230 X Address: 0x3fffc000 0x3ffFC000: 00000000 00000000	
Euroruisor	- 1 1 1	LPC2300.s Blinky.c		Address: Ox3ffFC000 0x3FFFC000: 00000000 0x3FFFC008: 00000000	
Euroruisor	- 1 1 1	LPC2300.s Blinky.c		Address: Ox3fffc000 0x3FFFC000: 00000000 0x3FFFC008: 00000000 0x3FFFC010: 600000DF	
Euroruisor	- 1 1 1	LPC2300.s Blinky.c		Address: Ox3ffC000 0x3FFFC000: 00000000 00000000 0x3FFFC08: 0000000 00000000 0x3FFFC010: 600000DF 00000000 0x3FFFC018: 00000000 00000000	
Euroruisor	- 1 1 1	LPC2300.s Blinky.c		Address: Disassembly CB230 Address: 0x3ffFc000 0x3FFFC000: 00000000 00000000 0x3FFFC010: 600000DF 00000000 0x3FFFC018: 0000000 0000000 0x3FFFC020: 600000DF 00000000	
Load "C:	- 1 1 1	LPC2300.s Blinky.c		Address Ox3fffc000 0x3FFFC000: 00000000 0000000 0x3FFFC010: 00000000 0000000 0x3FFFC010: 600000DF 00000000 0x3FFFC010: 600000DF 00000000 0x3FFFC020: 600000DF 00000000 0x3FFFC020: 600000DF 00000000	
Load "C:	\\Keil303o\\AR	LPC2300.s Blinky.c M\\RV30\\Boards\\K	eil\\MC	Address: 0x3fffc000 0x3 FFFC000: 00000000 00000000 0x3 FFFC008: 00000000 00000000 0x3 FFFC010: 600000DF 00000000 0x3 FFFC010: 600000DF 00000000 0x3 FFFC010: 600000DF 00000000 0x3 FFFC020: 600000DF 00000000 0x3 FFFC030: 00000000 00000000	
Load "C:	\\Keil303o\\AR	LPC2300.s Blinky.c	eil\\MC	Address: 0x3ffFC000 0x3 FFFC000: 00000000 00000000 0x3 FFFC010: 600000DF 00000000 0x3 FFFC018: 00000000 00000000 0x3 FFFC018: 00000000 00000000 0x3 FFFC018: 0000000 00000000 0x3 FFFC018: 0000000 00000000 0x3 FFFC018: 00000000 00000000 0x3 FFFC038: 00000000 00000000 0x3 FFFC038: 600000DF 00000000	
Load "C:	\\Keil303o\\AR	LPC2300.s Blinky.c M\\RV30\\Boards\\K	eil\\MC	.c Imple: IRQ.c Abstract.bt Disassembly CB230 X Address: 0x3fFfc000 0x3FFFC008: 00000000 00000000 0x3FFFC010: 600000DF 00000000 0x3FFFC018: 0000000 00000000 0x3FFFC028: 0000000 00000000 0x3FFFC028: 0000000 00000000 0x3FFFC038: 00000000 00000000 0x3FFFC038: 600000DF 00000000	

8.4 Configuration

This section describes the generic setup of J-Link RDI (same for all debuggers) using the J-Link RDI configuration dialog.

8.4.1 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>.

8.4.2 Using different configurations

It can be desirable to use different configurations for different targets. If this is the case, a new folder needs to be created and the <code>JLinkARM.dll</code> as well as the <code>JLinkRDI.dll</code> needs to be copied into it.

Project A needs to be configured to use JLinkRDI.dll A in the first folder, project B needs to be configured to use the DLL in the second folder. Both projects will use separate configuration files, stored in the same directory as the DLLs they are using.

If the debugger allows using a project-relative path (such as IAR 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.

8.4.3 Using mutliple J-Links simulatenously

Same procedure as using different configurations. Each debugger session will use their own instance of the $\tt JLinkRDI.dll$.

8.4.4 Configuration dialog

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

8.4.4.1 General tab

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 "FlashDI") and to set an unlimited number of software breakpoints in flash (Requires the add. license "FlashBP"). Connection to J-Link Image: State Sta	
<u>Beset Config</u>	
OK Cancel <u>Apply</u>	

Connection to J-Link

This setting allows the user to configure how the DLL should connect to the J-Link. Some J-Link models also come with an Ethernet interface which allows to use an emulator remotely via TCP/IP connection.

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.

J-	Link RDI License management		×
	Feature	Serial number	Expiration
	<u>Add license</u> <u>D</u> elete lice	nse Display <u>s</u> erial nu	mber OK

2. Click the Add license button and enter your license. Confirm your input by click-

ing the **OK** button.

Add licens	se	×
Please ent	iter your license(s)!	
<u>L</u> icense		
	ОКСС	ancel

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

J-	Link RDI License management		×
	Feature	Serial number	Expiration
	RDI	1	never expires
	Add license Delete lice	ense Display <u>s</u> erial nur	mber OK

8.4.4.2 Init tab

J-Link RDI Configuration
General Init JTAG Flash Breakpoints CPU Log
OK Cancel Apply

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 to use the PLL before the target application is downloaded, in order to speed up the download.

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);	Dende a 9/16/22 hit value
Read16(Addr);	Reads a 8/16/32 bit value, Addr = address to read (as hex value)
Read32(Addr);	Addr – address to read (as nex value)
<pre>Verify8(Addr, Data);</pre>	Verifies a 8/16/32 bit value,
<pre>Verify16(Addr, Data);</pre>	Addr = address to verify (as hex value)
<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)
<pre>Write32(Addr, Data);</pre>	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 8.2: Macro file commands	

Example of macro file

8.4.4.3 JTAG tab

J-Link RDI Configuration	? ×
General Init JTAG	Flash Breakpoints CPU Log
JTAG speed	
○ <u>A</u> uto selection	
C Adaptive clocking	
JTAG scan chain wit	h multiple devices
Position 0	jB len 0
0 is closest to TDI.	Sum of IRLens of devices closer to TD1. IRLen of ARM chips is 4.
	⊻erity JTAG config
	OK Cancel Apply

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

For more information about the different speed settings supported by J-Link, please refer to *JTAG Speed* on page 104.

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.

8.4.4.4 Flash tab

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		
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 Apply		

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.

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.

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.

J-Link flash programming		
Finished flash programming.		

8.4.4.5 Breakpoints tab

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 Apply	

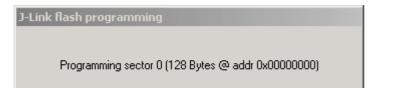
Use software breakpoints

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.

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.



8.4.4.6 CPU tab

J-Link RDI Configuration ? 🗙
General Init JTAG Flash Breakpoints CPU Log Image: Allow instruction set simulation Allows instruction set simulation Allows the emulator to simulate individual instructions when single stepping instructions. This does not normally have any disadvantages and makes debugging much faster, especially when using flash breakpoints.
Reset strategy 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. Hardware, halt after reset (normal) Pelay after reset 0 ms 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.
OK Cancel Apply

Instruction set simulation

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

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.

For more information about the different reset strategies which are supported by J-Link and why different reset strategies are necessary, please refer to *Reset strategies* on page 119.

8.4.4.7 Log tab

A log file can be generated for the J-Link DLL and for the J-Link RDI DLL. 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 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 (FFFFF130,0004) -- Data: 00 00 00 00 060:234 (0001) ARM_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 060:236 (0000) ARM_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 060:237 (0000) ARM_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 060:238 (0001) ARM_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 060:239 (0001) ARM_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 060:240 (0001) ARM_ReadMem (FFFFF130,0004) -- Data: 00 00 00 00 060:241 (0001) ARM_WriteMem(FFFFFD44,0004) -- Data: 00 80 00 00 - Writing 0x4 bytes @ 0xFFFFFD44 >195> 060:277 (0000) ARM_WriteMem(0000000,0178) -- Data: 0F 00 00 EA FE FF FF EA ... 060:277 (0000) ARM_WriteMem(000003C4,0020) -- Data: 01 00 00 00 02 00 00 00 ... Writing 0x178 bytes @ 0x00000000 060:277 (0000) ARM_WriteMem(000001CC,00F4) -- Data: 30 B5 15 48 01 68 82 68 ... -Writing 0x20 bytes @ 0x000003C4 060:277 (0000) ARM_WriteMem(000002C0,0002) -- Data: 00 47 060:278 (0000) ARM_WriteMem(000002C4,0068) -- Data: F0 B5 00 27 24 4C 34 4D ... -Writing 0xF6 bytes @ 0x000001CC 060:278 (0000) ARM_WriteMem(0000032C,0002) -- Data: 00 47 060:278 (0000) ARM_WriteMem(00000330,0074) -- Data: 30 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 ... -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.

. . .

8.5 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.

8.5.1 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.

8.5.2 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.

8.5.2.1 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.

8.5.3 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 193)

8.5.3.1 DCC semihosting

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

8.5.4 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.

8.5.4.1 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.

8.5.5 Unexpected / unhandled SWIs

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

J-Link RDI 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.
	<u>Yes</u> <u>N</u> o Cancel

© 2004-2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 9 Device specifics

This chapter describes for which devices some special handling is necessary to use them with J-Link.

9.1 Analog Devices

J-Link has been tested with the following MCUs from Analog Devices:

- AD7160
- ADuC7020x62
- ADuC7021x32
- ADuC7021x62
- ADuC7022x32
- ADuC7022x62
- ADuC7024x62
- ADuC7025x32
- ADuC7025x62
- ADuC7026x62
- ADuC7027x62
- ADuC7028x62
- ADuC7030
- ADuC7031
- ADuC7032
- ADuC7033
- ADuC7034
- ADuC7036
- ADuC7038
- ADuC7039
- ADuC7060
- ADuC7061
- ADuC7062
- ADuC7128
- ADuC7120
 ADuC7129
- ADuC7229x126
- ADuCRF02
- ADuCRF101

9.1.1 ADuC7xxx

9.1.1.1 Software reset

A special reset strategy has been implemented for Analog Devices ADuC7xxx MCUs. This special reset strategy is a software reset. "Software reset" means basically RESET pin is used to perform the reset, the reset is initiated by writing special function registers via software.

The software reset for Analog Devices ADuC7xxxx executes the following sequence:

- 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.

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

This information is applicable to the following devices:

- Analog ADuC7020x62
- Analog ADuC7021x32
- Analog ADuC7021x62
- Analog ADuC7022x32
- Analog ADuC7022x62
- Analog ADuC7024x62
- Analog ADuC7025x32
- Analog ADuC7025x62

- Analog ADuC7026x62 ٠
- Analog ADuC7027x62 •
- Analog ADuC7030 ٠
- Analog ADuC7031 ٠
- Analog ADuC7032 ٠
- Analog ADuC7033 Analog ADuC7128 ٠
- ٠
- Analog ADuC7129 ٠
- Analog ADuC7229x126 ٠

9.2 ATMEL

J-Link has been tested with the following ATMEL devices:

- AT91SAM3A2C
- AT91SAM3A4C
- AT91SAM3A8C
- AT91SAM3N1A
- AT91SAM3N1B
- AT91SAM3N1C
- AT91SAM3N2A
- AT91SAM3N2B
- AT91SAM3N2C
- AT91SAM3N4A
- AT91SAM3N4B
- AT91SAM3N4C
- AT91SAM3S1A
- AT91SAM3S1B
- AT91SAM3S1C
- AT91SAM3S2A
- AT91SAM3S2B
- AT91SAM3S2C
- AT91SAM3S4A
- AT91SAM3S4B
- AT91SAM3S4C
- AT91SAM3U1C
- AT91SAM3U2C
- AT91SAM3U4C
- AT91SAM3U1E
- AT91SAM3U2E
- AT91SAM3U4E
- AT91SAM3X2C
- AT91SAM3X2E
- AT91SAM3X2G
- AT91SAM3X2H
- AT91SAM3X4C
- AT91SAM3X4E
- AT91SAM3X4G
- AT91SAM3X4H
- AT91SAM3X8C
- AT91SAM3X8E
- AT91SAM3X8G
- AT91SAM3X8H
- AT91SAM7A3
- AT91SAM7L64
- AT91SAM7L128
- AT91SAM7S16
- AT91SAM7S161
- AT91SAM7S32
- AT91SAM7S321
- AT91SAM7S64
- AT91SAM7S128
- AT91SAM7S256
- AT91SAM7S512
- AT91SAM7SE32
- AT91SAM7SE256
- AT91SAM7SE512
- AT91SAM7X128
- AT91SAM7X256
- AT91SAM7X512
- AT91SAM7XC128
- AT91SAM7XC256

- AT91SAM7XC512
- AT91SAM9XE128
- AT91SAM9XE256

9.2.1 AT91SAM7

9.2.1.1 Reset strategy

The reset pin of the device is per default disabled. This means that the reset strategies which rely on the reset pin (low pulse on reset) do not work per default. For this reason a special reset strategy has been made available.

It is recommended to use this reset strategy. This special reset strategy resets the peripherals by writing to the RSTC_CR register. Resetting the peripherals puts all peripherals in the defined reset state. This includes memory mapping register, which means that after reset flash is mapped to address 0. It is also possible to achieve the same effect by writing 0x4 to the RSTC_CR register located at address 0xffffd00.

This information is applicable to the following devices:

- AT91SAM7S (all devices)
- AT91SAM7SE (all devices)
- AT91SAM7X (all devices)
- AT91SAM7XC (all devices)
- AT91SAM7A (all devices)

9.2.1.2 Memory mapping

Either flash or RAM can be mapped to address 0. After reset flash is mapped to address 0. In order to majlink_supported_devices.html RAM to address 0, a 1 can be written to the RSTC_CR register. Unfortunately, this remap register is a toggle register, which switches between RAM and flash with every time bit zero is written.

In order to achieve a defined mapping, there are two options:

- 1. Use the software reset described above.
- 2. Test if RAM is located at 0 using multiple read/write operations and testing the results.

Clearly 1. is the easiest solution and is recommended.

This information is applicable to the following devices:

- AT91SAM7S (all devices)
- AT91SAM7SE (all devices)
- AT91SAM7X (all devices)
- AT91SAM7XC (all devices)
- AT91SAM7A (all devices)

9.2.1.3 Recommended init sequence

In order to work with an ATMEL AT91SAM7 device, it has to be initialized. The following paragraph describes the steps of an init sequence. An example for different software tools, such as J-Link GDB Server, IAR Workbench and RDI, is given.

- Set JTAG speed to 30kHz
- Reset target
- Perform peripheral reset
- Disable watchdog
- Initialize PLL
- Use full JTAG speed

Samples

GDB Sample

```
# connect to the J-Link gdb server
target remote localhost:2331
monitor flash device = AT91SAM7S256
monitor flash download = 1
monitor flash breakpoints = 1
# Set JTAG speed to 30 kHz
monitor endian little
monitor speed 30
# Reset the target
monitor reset 8
monitor sleep 10
# Perform peripheral reset
monitor long 0xFFFFD00 = 0xA5000004
monitor sleep 10
# Disable watchdog
monitor long 0xFFFFD44 = 0x00008000
monitor sleep 10
# Initialize PLL
monitor long 0xFFFFFC20 = 0x00000601
monitor sleep 10
monitor long 0xFFFFFC2C = 0x00480a0e
monitor sleep 10
monitor long 0xFFFFC30 = 0x00000007
monitor sleep 10
monitor long 0xFFFFF60 = 0x00480100
monitor sleep 100
# Setup GDB for faster downloads
#set remote memory-write-packet-size 1024
set remote memory-write-packet-size 4096
set remote memory-write-packet-size fixed
monitor speed 12000
break main
load
continue
```

IAR Sample

```
_Init()
*
* /
Init() {
 __emulatorSpeed(30000);
                                             // Set JTAG speed to 30 kHz
// Perform peripheral reset
 ___writeMemory32(0xA5000004,0xFFFFFD00,"Memory");
 ___sleep(20000);
 __writeMemory32(0x00008000,0xFFFFFD44,"Memory");
                                             // Disable Watchdog
 ___sleep(20000);
 __writeMemory32(0x0000601,0xFFFFFC20,"Memory");
                                             // PLL
 ___sleep(20000);
 ___writeMemory32(0x10191c05,0xFFFFFC2C,"Memory");
                                             // PLL
 ___sleep(20000);
 __writeMemory32(0x0000007,0xFFFFFC30,"Memory");
                                             // PLL
 ___sleep(20000);
 __writeMemory32(0x002f0100,0xFFFFF60,"Memory");
                                             // Set 1 wait state for
 ___sleep(20000);
                                             // flash (2 cycles)
  __emulatorSpeed(12000000);
                                             // Use full JTAG speed
}
*
*
      execUserReset()
*/
execUserReset() {
  _message "execUserReset()";
 _Init();
}
*
      execUserPreload()
*/
execUserPreload() {
 __message "execUserPreload()";
 _Init();
}
```

RDI Sample

SetJTAGSpeed(30); Reset(0, 0); Write32(0xFFFFFD00, 0xA5000004); // Perform peripheral reset Write32(0xFFFFFD44, 0x00008000); // Disable watchdog Write32(0xFFFFFC20, 0x00000601); // Set PLL Delay(200); Write32(0xFFFFFC2C, 0x00191C05); Delay(200);// Select master clock anWrite32(0xFFFFFC30, 0x0000007);// Select master clock anWrite32(0xFFFFFF60, 0x00320300);// Set flash wait states SetJTAGSpeed(12000);

// Set JTAG speed to 30 kHz // Set PLL and divider

// Select master clock and processor clock

9.2.2 AT91SAM9

9.2.2.1 JTAG settings

We recommend using adaptive clocking.

This information is applicable to the following devices:

- AT91RM9200 •
- AT91SAM9260 •
- AT91SAM9261 •
- AT91SAM9262 •
- AT91SAM9263 •

9.3 DSPGroup

J-Link has been tested with the following DSPGroup devices:

• DA56KLF

9.4 Ember

J-Link has been tested with the following Ember devices:

- EM351
- EM357

9.5 Energy Micro

J-Link has been tested with the following Energy Micro devices:

- EFM32G200F16
- EFM32G200F32
- EFM32G200F64
- EFM32G210F128
- EFM32G230F32
- EFM32G230F64
- EFM32G230F128
- EFM32G280F32
- EFM32G280F64
 EFM32G280F128
- EFM32G280F128
 EFM32G290F32
- EFM32G290F32
 EFM32G290F64
- EFM32G290F04
 EFM32G290F128
- EFM32G840F32
- EFM32G840F64
- EFM32G840F128
- EFM32G880F32
- EFM32G880F52
 EFM32G880F64
- EFM32G880F64
- EFM32G880F128
- EFM32G890F32
- EFM32G890F64
- EFM32G890F128
- EFM32TG108F4
- EFM32TG108F8
- EFM32TG108F16
- EFM32TG108F32
- EFM32TG110F4
 EFM32TG110F8
- EFM32TG110F8
 EFM32TG110F16
- EFM32TG110F32
- EFM32TG110F32
 EFM32TG210F8
- EFM32TG210F16
- EFM32TG210F32
- EFM32TG230F8
- EFM32TG230F16
- EFM32TG230F32
- EFM32TG250F52
 EFM32TG840F8
- EFM32TG840F16
- EFM32TG840F32

9.6 Freescale

J-Link has been tested with the following Freescale devices:

- MAC7101
- MAC7106
- MAC7111
- MAC7112
- MAC7116
- MAC7121
- MAC7122
- MAC7126
- MAC7131
- MAC7136
- MAC7141
- MAC7142
- MK10DN512
- MK10DX128
 MK10DX256
- MK10DX256
 MK20DN512
- MK20DN512
 MK20DX128
- MK20DX128
 MK20DX256
- MK20DX230
 MK30DN512
- MK30DN312
 MK30DX128
- MK30DX128
 MK30DX256
- MK30DX230
 MK40N512
- MK40N312
 MK40X128
- MK40X120
 MK40X256
- MK50DN512
- MK50DX256
- MK50DN512
- MK50DX256
- MK51DX256
- MK51DN512
- MK51DX256
- MK51DN512
- MK51DN256
- MK51DN512
- MK52DN512
- MK53DN512
- MK53DX256
- MK60N256
- MK60N512
- MK60X256

9.6.1 Kinetis family

9.6.2 Unlocking

If your device has been locked by setting the MCU security status to "secure", and mass erase via debug interface is not disabled, J-Link is able to unlock your Kinetis K40/K60 device. The device can be unlocked by using the "unlock" command in J-Link Commander.

For more information regarding the MCU security status of the Kinetis devices, please refer to the user manual of your device.

9.6.3 Tracing

The first silicon of the Kinetis devices did not match the data setup and hold times which are necessary for ETM-Trace. On these devices, a low drive strength should be configured for the trace clock pin in order to match the timing requirements.

On later silicons, this has been corrected.

The J-Link software and documentation package comes with a sample project for the Kinetis K40 and K60 devices which is pre-configured for the TWR-40 and TWR-60 eval boards and ETM / ETB Trace. This sample project can be found at SamplesJLinkProjects.

9.7 Fujitsu

J-Link has been tested with the following Fujitsu devices:

- MB9AF102N
- MB9AF102R
- MB9AF104N
- MB9AF104R
- MB9BF104N
- MB9BF104R
- MB9BF105N
- MB9BF105R
- MB9BF106N
- MB9BF106R
- MB9BF304N
- MB9BF304R
- MB9BF305N
- MB9BF305R
- MB9BF306N
- MB9BF306R
- MB9BF404N
- MB9BF404R
- MB9BF405N
- MB9BF405R
- MB9BF406N
- MB9BF406R
- MB9BF504N
- MB9BF504R
- MB9BF505N
- MB9BF505R
- MB9BF506N
- MB9BF506R

9.8 Itron

J-Link has been tested with the following Itron devices:

TRIFECTA

9.9 Luminary Micro

J-Link has been tested with the following Luminary Micro devices:

- LM3S101
- LM3S102
- LM3S301
- LM3S310
- LM3S315
- LM3S316
- LM3S317
- LM3S328
- LM3S601
- LM3S610
 LM3S611
- LM3S611
- LM3S612LM3S613
- LM3S615
- LM3S617
- LM3S618
- LM3S628
- LM3S801
- LM3S811
- LM3S812
- LM3S815
- LM3S817
- LM3S818
- LM3S828
- LM3S2110
- LM3S2139
- LM3S2410
- LM3S2412
- LM3S2432
- LM3S2533
- LM3S2620
- LM3S2637
- LM3S2651
- LM3S2730
- LM3S2739
- LM3S2939
- LM3S2948
- LM3S2950
- LM3S2965
- LM3S6100
- LM3S6110
- LM3S6420
- LM3S6422
- LM3S6432
- LM3S6610
- LM3S6633
- LM3S6637
- LM3S6730
- LM3S6938
- LM3S6952
- LM3S6965

9.9.1 Unlocking LM3Sxxx devices

If your device has been "locked" accidentially (e.g. by bad application code in flash which mis-configures the PLL) and J-Link can not identify it anymore, there is a special unlock sequence which erases the flash memory of the device, even if it can not be identified. This unlock sequence can be send to the target, by using the "unlock" command in J-Link Commander.

9.10 NXP

J-Link has been tested with the following NXP devices:

- LPC1111
- LPC1113
- LPC1311
- LPC1313
- LPC1342
- LPC1343
- LPC1751
- LPC1751
- LPC1752
- LPC1754
 LPC1756
- LPC1756LPC1758
- LPC1758
 LPC1764
- LPC1765
- LPC1766
- LPC1768
- LPC2101
- LPC2102
- LPC2103
- LPC2104
- LPC2105
- LPC2106
- LPC2109
- LPC2114
- LPC2119
- LPC2124
- LPC2129
- LPC2131
- LPC2132
- LPC2134LPC2136
- LPC2130
- LPC2141
- LPC2142
- LPC2144
- LPC2146
- LPC2148
- LPC2194
- LPC2212
- LPC2214
- LPC2292
- LPC2294
- LPC2364
- LPC2366
- LPC2368
- LPC2378
- LPC2468
- LPC2478
- LPC2880
- LPC2888
- LPC2917
- LPC2919
- LPC2927
- LPC2929
- PCF87750
- SJA2010
- SJA2510

9.10.1 LPC ARM7-based devices

9.10.1.1 Fast GPIO bug

The values of the fast GPIO registers can not be read direct via JTAG from a debugger. The direct access to the registers corrupts the returned values. This means that the values in the fast GPIO registers normally can not be checked or changed from a debugger.

Solution / Workaround

J-Link supports command strings which can be used to read a memory area indirect. Indirectly reading means that a small code snippet will be written into RAM of the target device, which reads and transfers the data of the specified memory area to the debugger. Indirectly reading solves the fast GPIO problem, because only direct register access corrupts the register contents.

Define a 256 byte aligned area in RAM of the LPC target device with the J-Link command map ram and define afterwards the memory area which should be read indirect with the command map indirectread to use the indirectly reading feature of J-Link. Note that the data in the defined RAM area is saved and will be restored after using the RAM area.

This information is applicable to the following devices:

- LPC2101
- LPC2102
- LPC2103
- LPC213x/01
- LPC214x (all devices)
- LPC23xx (all devices)
- LPC24xx (all devices)

Example

J-Link commands line options can be used for example with the C-SPY debugger of the IAR Embedded Workbench. Open the **Project options** dialog and select **Debug-ger**. Select **Use command line options** in the **Extra Options** tap and enter in the textfield --jlink_exec_command "map ram 0x4000000-0x40003fff; map indirectread 0x3fffc000-0x3fffcfff; map exclude 0x3fffd000-0x3ffffff;" as shown in the screenshot below.

Options for node "Proj Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link/J-Trace LMI FTDI Macraigor RDI Third-Party Driver	Setup Download Extra Options Plugins Factory Settings Use command line options Command line options: (one per line) jlink_exec_command "map ram 0x40000000-0x40003fff; map indire =
RDI	OK Cancel

With these additional commands are the values of the fast GPIO registers in the C-SPY debugger correct and can be used for debugging. For more information about J-Link command line options refer to subchapter *Command strings* on page 135.

9.10.1.2 RDI

J-Link comes with a device-based RDI license for NXP LPC21xx-LPC24xx devices. This means the J-Link RDI software can be used with LPC21xx-LPC24xx devices free of charge. For more information about device-based licenses, please refer to *License types* on page 50.

9.10.2 Reset (Cortex-M3 based devices)

For Cortex-M3 based NXP LPC devices the reset itself does not differ from the one for other Cortex-M3 based devices: After the device has been reset, the core is halted before any instruction is performed. For the Cortex-M3 based LPC devices this means the CPU is halted before the bootloader which is mapped at address 0 after reset.

The user should write the memmap register after reset, to ensure that user flash is mapped at address 0. Moreover, the user have to correct the Stack pointer (R13) and the PC (R15) manually, after reset in order to debug the application.

9.10.3 LPC288x flash programming

In order to use the LPC288x devices in combination with the J-Link flash download feature, the application you are trying to debug, should be linked to the original flash (a) addr 0x10400000. Otherwise it is user's responsibility to ensure that flash is remapped to 0x0 in order to debug the application from addr 0x0.

9.11 OKI

J-Link has been tested with the following OKI devices:

- ML67Q4002 ML67Q4003 ٠
- •
- ML67Q4050 •
- ML67Q4051 ٠
- ML67Q4060 ٠
- ML67Q4061 •

9.12 Renesas

J-Link has been tested with the following Renesas devices:

- R5F56104
- R5F56106
- R5F56107
- R5F56108
- R5F56216
- R5F56217
- R5F56218
- R5F562N7
- R5F562N8
- R5F562T6
- R5F562T7
- R5F562TA

9.13 Samsung

J-Link has been tested with the following Samsung devices:

• S3FN60D

9.13.1 S3FN60D

On the S3FN60D the watchdog may be running after reset (depends on the content of the smart option bytes at addr. 0xC0). The watchdog keeps counting even if the CPU is in debug mode (e.g. halted). So, please do not use the watchdog when debugging to avoid unexpected behavior of the target application. A special reset strategy has been implemented for this device which disables the watchdog right after a reset has been performed. We recommend to use this reset strategy when debugging a Samsung S3FN60D device.

9.14 ST Microelectronics

J-Link has been tested with the following ST Microelectronics devices:

- STR710FZ1
- STR710FZ2
- STR711FR0
- STR711FR1
- STR711FR2
- STR712FR0
- STR712FR1
- STR712FR2
- STR715FR0
- STR730FZ1
- STR730FZ2
- STR731FV0
- STR731FV1
- STR731FV2
- STR735FZ1
- STR735FZ2
- STR736FV0
- STR736FV1
- STR736FV2
- STR750FV0
- STR750FV1
- STR750FV2
- STR751FR0
- STR751FR1
- STR751FR2
- STR752FR0
- STR752FR1
- STR752FR2
- STR755FR0
- STR755FR1
- STR755FR2
- STR755FV0
- STR755FV1
- STR755FV2
- STR911FM32
- STR911FM44
- STR911FW32
- STR911FW44
- STR912FM32
- STR912FM44
- STR912FW32
- STR912FW44
- STM32F101C6
- STM32F101C8
- STM32F101R6
- STM32F101R8
- STM32F101RB
- STM32F101V8
- STM32F101VB
- STM32F103C6
- STM32F103C8
- STM32F103R6
- STM32F103R8
- STM32F103RB
- STM32F103V8
- STM32F103VB

9.14.1 STR91x

9.14.1.1 JTAG settings

These device are ARM966E-S based. We recommend to use adaptive clocking for these devices.

9.14.1.2 Unlocking

The devices have 3 TAP controllers built-in. When starting J-Link.exe, it reports 3 JTAG devices. A special tool, J-Link STR9 Commander (JLinkSTR91x.exe) is available to directly access the flash controller of the device. This tool can be used to erase the flash of the controller even if a program is in flash which causes the ARM core to stall. For more information about the J-Link STR9 Commander, please refer to *J-Link STR91x Commander (Command line tool)* on page 67.

When starting the STR91x commander, a command sequence will be performed which brings MCU into Turbo Mode.

"While enabling the Turbo Mode, a dedicated test mode signal is set and controls the GPIOs in output. The IOs are maintained in this state until a next JTAG instruction is send." (ST Microelectronics)

Enabling Turbo Mode is necessary to guarantee proper function of all commands in the STR91x Commander.

9.14.1.3 Switching the boot bank

The bootbank of the STR91x devices can be switched by using the J-Link STR9 Commander which is part of the J-Link software and documentation package. For more information about the J-Link STR9 Commander, please refer to *J-Link STR91x Commander (Command line tool)* on page 67.

9.14.2 STM32F10x

These device are Cortex-M3 based. All devices of this family are supported by J-Link.

9.14.2.1 Option byte programming

J-Flash supports programming of the option bytes for STM32 devices. In order to program the option bytes simply choose the appropriate Device, which allows option byte programming, in the CPU settings tab (e.g. **STM32F103ZE** (allow opt. **bytes**)). J-Flash will allow programming a virtual 16-byte sector at address 0x06000000 which represents the 8 option bytes and their complements. You do not have to care about the option bytes' complements since they are computated automatically. The following table describes the structure of the option bytes sector

Address	[31:24]	[23:16]	[15:8]	[7:0]
0x06000000	complement	Option byte 1	complement	Option byte 0
0x06000004	complement	Option byte 3	complement	Option byte 2
0x06000008	complement	Option byte 5	complement	Option byte 4
0x0600000C	complement	Option byte 7	complement	Option byte 6
0x0600000C		Option byte 7	•	

Table 9.1: Option bytes sector description

Note: Writing a value of 0xFF inside option byte 0 will read-protect the STM32. In order to keep the device unprotected you have to write the key value 0xA5 into option byte 0.

Note: The address 0x06000000 is a virtual address only. The option bytes are originally located at address 0x1FFFF800. The remap from 0x06000000 to 0x1FFFF800 is done automatically by J-Flash.

Example

To program the option bytes 2 and 3 with the values 0xAA and 0xBB but leave the device unprotected your option byte sector (at addr 0x06000000) should look like as follows:

Address	[31:24]	[23:16]	[15:8]	[7:0]
0x06000000	0x00	0xFF	0x5A	0xA5
0x06000004	0x44	0xBB	0x55	0xAA
0x06000008	0x00	0xFF	0x00	0xFF
0x0600000C	0x00	0xFF	0x00	0xFF

 Table 9.2: Option bytes programming example

For a detailed description of each option byte, please refer to ST programming manual PM0042, section "Option byte description".

9.14.2.2 Securing/unsecuring the device

The user area internal flash of the STM32 devices can be protected (secured) against read by untrusted code. The J-Flash software allows securing a STM32F10x device. For more information about J-Flash, please refer to *UM08003*, *J-Flash User Guide*. In order to unsecure a read-protected STM32F10x device, SEGGER offers two software components:

- J-Flash
- J-Link STM32 Commander (command line utility)

For more information about J-Flash, please refer to UM08003, J-Flash User Guide. For more information about the J-Link STM32 Commander, please refer to J-Link STM32 Commander (Command line tool) on page 69.

Note: Unsecuring a secured device will cause a mass-erase of the internal flash memory.

9.14.2.3 Hardware watchdog

The hardware watchdog of a STM32F10x device can be enabled by programming the option bytes. If the hardware watchdog is enabled the device is reset periodically if the watchdog timer is not refreshed and reaches 0. If the hardware watchdog is enabled by an application which is located in flash and which does not refresh the watchdog timer, the device can not be debugged anymore.

Disabling the hardware watchdog

In order to disable the hardware watchdog the option bytes have to be re-programmed. SEGGER offers a free command line tool which reprograms the option bytes in order to disable the hardware watchdog. For more information about the STM32 commander, please refer to *J-Link STM32 Commander (Command line tool)* on page 69.

9.14.2.4 Software watchdog

If you enable the software watchdog (independed watchdog / window watchdog) in your target application and still want to debug it, you should make sure that the watchdog does not keep running while the CPU is in debug mode (e.g. halted by J-Link). This can be configured via the DBGMCU_CR register of the STM32F10x devices. To configure the watchdog timers to stop while the CPU is in debug mode, bits 8 and 9 of the DBGMCU_CR have to be set:

```
*((volatile int *)(0x40040520)) = (1 << 8) | (1 << 9);
```

9.15 Texas Instruments

J-Link has been tested with the following Texas Instruments devices:

- AM3352
- AM3354
- AM3356
- AM3357
- AM3358
- AM3359
- TMS470R1A64
- TMS470R1A128
- TMS470R1A256
- TMS470R1A288
- TMS470R1A384
- TMS470R1B512
- TMS470R1B768
- TMS470R1B1M
- TMS470R1VF288
- TMS470R1VF688
- TMS470R1VF689

9.15.1 AM335x

The AM335x series CPUs need some special handling in various cases so the J-Link DLL needs to know that it shall connect to a AM335x device.

9.15.1.1 Selecting the device in the IDE

When using J-Link in an IDE, this is done by selecting the correct device in the IDE. The device name will then be passed to the J-Link DLL when connecting to the target device.

Options for node "Gene	ric_AM335x" 🛛 🗙
Category: General Options	
C/C++ Compiler Assembler Output Converter Custom Build Build Actions Linker Debugger Simulator	Target Output Library Configuration Library Options MISRA-C:200 Processor variant C Cgre Cortex-A8
Angel GDB Server IAR ROM-monitor	© Device TexasInstruments am3359 □
J-Link/J-Trace TI Stellaris Macraigor PE micro RDI JTAGjet ST-LINK Third-Party Driver TI XDS100	Endian mode EPU © Little VFPv3 + NEON © Big © BE32 © BE2 © BE2

9.15.1.2 Selecting the device when using GDBServer

When using the J-Link GDBServer, the device needs to be known BEFORE GDB connects to the GDBServer since GDBServer connects to the device as soon as it is started. So selecting the device via monitor command is too late. In order to select the device before GDBServer connects to it, simply start it with the following command line:

```
-device=<DeviceName>
Example: JLinkGDBServer -device=AM3359
```

9.15.1.3 Selecting the device when using J-Link Commander

For J-Link Commander, type:

device <DeviceName>

Then J-Link Commander will perform a reconnect with the device name selected before.

9.15.1.4 Known values for <DeviceName>

Currently, the following device names for AM335x are known by the DLL:

- AM3352
- AM3354
- AM3356
- AM3357
- AM3358
- AM3359

9.16 Toshiba

J-Link has been tested with the following Toshiba devices:

- TMPM321F10FG
- TMPM322F10FG •
- TMPM323F10FG •
- TMPM324F10FG •
- TMPM330FDFG
- TMPM330FWFG
- TMPM330FYFG •
- TMPM332FWUG
- TMPM333FDFG •
- TMPM333FWFG .
- TMPM333FYFG •
- TMPM341FDXBG • •
- TMPM341FYXBG TMPM360F20FG •
- TMPM361F10FG
- TMPM362F10FG •
- TMPM363F10FG •
- •
- TMPM364F10FG
- TMPM366FDFG
- TMPM366FWFG .
- TMPM366FYFG .
- TMPM370FYDFG •
- TMPM370FYFG •
- TMPM372FWUG •
- TMPM373FWDUG
- TMPM374FWUG •
- TMPM380FWDFG • TMPM380FWFG •
- TMPM380FYDFG •
- TMPM380FYFG TMPM382FSFG •
- TMPM382FWFG •
- TMPM395FWXBG •

Currently, there are no specifics for these devices.

Chapter 10

Target interfaces and adapters

This chapter gives an overview about J-Link / J-Trace specific hardware details, such as the pinouts and available adapters.

10.1 20-pin JTAG/SWD connector

10.1.1 Pinout for JTAG

J-Link and J-Trace have a JTAG connector compatible to ARM's Multi-ICE. The JTAG connector is a 20 way Insulation Displacement Connector (IDC) keyed box header (2.54mm male) that mates with IDC sockets mounted on a ribbon cable.

*On later J-Link products like the J-Link Ultra, these pins are reserved for firmware extension purposes. They can be left open or connected to GND in normal debug environment. They are not essential for JTAG/SWD in general.

VTref	1	•	• 2	NC
nTRST	3	•	• 4	GND
TDI	5	•	• 6	GND
TMS	7	•	• 8	GND
тск	9	•	• 10	GND
RTCK	11	•	• 12	GND
TDO	13	•	• 14	GND*
RESET	15	•	• 16	GND*
DBGRQ	17	•	• 18	GND*
5V-Supply	19	•	• 20	GND*
	1			

PIN	SIGNAL	ТҮРЕ	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	Not con- nected	NC	This pin is not connected in J-Link.
3	nTRST	Output	JTAG Reset. Output from J-Link to the Reset signal of the target JTAG port. Typically connected to nTRST of the target CPU. This pin is normally pulled HIGH on the target to avoid unintentional resets when there is no connection.
5	TDI	Output	JTAG data input of target CPU It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU.
7	TMS	Output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU.
9	тск	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.
11	RTCK	Input	Return test clock signal from the target. Some targets must synchronize the JTAG inputs to internal clocks. To assist in meeting this requirement, you can use a returned, and retimed, TCK to dynamically control the TCK rate. J-Link supports adaptive clocking, which waits for TCK changes to be echoed correctly before making further changes. Con- nect to RTCK if available, otherwise to GND.
13	TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU.

The following table lists the J-Link / J-Trace JTAG pinout.

Table 10.1: J-Link / J-Trace pinout

PIN	SIGNAL	TYPE	Description
15	RESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
17	DBGRQ	NC	This pin is not connected in J-Link. It is reserved for com- patibility with other equipment to be used as a debug request signal to the target system. Typically connected to DBGRQ if available, otherwise left open.
19	5V-Sup- ply	Output	This pin can be used to supply power to the target hard- ware. Older J-Links may not be able to supply power on this pin. For more information about how to enable/disable the power supply, please refer to <i>Target power supply</i> on page 226.

Table 10.1: J-Link / J-Trace pinout

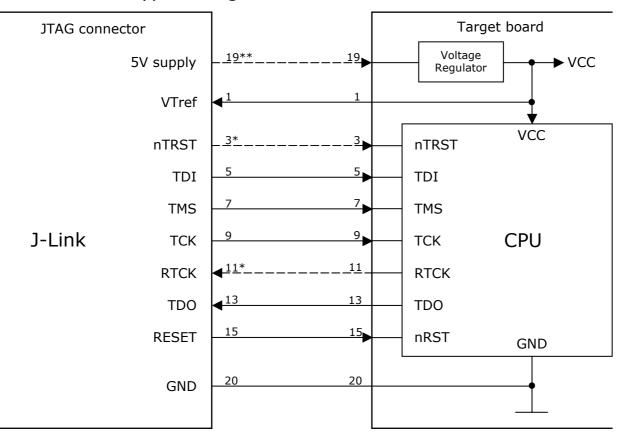
Pins 4, 6, 8, 10, 12, 14, 16, 18, 20 are GND pins connected to GND in J-Link. They should also be connected to GND in the target system.

10.1.1.1 Target board design

We strongly advise following the recommendations given by the chip manufacturer. These recommendations are normally in line with the recommendations given in the table *Pinout for JTAG* on page 224. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

You may take any female header following the specifications of DIN 41651. For example:

Harting	part-no. 09185206803
Molex	part-no. 90635-1202
Tyco Electronics	part-no. 2-215882-0



Typical target connection for JTAG

* NTRST and RTCK may not be available on some CPUs.

** Optional to supply the target board from J-Link.

10.1.1.2 Pull-up/pull-down resistors

Unless otherwise specified by developer's manual, pull-ups/pull-downs are recommended to 100 kOhms.

10.1.1.3 Target power supply

Pin 19 of the connector can be used to supply power to the target hardware. Supply voltage is 5V, max. current is 300mA. The output current is monitored and protected against overload and short-circuit. Power can be controlled via the J-Link commander. The following commands are available to control power:

Explanation
Switch target power on
Switch target power off
Set target power supply default to "on"
Set target power supply default to "off"

Table 10.2: Command List

10.1.2 Pinout for SWD

The J-Link and J-Trace JTAG connector is also compatible to ARM's Serial Wire Debug (SWD).

*On later J-Link products like the J-Link Ultra, these pins are reserved for firmware extension purposes. They can be left open or connected to GND in normal debug environment. They are not essential for JTAG/SWD in general.

	-		1
VTref	1 •	• 2	NC
Not used	3 •	• 4	GND
Not used	5 •	• 6	GND
SWDIO	7•	• 8	GND
SWCLK	9•	• 10	GND
Not used	_11 •	• 12	GND
swo	13 •	• 14	GND*
RESET	15 •	• 16	GND*
Not used	17 •	• 18	GND*
5V-Supply	19 🔹	• 20	GND*

The following table lists the J-Link / J-Trace SWD 5v pinout.

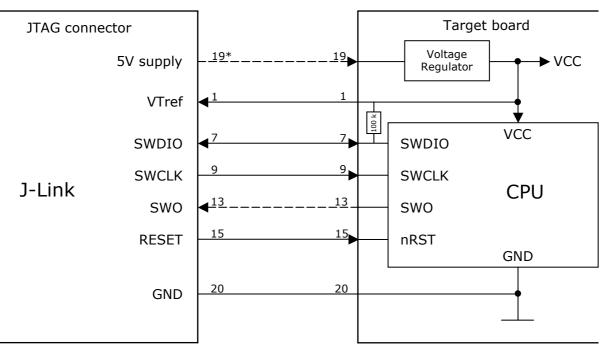
PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	Not con- nected	NC	This pin is not connected in J-Link.
3	Not Used	NC	This pin is not used by J-Link. If the device may also be accessed via JTAG, this pin may be connected to nTRST, otherwise leave open.
5	Not used	NC	This pin is not used by J-Link. If the device may also be accessed via JTAG, this pin may be connected to TDI, otherwise leave open.
7	SWDIO	I/O	Single bi-directional data pin. A pull-up resistor is required. ARM recommends 100 kOhms.
9	SWCLK	Output	Clock signal to target CPU. It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TCK of target CPU.
11	Not used	NC	This pin is not used by J-Link when operating in SWD mode. If the device may also be accessed via JTAG, this pin may be connected to RTCK, otherwise leave open.
13	SWO	Output	Serial Wire Output trace port. (Optional, not required for SWD communication.)
15	RESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
17	Not used	NC	This pin is not connected in J-Link.
19	5V-Sup- ply	Output	This pin can be used to supply power to the target hard- ware. Older J-Links may not be able to supply power on this pin. For more information about how to enable/disable the power supply, please refer to <i>Target power supply</i> on page 228.

Table 10.3: J-Link / J-Trace SWD pinout

Pins 4, 6, 8, 10, 12, 14, 16, 18, 20 are GND pins connected to GND in J-Link. They should also be connected to GND in the target system.

10.1.2.1 Target board design

We strongly advise following the recommendations given by the chip manufacturer. These recommendations are normally in line with the recommendations given in the table *Pinout for SWD* on page 227. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.



Typical target connection for SWD

* Optional to supply the target board from J-Link.

10.1.2.2 Pull-up/pull-down resistors

A pull-up resistor is required on SWDIO on the target board. ARM recommends 100 kOhms.

In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

10.1.2.3 Target power supply

Pin 19 of the connector can be used to supply power to the target hardware. Supply voltage is 5V, max. current is 300mA. The output current is monitored and protected against overload and short-circuit.

Power can be controlled via the J-Link commander. The following commands are available to control power:

Command	Explanation
power on	Switch target power on
power off	Switch target power off
power on perm	Set target power supply default to "on"
power off perm	Set target power supply default to "off"

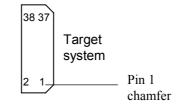
Table 10.4: Command List

10.2 38-pin Mictor JTAG and Trace connector

J-Trace provides a JTAG+Trace connector. This connector is a 38-pin mictor plug. It connects to the target via a 1-1 cable.

The connector on the target board should be "TYCO type 5767054-1" or a compatible receptacle. J-Trace supports 4, 8, and 16-bit data port widths with the high density target connector described below.

Target board trace connector

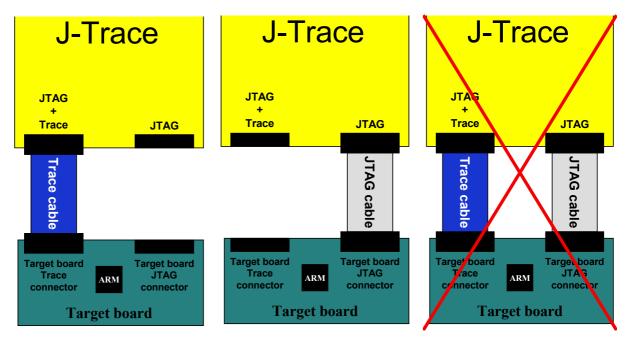


J-Trace can capture the state of signals PIPESTAT[2:0], TRACESYNC and TRACEPKT[n:0] at each rising edge of each TRACECLK or on each alternate rising or falling edge.

10.2.1 Connecting the target board

J-Trace connects to the target board via a 38-pin trace cable. This cable has a receptacle on the one side, and a plug on the other side. Alternatively J-Trace can be connected with a 20-pin JTAG cable.

Warning: Never connect trace cable and JTAG cable at the same time because this may harm your J-Trace and/or your target.



10.2.2 Pinout

The following table lists the JTAG+Trace connector pinout. It is compatible to the "Trace Port Physical Interface" described in [ETM], 8.2.2 "Single target connector pinout".

PIN	SIGNAL	Description
1	NC	No connected.
2	NC	No connected.
3	NC	No connected.
4	NC	No connected.
5	GND	Signal ground.
6	TRACECLK	Clocks trace data on rising edge or both edges.
7	DBGRQ	Debug request.
8	DBGACK	Debug acknowledge from the test chip, high when in debug state.
9	RESET	Open-collector output from the run control to the target system reset.
10	EXTTRIG	Optional external trigger signal to the Embedded trace Macrocell (ETM). Not used. Leave open on target system.
11	TDO	Test data output from target JTAG port.
12	VTRef	Signal level reference. It is normally fed from Vdd of the target board and must not have a series resistor.
13	RTCK	Return test clock from the target JTAG port.
14	VSupply	Supply voltage. It is normally fed from Vdd of the target board and must not have a series resistor.
15	ТСК	Test clock to the run control unit from the JTAG port.
16	Trace signal 12	Trace signal. For more information, please refer to <i>Assignment of trace information pins between ETM architecture versions</i> on page 232.
17	TMS	Test mode select from run control to the JTAG port.
18	Trace signal 11	Trace signal. For more information, please refer to <i>Assignment of trace information pins between ETM architecture versions</i> on page 232.
19	TDI	Test data input from run control to the JTAG port.
20	Trace signal 10	Trace signal. For more information, please refer to <i>Assignment of trace information pins between ETM architecture versions</i> on page 232.
21	nTRST	Active-low JTAG reset
Tabla	10 5. ITAG+Trace connec	to a set of the sector

Table 10.5: JTAG+Trace connector pinout

PIN	SIGNAL	Description
22	Trace signal 9	
23	Trace signal 20	
24	Trace signal 8	
25	Trace signal 19	
26	Trace signal 7	
27	Trace signal 18	
28	Trace signal 6	
29	Trace signal 17	Trace signals. For more information, please refer to
30	Trace signal 5	Assignment of trace information pins between ETM archi-
31	Trace signal 16	tecture versions on page 232.
32	Trace signal 4	
33	Trace signal 15	
34	Trace signal 3	
35	Trace signal 14	
36	Trace signal 2	
37	Trace signal 13	
38	Trace signal 1	

Table 10.5: JTAG+Trace connector pinout

10.2.3 Assignment of trace information pins between ETM architecture versions

The following table show different names for the trace signals depending on the ETM architecture version.

Trace signal	ETMv1	ETMv2	ETMv3
Trace signal 1	PIPESTAT[0]	PIPESTAT[0]	TRACEDATA[0]
Trace signal 2	PIPESTAT[1]	PIPESTAT[1]	TRACECTL
Trace signal 3	PIPESTAT[2]	PIPESTAT[2]	Logic 1
Trace signal 4	TRACESYNC	PIPESTAT[3]	Logic 0
Trace signal 5	TRACEPKT[0]	TRACEPKT[0]	Logic 0
Trace signal 6	TRACEPKT[1]	TRACEPKT[1]	TRACEDATA[1]
Trace signal 7	TRACEPKT[2]	TRACEPKT[2]	TRACEDATA[2]
Trace signal 8	TRACEPKT[3]	TRACEPKT[3]	TRACEDATA[3]
Trace signal 9	TRACEPKT[4]	TRACEPKT[4]	TRACEDATA[4]
Trace signal 10	TRACEPKT[5]	TRACEPKT[5]	TRACEDATA[5]
Trace signal 11	TRACEPKT[6]	TRACEPKT[6]	TRACEDATA[6]
Trace signal 12	TRACEPKT[7]	TRACEPKT[7]	TRACEDATA[7]
Trace signal 13	TRACEPKT[8]	TRACEPKT[8]	TRACEDATA[8]
Trace signal 14	TRACEPKT[9]	TRACEPKT[9]	TRACEDATA[9]
Trace signal 15	TRACEPKT[10]	TRACEPKT[10]	TRACEDATA[10]
Trace signal 16	TRACEPKT[11]	TRACEPKT[11]	TRACEDATA[11]
Trace signal 17	TRACEPKT[12]	TRACEPKT[12]	TRACEDATA[12]
Trace signal 18	TRACEPKT[13]	TRACEPKT[13]	TRACEDATA[13]
Trace signal 19	TRACEPKT[14]	TRACEPKT[14]	TRACEDATA[14]
Trace signal 20	TRACEPKT[15]	TRACEPKT[15]	TRACEDATA[15]

Table 10.6: Assignment of trace information pins between ETM architecture versions

10.2.4 Trace signals

Data transfer is synchronized by TRACECLK.

10.2.4.1 Signal levels

The maximum capacitance presented by J-Trace at the trace port connector,

including the connector and interfacing logic, is less than 6pF. The trace port lines have a matched impedance of 50.

The J-Trace unit will operate with a target board that has a supply voltage range of 3.0V-3.6V.

10.2.4.2 Clock frequency

For capturing trace port signals synchronous to TRACECLK, J-Trace supports

a TRACECLK frequency of up to 200MHz. The following table shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACE-CLK.

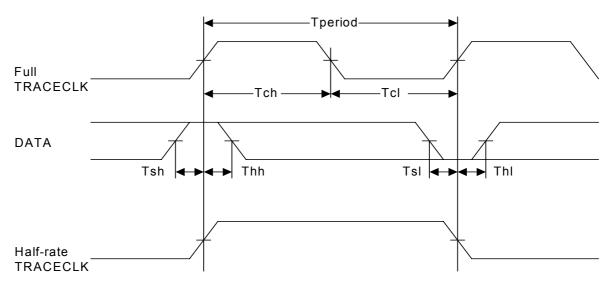
Parameter	Min.	Max.	Explanation
Tperiod	5ns	1000ns	Clock period
Fmax	1MHz	200MHz	Maximum trace frequency
Tch	2.5ns	-	High pulse width
Tcl	2.5ns	-	Low pulse width

Table 10.7: Clock frequency

Parameter	Min.	Max.	Explanation
Tsh	2.5ns	-	Data setup high
Thh	1.5ns	-	Data hold high
Tsl	2.5ns	-	Data setup low
ThI	1.5ns	-	Data hold low

Table 10.7: Clock frequency

The diagram below shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACECLK.



Note: J-Trace supports half-rate clocking mode. Data is output on each edge of the TRACECLK signal and TRACECLK (max) ≤ 100 MHz. For half-rate clocking, the setup and hold times at the JTAG+Trace connector must be observed.

10.3 19-pin JTAG/SWD and Trace connector

J-Trace provides a JTAG/SWD+Trace connector. This connector is a 19-pin connector. It connects to the target via an 1-1 cable.

	_				
VTref	1	•	•	2	SWDIO/TMS
GND	3	•	•	4	SWCLK/TCK
GND	5	•	•	6	SWO/TDO
	7		•	8	TDI
NC	9	•	•	10	nRESET
5V-Supply	_11	•	•	12	TRACECLK
5V-Supply	13	•	•	14	TRACEDATA[0]
GND	15	٠	•	16	TRACEDATA[1]
GND	17	٠	•	18	TRACEDATA[2]
GND	19	•	•	20	TRACEDATA[3]

The following table lists the J-Link / J-Trace SWD pinout.

PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	SWDIO/ TMS	I/O / output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU.
4	SWCLK/TCK	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.
6	SWO/TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU. When using SWD, this pin is used as Serial Wire Output trace port. (Optional, not required for SWD communication)
			This pin (normally pin 7) is not existent on the 19-pin JTAG/SWD and Trace connector.
8	TDI	Output	JTAG data input of target CPU It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU. For CPUs which do not provide TDI (SWD-only devices), this pin is not used. J-Link will ignore the signal on this pin when using SWD.
9	NC	NC	Not connected inside J-Link. Leave open on target hard- ware.
10	nRESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
11	5V-Supply	Output	This pin can be used to supply power to the target hard- ware. For more information about how to enable/disable the power supply, please refer to <i>Target power supply</i> on page 235.
12	TRACECLK	Input	Input trace clock. Trace clock = $1/2$ CPU clock.
13	5V-Supply	Output	This pin can be used to supply power to the target hard- ware. For more information about how to enable/disable the power supply, please refer to <i>Target power supply</i> on page 235.
14	TRACE- DATA[0]	Input	Input Trace data pin 0.

Table 10.8: 19-pin JTAG/SWD and Trace pinout

PIN	SIGNAL	TYPE	Description
16	TRACE- DATA[1]	Input	Input Trace data pin 0.
18	TRACE- DATA[2]	Input	Input Trace data pin 0.
20	TRACE- DATA[3]	Input	Input Trace data pin 0.

Table 10.8: 19-pin JTAG/SWD and Trace pinout

Pins 3, 5, 15, 17, 19 are GND pins connected to GND in J-Trace CM3. They should also be connected to GND in the target system.

10.3.1 Target power supply

Pins 11 and 13 of the connector can be used to supply power to the target hardware. Supply voltage is 5V, max. current is 300mA. The output current is monitored and protected against overload and short-circuit.

Power can be controlled via the J-Link commander. The following commands are available to control power:

target power on
a target power on
target power off
rget power supply default to "on"
rget power supply default to "off"
1

Table 10.9: Command List

10.4 9-pin JTAG/SWD connector

Some target boards only provide a 9-pin JTAG/ SWD connector for Cortex-M. For these devices SEGGER provides a 20-pin -> 9-pin Cortex-M adapter.

		7
VTref		SWDIO / TMS
GND 🗌	3 • • 4	SWCLK / TCK
GND	5 • • 6	SWO / TDO
	7 • 8	
NC	9 • • 10	nRESET

The following table lists the output of the 9-pin Cortex-M connector.

PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	SWDIO/ TMS	I/O / output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU. When using SWD, this pin is used as Serial Wire Output trace port. (Optional, not required for SWD communication)
4	SWCLK/TCK	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.
6	SWO/TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU.
			This pin (normally pin 7) is not existent on the 19-pin JTAG/SWD and Trace connector.
8	TDI	Output	JTAG data input of target CPU It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU. For CPUs which do not provide TDI (SWD-only devices), this pin is not used. J-Link will ignore the signal on this pin when using SWD.
9	NC	NC	Not connected inside J-Link. Leave open on target hard- ware.

Table 10.10: 9-pin JTAG/SWD pinout

Pins 3 and 5 are GND pins connected to GND on the Cortex-M adapter. They should also be connected to GND in the target system.

10.5 Adapters

There are various adapters available for J-Link as for example the JTAG isolator, the J-Link RX adapter or the J-Link Cortex-M adapter.

For more information about the different adapters, please refer to *http://www.segger.com/jlink-adapters.html*.

Chapter 11 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 the related decode mechanism are greatly simplified compared with microprogrammed *Complex Instruction Set Computer* (CISC).

11.1 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.

11.1.1 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.
nTRST	Input (optional)	The optional test reset (nTRST) input provides for asyn- chronous initialization of the TAP controller.

Table 11.1: Test access port

11.1.2 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.

11.1.3 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.

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.

Reset tm s= DR-Scan Idle IR-Scan tms=0 tms=0 tms Capture-DR Capture-IR tms=0 tms=0 Shift-DR Shift-IR tms=1 tms=0 tms=1 tms=0 Exit1-DR Fxit1-IR tmetms= tms=0 tms=0 Pause-DR Pause-IR tms=0 tms=0 tms=1 tms=1 Exit2-DR Exit2-IR tms=1 tms=1 Update-DR Update-IR tms=1 tms=0 tms=1 tms=0

TAP controller state diagram

11.1.4.1 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.

11.2 Embedded Trace Macrocell (ETM)

Embedded Trace Macrocell (ETM) provides comprehensive debug and trace facilities for ARM processors. ETM allows to capture information on the processor's state without affecting the processor's performance. The trace information is exported immediately after it has been captured, through a special trace port.

Microcontrollers that include an ETM allow detailed program execution to be recorded and saved in real time. This information can be used to analyze program flow and execution time, perform profiling and locate software bugs that are otherwise very hard to locate. A typical situation in which code trace is extremely valuable, is to find out how and why a "program crash" occurred in case of a runaway program count.

A debugger provides the user interface to J-Trace and the stored trace data. The debugger enables all the ETM facilities and displays the trace information that has been captured. J-Trace is seamlessly integrated into the IAR Embedded Workbench® IDE. The advanced trace debugging features can be used with the IAR C-SPY debugger.

11.2.1 Trigger condition

The ETM can be configured in software to store trace information only after a specific sequence of conditions. When the trigger condition occurs the trace capture stops after a programmable period.

11.2.2 Code tracing and data tracing

Code trace

Code tracing means that the processor outputs trace data which contain information about the instructions that have been executed at last.

Data trace

Data tracing means that the processor outputs trace data about memory accesses (read / write access to which address and which data has been read / stored). In general, J-Trace supports data tracing, but it depends on the debugger if this option is available or not. Note that when using data trace, the amount of trace data to be captured rises enormously.

11.2.3 J-Trace integration example - IAR Embedded Workbench for ARM

In the following a sample integration of J-Trace and the trace functionality on the debugger side is shown. The sample is based on IAR's Embedded Workbench for ARM integration of J-Trace.

11.2.3.1 Code coverage - Disassembly tracing

	J @ X h C		•	1 > 7 2 🛙	> +> +2 (b) 05 🥨 98 🕭 🎂	₫₽				
	26333	X				Nasaaabba				
c glcd_l.c	stm32f10x_nvic.c				•	Disassembly Ge to	Ţ	Memory	• 🗈	
194 d	fdef DEBUG debug<>; ndif				1					
						??DrawTable 0800BFA0 0800BFA2 ??DrawTable	B01A ADD BD70 POP	SP (R	, SP, #0×68 4,R5,R6,PC}	
197	ENTR_CRT_SECTION	system				??DrawTable 0800BFA4 0800BFA6	_0: DDC0 BLE 0800 LSR	0×	8008F28 , RO, #0×0	
						void main(v	0800 LSR 01d)	S RO	, RO, #OXO	
201 202 #if	// NUIC init <mark>fndef</mark> EMB_FLAS	SH tor Table base 1 Table(NUIC_UectI AB_FLASH */ tor Table base 1 Table(NUIC_UectI Table(NUIC_UectI				main:				
203 204 M	/* Set the Vect NVIC_SetVector	tor Table base 1 Table(NVIC_VectI	location at 0x20 `ab_RAM, 0x0);	3000000 ×/		.text_14: 0800BFA8	0510 000		4.103	
205 #e] 206	lse /* UECT_TH /* Set the Vect	AB_FLASH */ tor Table base]	location at 0×08	3000000 ×/		0800BFAA	B510 PUS B088 SUE	H (R SP	4,LR} , SP, #0×20	
207 N 208 tter	NUIC_SetVector ndif	Table(NUIC_VectI	ab_FLASH, 0×0>;	;		0800BFAC	F001F8A8 BL	de	bug	
207 1	NOIC_FFIOFIC9G	Poupconrig(Noic_	FriorityGroup	1/,			F002F87E BL	En	trCritSection	
211 212 213 214 215 215	// SysTick end	of count event	each 0.1s with	input clock e	qual to 9MHz (HCLK/8, defau]	0800BFB4	F7FFFF62 BL	C VectTab EL	k_Init	
213	// Enable SysT:	ick interrupt				0800BFB8 0800BFBA	ectorTable(NVI 2100 MBV F05F6000 MDV F001FC65 BL	S R1 S R0	ASH. 0×0); , #0×0 , #0×800000 IC_SetVectorTable	
215 8	SysTick_Counter	rCmd(SysTick_Cou	unter_Enable);			0800BFBE NVIC Price	F001FC65 BL ritvGroupConfi	NV aíNVIC Prior	IC_SetVectorTable	
						000000560	E44ET040 MOL	R0 NV	ityGroup_4): , #0x300 IC_PriorityGroupConfi	a
217 218 219	// GP10 enable RCC_APB2Periphl	clock and relea ResetCnd(RCC_A	ise Reset IPB2Periph_GPIO	1		SysTick_S 0800BFCA	F001FC41 BL etReload(90000 4876 LDR	0); R0	, [PC, #0×1D8]	-
220 221 222	RCC_APB2Periph	t init clock and relea ResetCmd< RCC_A I RCC_A ClockCmd< RCC_A	IFB2Periph_GPI00 IPB2Periph_GPI06	G, DISABLE>;		OBOOBFCC SysTick_J	F001FAE2 BL TConfig(ENABLE	⇒y ⊇∙	sTick_SetReload	
666			PB2Periph_GPI00			0800BFD0 0800BFD2	FOOIFAE2 BL TConfig(ENABLE 2001 MOV FOOIFB1A BL ounterCmd(SysT	S RO Sy	, #0x1 sTick_ITConfig Enable): , #0x1 sTick_CounterCmd rinh GTOA	
223 224 225 226 227	GPIO_InitStruct							<u>ick_Counter</u> S RO	Enable): ,#0×1	
226 0	GPIO_InitStruct	ture.GPI0_Pin = ture.GPI0_Mode = ture.GPI0_Speed DRT, &GPI0_InitS	GPIO_Mode_IN_I	FLOATING;		0800BFD8 RCC_APB2F	F001FAEB BL eriphResetCmd(Sy RCC_APB2Pe	STick_CounterCmd riph_GPIOA riph_GPIOG, DISABLE):	
228 0						0800BFDC	2100 MOV	S R1	. #0×0	
228 229 230 231	GPIO_InitStruct	ture.GPI0_Pin = ture.GPI0_Mode = ture.GPI0_Speed ORT, &GPI0_InitS	B2_MASK; GPIO Mode IN 1	PLOAT INC-		0800BFDE 0800BFE2	F44F7082 MDV F7FFFA80 BL eriphClockCmd(R0 RC	, #0x104 C_APB2PeriphResetCmd riph_GPIOA	
232 233	GPIO_InitStruct	ture.GPI0_Speed	= GPIO_Speed_56	MHz;						
			cructure),			0800BFE6	2101 MOV F44F7082 MOV F7FFFA20 BL Structure.GPIC F44F7080 MOV	S R1	riph_GPIOG, ENABLE); , #0x1 , #0x104 C_APB2PeriphClockCmd	
	EXT_CRT_SECTION					0800BFEC	F7FFFA20 BL	Pin = B1 M	C_APB2PeriphClockCmd	
237 238 239 F	// AN_TR port a // Enable ADC1	and ADC init and GPIOC clock	ç		oh_GPIOC, DISABLE>; oh_GPIOC, ENABLE>;	0800BFF0	Structure.GPIC F44F7080 MOV F8AD0000 STR Structure.GPIC	RO	ASK: , #0x100 , [SP] 	
239 F 240 F	RCC_APB2Periph RCC_APB2Periph	ResetCnd <rcc_ape ClockCnd<rcc_ape< td=""><td>2Periph_ADC1 2Periph_ADC1 </td><td>RCC_APB2Perip RCC_APB2Perip</td><td><pre>bh_GPIOC, DISABLE>; bh_GPIOC, ENABLE>;</pre></td><td>GPIO_Init</td><td>Structure.GPIC</td><td>_Mode = GPIO</td><td>Mode_IN_FLOATING:</td><td></td></rcc_ape<></rcc_ape 	2Periph_ADC1 2Periph_ADC1	RCC_APB2Perip RCC_APB2Perip	<pre>bh_GPIOC, DISABLE>; bh_GPIOC, ENABLE>;</pre>	GPIO_Init	Structure.GPIC	_Mode = GPIO	Mode_IN_FLOATING:	
					}					
	🔍 🏷 🖬 🛤 📗									
ndex	Frame	Address	Opcode E004	Trace	22NUTC SetVertorTable ->			Comment		
ndex		Address 0×0800D89E	Opcode E004	B assert	<pre>??NVIC_SetVectorTable_2 _param(IS_NVIC_OFFSET(Offset));</pre>			Comment		
ndex 03064	Frame			B assert	_param(IS_NVIC_OFFSET(Offset)); etVectorTable_2:			Comment		
ndex 03064 03065 03066	Frame 003382 003383 003384	0×0800D89E 0×0800D8AA 0×0800D8AC	E004 4807 4285	B assert ??NVIC_S LDR CMP	<pre>_param(IS_NVIC_OFFSET(Offset)); etVectorTable_2: R0, [PC, #0x1C] R5, R0</pre>			Comment		
ndex 03064 03065 03066	Frame 003382 003383	0×0800D89E 0×0800D8AA	E004 4807	B assert ??NVIC_S LDR CMP BCC SCB->V	_param(IS_NVIC_OFFSET(Offset)); tetVectorTable_2: R0, [PC, #0x1C] R5, R0 ?NVIC_SetVectorTable_4 TOR = NVIC_VectTab (Offset & (1)	u32)0×1FFFFF80);		Comment		
ndex 03064 03065 03066 03067	Frame 003382 003383 003384 003384	0×0800D89E 0×0800D8AA 0×0800D8AC 0×0800D8AE	E004 4807 4285 D304	B assert ??NVIC_S LDR CMP BCC SCB->V ??NVIC_S	<pre>_param(IS_NVIC_OFFSET(Offset)); ietvectorTable_2: R0, [PC, #0x1C] R5, R0 ??NVIC_SetvectorTable_4 rOR = NVIC_VectTable_4 (ietvectorTable_4:</pre>	u32)0x1FFFF80);		Comment		
ndex 03064 03065 03066 03067 03068 03068 03069	Frame 003382 003383 003384 003385 003386 003386	0×0800D89E 0×0800D8AA 0×0800D8AC 0×0800D8AE 0×0800D8BA 0×0800D8BA	E004 4807 4285 D304 4804 4028	B assert ??NVIC_S LDR CMP BCC SCB->V ??NVIC_S LDR ANDS	<pre>param(II_NVIC_OPFSET(Offset)); ietvectorTable_2: Ro, [PC, #0xIC] RS, RO ??NVIC_SetVectorTable_4 ??NVIC_SetVectorTable_4: RO, [PC, #0xI0] RO, RO, RS</pre>	u32)0×1FFFFF80);		Comment		
ndex 03064 03065 03066 03067 03068 03069 03070	Frame 003382 003383 003384 003385 003386 003386 003387 003389	0×0800D89E 0×0800D8AA 0×0800D8AC 0×0800D8AE 0×0800D8BA 0×0800D8BC 0×0800D8BE 0×0800D8C0	E004 4807 4285 D304 4804 4028 4320 4904	B assert ??NVIC_S LDR CMP BCC SCB->V ??NVIC_S LDR	param(15WVIC_OFFSet()); etvectorTable_2: R5, R0, [PC, #0xLC] R5, R0 PTWIC_SetvectorTable_4 PTWIC_SetvectorTable_4 R0, [PC, excl0] R0, R0, excl0] R0, R0, R4 R1, [PC, #0xL0]	u32)0×1FFFFF80);		Comment		
ndex 03064 03065 03066 03067 03068 03069 03070 03071 03072	Frame 003382 003383 003384 003385 003386 003387 003388 003389 003390	0x0800089E 0x080008AA 0x080008AC 0x080008AE 0x0800088A 0x0800088A 0x0800088B 0x0800088E 0x0800088C	4807 4285 D304 4804 4028 4320 4904 6809	B ASSETC ??NVIC_5 LDR CMP BCC SCB->V ??NVIC_5 LDR ANDS ORRS LDR LDR	param(IS_NVIC_OFFSF(OffSet)); RO, [PC, #0x1C] RS, R0 79NVIC_SetvectorTable_4 70NVIC_SetvectorTable_4 RO, [PC, #0x10] RO, RO, RS RO, RO, RS RO, RO, RS RO, RO, R4 R1, [PC, #0x10] R1, [R1]	u32)0×1FFFFF80);		Comment		
ndex 03064 03066 03066 03067 03068 03069 03070 03071 03072 03073	Frame 003382 003383 003384 003385 003385 003386 003386 003389 003389 003391	0×0800089E 0×080008AA 0×080008AC 0×080008AC 0×080008AE 0×080008BE 0×080008BE 0×080008BE 0×080008C2 0×080008C2	E004 4807 4285 0304 4804 4028 4320 4904 6809 6088	B ASSETT ??NVIC_5 LDR CMP BCC SCB->V ??NVIC_5 LDR ANDS ORRS LDR LDR STR }	param(15_WVLC_PFSF(OffSet)); evectorTable L2: R5, R0structure Table L3 R5, R0structure Table L4 R5, R0structure Table L4 R5, R5, R5, R5, R5, R5, R5, R5, R5, R5,	u32)0×1FFFFF80);		Comment		
dex 03064 03065 03066 03067 03068 03069 03070 03071 03072 03072 03073	Frame 003382 003383 003384 003385 003386 003387 003388 003389 003390	0x0800089E 0x080008AA 0x080008AC 0x080008AE 0x0800088A 0x0800088A 0x0800088B 0x0800088E 0x0800088C	4807 4285 D304 4804 4028 4320 4904 6809	B assent ??NVIC_S LDR CMP BCC SCB->V ??NVIC_S LDR ANDS ORRS LDR LDR LDR STR } POP	param(15WVLC_PFSF(QffSet)); etvectorTable_42: R5, R0 77WVLC_SetVectorTable_4 78WVLC_SetVectorTable_4 R0, CFC, #0x101 R0, R0, R3 R0, CFC, #0x101 R1, [PC, #0x101 R1, [R1] R0, [R1, #0x8] (R0,44,85,PC)			Comment		
dex 03065 03066 03066 03067 03068 03069 03070 03071 03072 03073 03074 03075	Frame 003382 003384 003384 003385 003386 003386 003386 003389 003389 003391 003392 003392	0x0800089E 0x080008AA 0x080008AC 0x080008AA 0x080008BA 0x080008BC 0x080008BC 0x080008C0 0x080008C4 0x080008C6 0x080008C6	E004 4807 4285 0304 4804 4028 4320 4304 6809 6088 B031 F44F	B 355000 271V/IC_5 LDR CMP BCC SCB→V 271V/IC_5 LDR ANDS ORRS LDR LDR STR J POP NVIC_P NVIC_9	param(15_WVLC_PFSF(0ffset)); etvectorTable_2; R0, [PC, #0x1C] R5, R0 77WVLC_SetVectorTable_4 77WVLC_SetVectorTable_4 R0, [PC, #0x10] R0, R0, R5 R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R0, [R1, #0x3] (R0, R4, R5, RC) rtortyproupconfig(WVLC_PriorityC R0, #0x300			Comment		
dex 03065 03066 03066 03067 03068 03069 03070 03071 03072 03073 03074 03075	Frame 003362 003363 003384 003385 003386 003386 003386 003389 003389 003391 003391 003392	0×0800089E 0×080008AC 0×080008AC 0×080008AC 0×080008AC 0×080008BC 0×080008BC 0×080008C2 0×080008C2 0×080008C4	E004 4807 4285 D304 4804 4028 4320 4904 6809 6088 BD31	В <u>35500000000000000000000000000000000000</u>	param(15_WVLC_PFSF(Q)ffset)); etevectorTable_2: R0, [PC, #0x12] R5, R0 Tom = TWVLC_VENECOTTable_4 TOM = TWVLC_VENECOTTable_4 R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R1, [R1, #0x8] (R0,R4,R5,PC) (R0,R4,R5,PC)	Group_4);		Comment		
ndex 03064 03065 03066 03067 03068 03069 03070 03071 03072 03073 03074 03075	Frame 003382 003384 003384 003385 003386 003386 003386 003389 003389 003391 003392 003392	0x0800089E 0x080008AA 0x080008AC 0x080008AA 0x080008BA 0x080008BC 0x080008BC 0x080008C0 0x080008C4 0x080008C6 0x080008C6	E004 4807 4285 0304 4804 4028 4320 4304 6809 6088 B031 F44F	B 35 Sent LDR CMP BCCC SCD=>V ??NVIC_S LDR ANDS ORKS LDR STR J POP NVIC_P MOV BL VOID IVIC	param(15_WVC_OFFST(OffSet)); EVEX.config.e2; R0, [PC, #0x12] R5, R0 WVC_SEVECTOTAble_4 TWVC_SEVECTOTAble_4 TWVC_SEVECTOTAble_4 TWVC_SEVECTOTAble_4 TWVC_SEVECTOTAble_4 R0, R0, R4 R0, R0, R4 R1, [PC, #0x10] R1, R1, [PC, #0x10] R1, R1, #0x8] (00, R4, R5, PC) (00, R4, R5, PC) (00, R4, R5, PC) (00, R4, R5, PC) VVC_PTIOTSyGnupConfig(UNIC_PTIOTSyGnupConfig NVC_PTIOTSyGnupConfig(UNIC_PTIOTSyGnupConfig (C_PTIOTSyGnupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTSymupConfig(UNIC_PTIOTS	Group_4);		Comment		
adex 03 064 03 065 03 066 03 067 03 068 03 069 03 070 03 071 03 072 03 073 03 074 03 075 03 075 03 076	Frame 003382 003383 003384 003385 003385 003386 003385 003385 003386 003385 003385 003385 003385 003385 003385 003385 003385 003385 003384 003392 003394 003395	0x0800089E 0x080008AA 0x080008AC 0x080008AE 0x080008AE 0x080008E0 0x080008E0 0x080008E0 0x080008E0 0x080008FC6 0x080008FC6 0x080008FC6	E004 4807 4285 0304 4804 4028 4320 4304 4320 4304 6085 8031 F44F F001 8510	B 355000 2781VIC-5 LDR CMP BCC SCB-5 2781VIC-7 2781VIC-7 NOV BL VOID NUIC-7 NUI	param(11WIC_DFFSF(0ffset)); etvectorTable_2: Ro, [Pc, #0x12] RS, R0 TWIC_SetVectorTable_4 TWIC_SetVectorTable_4 TWIC_SetVectorTable_4 TWIC_SetVectorTable_4 TWIC_SetVectorTable_4 RevectorTable_4	Group_4);		Comment		
adex 03 064 03 065 03 066 03 067 03 068 03 069 03 070 03 071 03 072 03 073 03 074 03 075 03 075 03 076	Frame 003382 003384 003384 003385 003386 003386 003386 003380 003380 003390 003390 003391 003392	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2	E004 4807 4285 0204 4804 4028 4320 4320 4320 6088 8031 F44F F001	B	param(15_WVLC_PFSF(0ffset)); evectorTable L2: R0, [PC, #0x12] R5, R0 R0, R0, R4 R0, R5 R0, R5 R0, R4 R1, [PC, #0x10] R0, R0, R4 R1, [PC, #0x10] R0, R0, R4 R1, [R1, #0x3] (R0,R4,R5,PC) (R0,R4,R5,PC) R0, ForthyGroupConfig NVTC_PFiorthyGroupConfig NVTC_PFiorthyGroupConfig RVT_GFIORTHyGroupConfig R4, R8] R4, R8 R4, R8 R5	Group_4); riorityGroup)		Comment		
ndex 03064 03065 03066 03066 03067 03070 03071 03072 03073 03074 03075 03076 03077 03077 03077	Frame 003362 003363 003384 003385 003386 003387 003389 003389 003391 003392 003394 003395 003396 003397	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008AC 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x0800008C2 0x080008C2 0x080008C2 0x080008C2 0x080008C2 0x080000	E004 4807 4285 0304 4804 4028 4320 4304 4320 4304 6085 8031 F44F F001 8510	B 355000 2781VIC-5 LDR CMP BCC SCB-5 2781VIC-7 2781VIC-7 NOV BL VOID NUIC-7 NUI	param(15_WVLC_PFSF(0ffset)); evectorTable L2: R0, [PC, #0x12] F5, R0 The Structure DirectorTable L4 R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R4, R5 R0, R4, R5 R0, R4, R5 R0, R4, R5 R0, R4, R5 R0, R4, R5 R0, Piortsycroupconfig WVLC_Priortsycroupconfig WVLC_Priortsycroupconfig R0, R4, R5 R0, R0, R4 R0, R0, R0 R0, R0, R0 R0, R0 R0	Group_4); riorityGroup)				
ndex 03064 03065 03066 03066 03067 03070 03072 03073 03074 03075 03075 03075 03075 03077 03077 03077 03077	Frame 003383 003384 003384 003385 003386 003387 003386 003387 003389 003390 003391 003392 003394 003395 003396 003397 003396 003397 003396 003397 003396 003397	0x0800089E 0x080008AA 0x080008AC 0x080008AE 0x080008AE 0x080008E 0x080008E 0x080008E 0x080008C2 0x080008C2 0x080008C4 0x080008C4 0x080008C4 0x080008C4 0x080008C4 0x080008C4	E004 4807 4285 0304 4804 4026 4320 4304 6088 8031 F44F F001 8510 0004 F584	B SSSEPT 27NVIC_S LDR CMP BCC CMP BCC CMP BCC CMP BCC CMP BCC CMP BL VOID NVIC_PT PUSH MOV BL VOID NVIC_PT CMP CMP CMP CMP CMP CMP CMP CMP	param(15_WXC_0FFST(0ffset)); etvectorTable_4: R0, [Pc, #0x1c] R5, R0 YWXC_SetvectorTable_4 YWXC_SetvectorTable_4 R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R1, [Pc, #0x1c] R1, [Pc, #0x1c] R0, [Pl, #0x1c] R0, [Pl, #0x1c] NUC_PFiortSyStempConfig NUC_PFiortSyStempConfig R4, #0x700 R4, #0x700 R0, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R4, #0x700 R1, #0x700	Group_4); riorityGroup)		Comment	d	
ndex 03064 03065 03066 03066 03066 03069 03070 03071 03072 03072 03075 03075 03076 03075 03076	Frame 007382 003383 003384 003385 003386 003386 003387 003386 003387 003386 003387 003389 003390 003392 003394 003395 003396 003396 003396 003396 003396 003396 003396	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008EC 0x080008EC 0x080008EC 0x080008C2 0x080008C4 0x080008C4 0x080008C6 0x08008C6 0x080	E004 4807 4285 0204 4804 4026 4320 4304 4304 4309 6088 8039 6088 8031 8544 F001 8510 0004 F584 F584	В	param(15_WXC_PF3F(0ffset)); etvectorTable_4:	Group_4); riorityGroup)				
ndex 03064 03065 03066 03066 03069 03070 03072 03073 03074 03075 03077 03077 03076 03077 03078 03079 03081 03081 03082	Frame 003382 003383 003384 003384 003385 003386 003386 003387 003389 003389 003389 003390 003391 003392 003394 003395 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397	0x0800089E 0x080008AA 0x080008AC 0x080008AE 0x080008AE 0x080008E 0x080008E 0x080008E 0x080008E 0x080008E 0x08008EC 0x08008EC 0x08008EC 0x08008EC 0x08008EC 0x08008EC 0x08008EC 0x08008EC	E004 4807 4285 0304 4804 4026 4320 4304 6088 8031 F44F F001 8510 0004 F584	B SSSEPT 27NVIC_S LDR CMP BCC CMP BCC CMP BCC CMP BCC CMP BCC CMP BL VOID NVIC_PT PUSH MOV BL VOID NVIC_PT CMP CMP CMP CMP CMP CMP CMP CMP	param(15_WXC_0FFST(0ffset)); etvectorTable_4: R0, [Pc, #0x1c] R5, R0 YWXC_SetvectorTable_4 YWXC_SetvectorTable_4 R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R1, [Pc, #0x1c] R1, [Pc, #0x1c] R0, [Pl, #0x1c] R0, [Pl, #0x1c] NUC_PFiortSyStempConfig NUC_PFiortSyStempConfig R4, #0x700 R4, #0x700 R0, #0x1c] R0, [Pc, #0x1c] R0, [Pc, #0x1c] R4, #0x700 R1, #0x700	Group_4); riorityGroup)		Not execute Not execute	d	
ndex 03064 03065 03066 03066 03069 03070 03071 03072 03074 03075 03075 03075 03076 03075 03076 03075 03076 03077 03078 03078 03081 03081 03082 03084 03085	Frame 003382 003383 003384 003385 003385 003386 003386 003386 003386 003386 003386 003386 003389 003389 003390 003391 003392 003394 003395 003395 003396 003397 003398 003399 003397 003396 003397 003396 003397 003396 003397 003396 003397 003402 003402	0x0800089E 0x080008AA 0x080008AC 0x08008AC 0x08008C 0x0800008C 0x0800008C 0x0800008C 0x0800008C 0x0800000	E004 4807 4285 0204 4804 4026 4320 4304 4304 4309 6088 8039 6088 8031 8544 F001 8510 0004 F584 F584	В	param(15_WXC_PF3F(0ffset)); etvectorTable_4:	Group_4); riorityGroup)		Not execute Not execute	d d	
ndex 03064 03065 03066 03066 03067 03067 03071 03072 03073 03074 03075 03076 03077 03075 03076 03077 03078 03078 03081 03081 03083 03085 03085 03085	Frame 003382 003383 003384 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003392 003395 003396 003397 003398 003399 003390 003390 003390 003390 003390 003390 003401 003402 003403 003404 003405	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008AC 0x080008BE 0x080008BE 0x080008BE 0x080008C2 0x08008C2 0x08008C	E004 4807 4285 5285 4204 4804 4026 4320 4904 6809 6088 8031 F44F F001 B510 0004 F584 F584 F584	B B B SSEPT 27NVICS LDR CMP BCC_BDV 17NVICS LDR LDR LDR STR) POP WOV VOID NVIC_PT NVICS_PT NVICS ASSET CMP CMP CMP	param(15_WVLC_PFSF(0ffset)); etevetotrable_2: Ro, {PC, #0x12} R5, R0 R5, R0 R0, R0, R4 R1, PC, #0x10] R0, R0, R5 R0, R4, R3 R0, R0, R4 R0, R4, R3 R0, R0, R4 R0, R0, R4 R0, R4, R3 R0, R0, R4 R0, R0 R4, #0x500 R4, #0x500	Group_4); riorityGroup)		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
ndex 03064 03065 03066 03067 03068 03069 03070 03071	Frame 003362 003363 003384 003385 003386 003386 003387 003386 003387 003387 003389 003390 003392 003392 003394 003395 003396 003397 003396 003397 003396 003397 003396 003400	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008C2 0x08008C2 0x080008C2 0x080008C2 0x0800008C2 0x080008C2 0x080008C2 0x080000	E004 4807 4285 4285 4285 4204 4026 4320 4904 6609 6088 8031 F44F F001 8510 05004 F584 F584 F584 F584	B ASSENT 2PHVIC-S LDR SCID-S CMP BCC CMP MOV BL CMP CMP CMP CMP	param(15WVC_DPFSF(0ffset)); EverectorTable_2: Ro, [Pc, #0x12] RS, RO RO, [Pc, #0x12] RS, RO RO, RO, PC, #0x10] RO, RO, RS RO, RO, RS RO, RS, RA RO, RS, RO RO, RS, RA RO, RS, RO RO, RS, RO RO RO, RS, RO	Group_4); riorityGroup)		Not execute Not execute	d d	
ndex. 03064 03066 03066 03066 03067 03071 03072 03074 03077 03075 03076 03077 03076 03077 03076 03077 03076 03077 03076 03077 03080 03081	Frame 003382 003383 003384 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003385 003392 003395 003396 003397 003398 003399 003390 003390 003390 003390 003390 003390 003401 003402 003403 003404 003405	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008AC 0x080008BE 0x080008BE 0x080008BE 0x080008C2 0x08008C2 0x08008C	E004 4807 4285 4285 4285 4204 4026 4320 4904 6609 6088 8031 F44F F001 8510 05004 F584 F584 F584 F584	B assert ??NVIC_P LDR LDR CMP SCB-SV PRVIC ANDS CMP ORRS LDR NVIC PNP NVIC PNP NVIC PNP MOVS assert CMP CMP CMP CMP CMP B B CMP	param(15WIC_DPFSF(QffSet)); EVEXeCOTABLE_2: R0, [PC, #0x12] R5, R0 ietVectorTable_4: ietVectorTable_4: ietVectorTable_4: ietVectorTable_4: R0, R0, R5 R0, R0, R4 R1, PC, #0x10] R1, R1, #0x83] (00.44,R5,PC] R0, R0, R4 R1, PC, #0x10] R1, #0x83] (00.44,R5,PC] R1, #0x83] (00.44,R5,PC] R1, #0x83] (00.44,R5,PC] R1, #0x83] R4, #0x800 R4, #0x800	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
ndex. 03064 03066 03066 03066 03067 03071 03072 03074 03077 03075 03076 03077 03076 03077 03076 03077 03076 03077 03076 03077 03080 03081	Frame 007362 003363 003364 003365 003366 003367 003367 003387 003389 003390 003391 003392 003393 003394 003395 003396 003397 003396 003397 003396 003397 003396 003397 003396 003397 003396 003396 003397 003396 003397 003396 003400 003401 003402 003403 003404 003405	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008BE 0x080008C2 0x08008C2 0x0808C2 0x08008C2 0x08008C2 0x08008C2 0	E004 4807 4285 0204 4804 4026 4320 4304 4304 6088 8039 6088 8031 8510 0004 8510 0004 8510 0004 8510 0004 8510 4584 5584 5584	B assent PRVICE_S LDR CMP BCC_BDSU CMP CMP CMP CMP CMP CMP CMP CMP	param(15_WVLC_PFFF(0ffset)); Everetevetortable_2: R0, [Pc, #0x12] R5, R0 R0, R0, Pc, #0x12] R5, R0 R0, R0, R5 R0, R0, R4 R1, [Pc, #0x10] R0, R0, R5 R0, R0, R4 R1, [Pc, #0x10] R1, R1, [R1, #0x10] R0, R0, R5 R0, R4, R3 [R0,R4,R5, PC] (R0,R4,R5, PC] (R0,R4,R5, PC] (R0,R4,R5, PC] (R0,R4,R5, PC] (R1, #0x500 R4, #0x500 R4, #0x500 R4, #0x600 R4, #0x600 R4	Group_4); riorityGroup) IC_PriorityGroup)		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
udex. 03064 03066 03066 03067 03068 03069 03070 03071 03072 03073 03072 03073 03074 03075 03076 03076 03077 03076 03077 03076 03077 03076 03076 03076 03076 03077 03076 03076 03076 03076 03077 03076 03076 03077 03076 03076 03077 03076 03076 03077 03077 03076 03076 03076 03076 03076 03077 03077 03077 03076 03076 03077 03076 03076 03076 03077 03077 03076 03077 03076 03076 03076 03077 03076 03076 03076 03077 03077 03076 03076 03076 03076 03076 03076 03076 03076 03077 03076 03076 03076 03076 03076 03076 03076 03077 03077 03076 03076 03070 03070 03070 03072 03070 03072 03078 03080 00	Frame 003362 003363 003384 003385 003386 003386 003387 003386 003387 003387 003387 003387 003387 003387 003387 003389 003396 003397 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003406 003406 003406 003408	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008E 0x080008EC 0x08008EC 0x0808EC 0x0808EC 0x0808EC 0x0808EC 0x0808EC 0x0808EC 0x08	E004 4807 4285 5285 4204 4804 4026 4320 4904 4909 6088 8031 F44F F001 B510 0004 F584	B B B SSEPT PRVICE_S LDR CMP BCC_BDV CMP CMP CMP NVICE_PT NVICE_PT NVICE_PT CMP CMP CMP CMP CMP CMP CMP CMP	param(15WIC_DPFSF(Q)ffsc1)); EVEXCOTABLE_2: R0, [PC, #0x12] R5, R0 R0, R0, R4 R0, R0, R5 R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R1, R1, [R1] R0, R1, R1, 8 R0, R4, R5, PC] R0, R0, R4 R1, [PC, #0x10] R1, [R1] R0, [R1, #0x8] (R0,R4,R5, PC] R0, 40x300 NUC_PFIortSyCoupConfig R4, #0x800 R4, #0x800 R5, PC R5, PC R	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
 idex <li< td=""><td>Frame 003362 003363 003364 003385 003385 003386 003387 003387 003387 003387 003387 003387 003387 003387 003387 003392 003392 003394 003395 003396 003396 003397 003396 003397 003396 003397 003396 003397 003396 003401 003402 003404 003405 003406 003406 003406 003406 003408 003409</td><td>0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008C2 0x08008C2 0x0808C2 0x08008C2</td><td>E004 4807 4285 2285 4204 4604 4026 4320 4504 6085 8031 F44F F001 B510 0004 F584</td><td>B B SSEPT PNVCC_PT CMP CMP CMP CMP CMP CMP CMP CMP CMP CMP</td><td>param(15_WVC_DPFSF(Q)ffsc1)); EVEXCOTABLE_2: R0, [PC, #0x12] R5, R0 R0, R0, R2 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R1, [R1, #0x8] (R0,R4,R5,PC] R0, R0, R4 R1, [PC, #0x10] R1, [R1, #0x8] (R0,R4,R5,PC] R0, R0, R4 R1, [PC, #0x10] R4, #0 NVC_PFIortSyCoupConfig R4, #0x800 R4, #0x80</td><td>Group_4); riorityGroup) IC_PriorityGroup; 2</td><td></td><td>NOT EXECUTE NOT EXECUTE NOT EXECUTE</td><td>d d</td><td></td></li<>	Frame 003362 003363 003364 003385 003385 003386 003387 003387 003387 003387 003387 003387 003387 003387 003387 003392 003392 003394 003395 003396 003396 003397 003396 003397 003396 003397 003396 003397 003396 003401 003402 003404 003405 003406 003406 003406 003406 003408 003409	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008C2 0x08008C2 0x0808C2 0x08008C2	E004 4807 4285 2285 4204 4604 4026 4320 4504 6085 8031 F44F F001 B510 0004 F584	B B SSEPT PNVCC_PT CMP	param(15_WVC_DPFSF(Q)ffsc1)); EVEXCOTABLE_2: R0, [PC, #0x12] R5, R0 R0, R0, R2 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R1, [R1, #0x8] (R0,R4,R5,PC] R0, R0, R4 R1, [PC, #0x10] R1, [R1, #0x8] (R0,R4,R5,PC] R0, R0, R4 R1, [PC, #0x10] R4, #0 NVC_PFIortSyCoupConfig R4, #0x800 R4, #0x80	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
ndex. 03064 03066 03066 03066 03066 03066 03066 03066 03066 03070 03073 03073 03074 03074 03075 03076 03074 03075 03076 03074 03077 03076 03077 03076 03077 03076 03077 03078 03079 03080 03080 03080 03088 03088 03088 03088	Frame 007382 003383 003384 003385 003386 003386 003386 003387 003386 003387 003388 003389 003390 003391 003392 003394 003395 003396 003397 003396 003397 003396 003397 003396 003397 003397 003396 003397 003397 003396 003400 003401 003402 003403 0034046 003407 003401 003402 003403 003404 003403 003404 003403 003404 003403 003403 00	0x080089E 0x08008AA 0x08008AC 0x080008AC 0x080008AC 0x080008AC 0x080008EC 0x080008CC 0x080008CC 0x080008CC 0x080008CC 0x080008CC 0x080008CC 0x0800085A 0x0800085A 0x0800085C 0x080085 0x08008	E004 4807 4285 1204 4804 4026 4220 4904 4904 4904 6088 8003 6088 8003 8004 F587 F587 F577 F577 F577 F577 F577 F5777 F57777777777	B SSSEPT PRVIC-P LDR CMP SSCB-5-V CMP CMP STR DR STR DR STR DR STR DR STR DR STR DR STR DR STR CMP CMP CMP CMP CMP CMP CMP CMP	param(15_WIC_DPFSF(Q)ffset)); EvertextorTable L2: R0, [PC, #0x12] FS, R0 FS, R0 FS, R0 FS, R0 FS, R0 FS, R0 FS, R0 FS, R0 FS, R0 R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R1, B1, B1 R1, R1, B2 R0, R0, R4 R1, PC, #0x10] R0, R0, R5 R0, R1, B2 R0, R0, R4 R1, PC, #0x10] R1, R0 R1, PC, PT ortsynourconfig R1, #0x500 R4, #0x500 R5, R5, R5, R0, R5, R0, R0, R0, R0, R0, R0, R0, R0, R0, R0	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
udex: 03064 03064 03066 03066 03067 03067 03067 03073 03073 03073 03073 03073 03073 03073 03073 03073 03073 03074 03075 03076 03077 03077 03076 03079 03070 03079 03070 03077 03076 03079 03070 030000 0300000000	Frame 003362 003363 003384 003385 003386 003386 003387 003386 003387 003387 003389 003390 003392 003392 003394 003395 003395 003396 003396 003396 003396 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003407	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008BE 0x080008C2 0x08008C2	E004 4807 4285 0204 4804 4026 4320 4904 4904 4904 6088 8031 8510 0004 F585 F585	В	Lparam(12, NVLC_PF2F(Q)ff3ct)); every cortrable L2: R0, [PC, #0x12] R5, R0 R5, R0 R5, R0 R5, R0 R5, R0 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R1, R1 R0, R1, R1 R1, R0 R1, 90x10 R1, 40x10 R4, #0x500 R4, #0x500 R5, R0, R0, R1 R5, R0, R0, R1 R5, R0, R0, R1 R5, R0, R0 R5, R0	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
1dex 03064 03066 03066 03066 03066 03066 03067 03069 03071 03074 03074 03075 03088 03095 03095 03095 03095 03095 03095 03005 03055 030	Frame 003362 003363 003364 003385 003386 003386 003387 003386 003387 003387 003387 003387 003387 003387 003387 003389 003396 003397 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003386 003407 003406 003406 003407 003408 003410 003412 003423	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008E 0x080008EC 0x08008EC 0x0808C 0x08008EC 0x0808C	E004 4807 4285 0204 4804 4028 4220 4804 4220 4804 6088 B031 F001 B510 0004 F584	B B SSSEPT PRVICES LDR CMP BCC_BDSV CMP CMP CMP CMP CMP CMP CMP CMP	_param(15_WNC_DPFSF(Q)ffsc1)); EveretorCarbieL2: R0, [PC, #0x12] R5, R0 R0, R0, PC, #0x12] R5, R0 R0, R0, R1 R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R4, R4 R1, [PC, #0x10] R1, R1, #0 R0, R0, R4 R1, [PC, #0x10] R1, R1, #0 R0, R0, R4 R1, PC, #0x10 R4, #0x500 R4, #0x500 R5, [PC, #0x51] R0,	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	
udex: 33064 33064 33066 33067 33066 33067 33066 33067 33073 33073 33074 33073 33073 33074 33075 33077 33076 33077 33076 33077 33076 33077 33076 33077 33078 33077 33078 33079 33080 33080 33089 33089 33089	Frame 003362 003363 003384 003385 003386 003386 003387 003386 003387 003387 003389 003390 003392 003392 003394 003395 003395 003396 003396 003396 003396 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003406 003407	0x0800089E 0x080008AA 0x080008AC 0x080008AC 0x080008AC 0x080008BE 0x080008C2 0x08008C2	E004 4807 4285 0204 4804 4026 4320 4904 4904 4904 6088 8031 8510 0004 F585 F585	В	Lparam(12, NVLC_PF2F(Q)ff3ct)); every cortrable L2: R0, [PC, #0x12] R5, R0 R5, R0 R5, R0 R5, R0 R5, R0 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R0, R4 R1, [PC, #0x10] R0, R0, R5 R0, R1, R1 R0, R1, R1 R1, R0 R1, 90x10 R1, 40x10 R4, #0x500 R4, #0x500 R5, R0, R0, R1 R5, R0, R0, R1 R5, R0, R0, R1 R5, R0, R0 R5, R0	Group_4); riorityGroup) IC_PriorityGroup; 2		NOT EXECUTE NOT EXECUTE NOT EXECUTE	d d	

	52.63	5 25 😕 🗙 🗧	×	☑ ✔ ≫ 🐂 注 国 🗠 🖗 🕼 🔤 🤓 😣 🎂	
	.c stm32f10x			• × Dicassembly	
193	ifdef DEB	UG		Go to Memory Memory	
194	ifdef DEB debug(); endif			Professional DECO BLE DX800BE28	
197	ENTR CRT	SECTIONCY		0800BFA6 0800 LSRS RO, RO, #0×0 void main(void)	
198 199 200		clock system ();		main:	
201	// NUIC	init MB FLASH		main: 1:1: .constin at 8x20000000 */ .constin at 8x20000000 */ Iso_MM1, 8x00; .constin at 8x20000000 */ Iso_S0FAC F001FAS Bild .constin at 8x20000000 */	
203	/* Set t NUIC Set	he Vector Ta VectorTable(ble base : NUIC Vect1	location at 8×20000000 m/ Tab_RAM, 8×0); 08000FAA 8088 SUB SP, SP, #0×20 debug(): 08000FAC FO01F8A8 BL debug	
205 1	else /* /* Set t	UECT_TAB_FLA he Vector Ta	SH */ ble base :	lab_KHT, 0x07; location at 0x00000000 */	
207					
208 1 209 210				PriorityGroup_41: Number of the structors of the st	
	// SysTi SysTick_	ck end of co <mark>SetReload(90</mark>	unt event 0000);	each 0.1s with input clock equal to 9MHz (HCLK/8, defaul 0800BFBA F05F6000 MOVS R0, #0x8000000 NVIC_SetVectorTable	
212 213 214	<pre>// Enabl SysTick_</pre>	e SysTick in ITConfig <ena< td=""><td>terrupt BLE>;</td><td>each 0.15 with input clock equal to YMME (NGLA/8, defaul NTC_Pictory Constraints, State of the State of the</td><td>ntia</td></ena<>	terrupt BLE>;	each 0.15 with input clock equal to YMME (NGLA/8, defaul NTC_Pictory Constraints, State of the	ntia
215 216 217	SysTick_	CounterCmd <s< td=""><td>ysTick_Cou</td><td></td><td>ntig</td></s<>	ysTick_Cou		ntig
217 218 219	// Butto // GPIO	ns port init enable clock	and relea	ase Reset	
219 220 221	RCC_APB2	PeriphResetC	nd< RCC_f	Strick Inconfig(ENRER); PB22Periph_CP106, DISABLE); Swfick Inconfig(ENRER); PB22Periph_CP106, DISABLE); 0600FD0 2001 W0V5 R0, #0x1 PB22Periph_CP106, ENRBLE); 0800FD0 2001 W0V5 R0, #0x1	
	RCC_APB2	PeriphClockC		APB2Periph_GPI0A Systick_counterCmd(SysTick_counter_Enable): 0800BFD6_2001 MOVS R0, #0x1 APB2Periph_GPI0G_ENABLED: 0800BFD8_F00IFAEB_BL	
223	CRIC I				E):
225 226 227	GPIO_Ini GPIO_Ini	tStructure.G tStructure.G	PIO_Pin = PIO_Mode	0800BFDE F44F7082 MOV R0, #0×104	<u>- 7 4</u>
228	GPI0_Ini GPI0_Ini	tStructure.G t(B1_PORT, &	PIO_Speed GPIO_Inits	= GPIO_Speed_SUPHz; 0800BFE2 F7FFFA80 BL RCC_APB2PeriphResetC Structure); RCC_APB2PeriphClockCmd(_RCC_APB2Periph_GPIOA	md
229 230 231	GPI0_Ini	tStructure.G	PIO_Pin =):
232	GPI0_Ini GPI0_Ini	tStructure.G	PIO_Speed	B2_MBSK; IRC_AP82Pertiph GPTOG, ENABLE GPT0_Mode_IN_FLOATING; 08008FE6 GPL0_Speed_S0HHz; 08008FE8 Structure); 08008FE8 GPL0_Speed_S0HHz; 08008FE8 RC_AP82Pertiph Clock RC_AP82Pertiph Clock	
233 234 235		SECTION();	GPIU_Init:		
235 236 237				DSDDBEE4 ESADDDDD STEH RD [SP]	
237 238 239	// HN_IR	e ADC1 and G	PIOC clock	k B2Periph_ABC1 RCC_APB2Periph_GFIOC, DISABLED; S2Periph_ABC1 RCC_APB2Periph_GFIOC, ENSABLED; S2Periph_ABC1 RCC_APB2Periph_GFIOC, ENSABLED;	
239	RCC_APB2	PeriphClockC	nd (RCC_API	B2Periph, HDC1 RCC_APB2Periph_CPIOC, ENBLES;	
	— (4				
	Frame	Address	Oncode	Trace Comment	
idex 02368	Frame 002686	Address 0x0800B5A4	Opcode B510	Trace Comment RCC_GetFlagStatus(u8)	
idex 02368 02403 02407	Frame 002686 002721 002725	0x0800B5A4 0x0800BEBE 0x0800B5A4	8510 2800 8510	RCC_GetFlagStatus(u8) Clk_Init() + 66 RCC_GetFlagStatus(u8)	
idex 02368 02403 02407 02442 02446	Frame 002686 002721 002725 002760 002764	0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800BEBE 0x0800B5A4	8510 2800 8510 2800 8510	RCC_GetFlagStatus(u8) Clk_Init() + 66 RCC_GetFlagStatus(u8) Clk_Init() + 66 RCC_GetFlagStatus(u8)	
idex 02368 02403 02407 02442 02446 02481	Frame 002686 002721 002725 002760 002764 002799	0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800B5A4	8510 2800 8510 2800 8510 2800	Rcc_setFlapStatus(u8) clk_int() + s6 Rcc_setFlapStatus(u8) clk_int() + s6 Rcc_setFlapStatus(u8) clk_int() + s6	
ndex 02368 02403 02407 02442 02446 02481 02485 02520	Frame 002686 002721 002725 002760 002764 002799 002803 002838	0x0800B5A4 0x0800B5B4 0x0800B5A4 0x0800B5BE 0x0800B5A4 0x0800B5B4 0x0800B5A4 0x0800B5B4	B510 2800 B510 2800 B510 2800 B510 2800	RcC_setPlapStatus(U8) clk_init() + 66	
ndex 02368 02403 02407 02442 02446 02481 02485 02520 02524 02559	Frame 002686 002721 002755 002760 002764 002799 002803 002838 002838 002842 002877	0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800	RcC_setPlapStatus(U8) clk_init() + 66	
ndex 02368 02403 02407 02442 02446 02481 02485 02520 02524	Frame 002686 002721 002725 002760 002764 002799 002803 002838 002842	0x0800B5A4 0x0800B5BE 0x0800B5BE 0x0800B5BE 0x0800B5B4 0x0800B5B4 0x0800B5A4 0x0800B5B4	B510 2800 B510 2800 B510 2800 B510 2800 B510	RcC_setPlapStatus(U8) clk_init() + 66	
ndex 22368 22403 22407 22442 22446 22481 22481 22520 22524 22559 22559 22563 22598 22602	Frame 002686 002721 002725 002760 002764 002799 002803 002838 002842 002877 002881 002916 002916	0x0800B5A4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B5 0x0800B5B4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510	Rcc_cetTlaptatus(u8) clk_init() + 66 Rcc_cetTlaptatus(u8) clk_init() + 66 Rcc_cetTlaptatus(u8) clk_init() + 66 Rcc_setTlaptatus(u8)	
ndex 22368 22407 32407 32442 32446 32485 32520 32524 32559 32559 32563 3259 32563 3259 32602 32702 32602 3270	Frame 002686 002721 002725 002764 002769 002803 002838 002842 002877 002881 002916 002920 002955 002959	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510	Rcc_cetTlapStatus(u8) clk_Init() + 66 Rcc_cetTlapStatus(u8)	
ndex 02368 02403 02407 02442 02446 02481 02485 02524 02529 02524 02559 02524 02559 02524 02598 02602 02670 02680	Frame 002686 002721 002725 002760 002764 002799 002803 002838 002842 002877 002881 002916 002920 002955 002959 002994 002998	0x080085A4 0x080085A4 0x0800854 0x0800854 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10	Rcc_setPlastatus(u8) clk_init() + 66 Rcc_setPlastatus(u8)	
ndex 02368 02403 02407 02442 02481 02485 02520 02524 02559 02525 025263 02598 02602 02641 02676	Frame 002686 002721 002725 002764 002799 002803 002838 002842 002877 002881 002916 002920 002955 002959 002954	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085B4 0x080085B4 0x080085B4 0x080085B4 0x080085B4 0x080085B4 0x080085B4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800	RcC_getPlaStatus(u8) clk_Init() + 66	
dex 02368 02407 02442 02442 02485 02520 02529 02529 02529 02529 02529 02546 02641 02641 02640 02641 02641 02676 02641 02675 02719 02754	Frame 002686 002725 002750 002760 002780 002780 002838 002838 002842 00295 00295 002994 002994 003033 003037	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800	RcC_getPlagStatus(u8) clk_Init() + 66	
dex 22368 2403 2407 22442 22485 22529 22559 22563 22563 22602 22637 22641 22676 22602 22637 22641 22676 22719 22754 22793	Frame 002666 002721 002725 002764 002764 002803 002803 002842 002842 002855 002956 001994 001994 001994 001994 001003 001003 001076 003011	0x0800854 0x0800855 0x0800855 0x0800855 0x0800855 0x0800855 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x080085 0x085 0x085 0x08085 0x085 0x085 0x085 0x085 0x085 0x085	B\$10 2800 B\$10 8510 B\$10 2800 2800 2800 2800 2800 2800 2800 28	RcC_getPlapStatus(u8) clk_Thit() + 66 RcC_getPlapStatus(u8) <t< td=""><td></td></t<>	
ddex 22368 22403 2247 22446 22481 22486 22520 22524 22524 22528 22528 22528 22637 22641 22641 22641 22641 22641 22676 22719 22758 22758 22793 22793 22793	Frame 002666 002721 002725 002764 002764 002780 002803 002803 002842 002855 002956 001994 001994 001303 003037 003037 003037 003015 003115 003150	0x08008548 0x08008 0x08008548 0x08008548 0x08008548 0x08008548 0x08008548 0x0	B510 2800 B510 2800 B510 2800 2800 8510 2800 850 850 850 850 850 850 850 850 850	Rcc_getPlayStatus(u8) clk_Init() + 66 Rcc_getPlayStatus(u8) <td></td>	
ddex 22368 22403 32407 22442 32442 32446 32442 32445 32520 32524 32553 32598 32598 32607 32641 32663 32715 32715 32715 32715 32715 32715 32754 32755 322755 327555 327555 327555 327555 327555 327555 3275555 3275555 3275555 32755555 32755555 327555555 327555555555555555555555555555555555555	Frame 002686 002721 002725 002764 002760 002764 002780 002838 002838 00290 002830 002832 002916 002916 002920 002939 002939 003072 003072 003111 003150	0x08008544 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544	B510 2800 8500 2800 8500 2800 8500 8500 850	RcC_getPlaStatus(u8) Clk_Init() + 66 RcC_getPlaStatus(u8) RcC_	
dex 22 407 22 407 22 407 22 442 22 481 22 481 22 481 22 482 22 52 42 55 22 52 22 53 22 53 22 55 22 55 22 63 22 63 22 63 22 63 22 63 22 63 22 715 22 75 22 75 25	Frame 002686 002721 002721 002725 002725 002702 002702 002802 001802 001802 001802 001802 001802 001802 001802 001910 001911 001011 001180 001180 001180	0x08008544 0x08008545 0x0800854 0x08008008 0x080080080 0x08000800000000	B510 2800 B510 2800 B510 2800 B510 2801 2802 2803 B510 2801 2801 2802 8510 2800 B510	RcC_GetPlaStatus(u8) Clk_Init() + 66 RcC_SetPlaStatus(u8) Clk_	
dex 22368 22407 22407 22442 22442 22481 12485 2252 2254 22555 22524 22555 22525 22555 22525 22555 22755 22755 22575 225852 22687 22687 22687 22687 22687 22687 22687 22687 22687 22687 22687 22685	Frame 002686 002721 002725 002725 002764 002780 002803 002803 001803 001803 001803 001803 0018916 001995 001995 001995 001072 001072 001072 001111 001150 001150 001189 001189 001201 001201 001201	0x0000544 0x0000554 0x000555 0x000554 0x0005554 0x0005554 0x0005554 0x0005556 0x0005556 0x0005556 0x0005556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000556 0x000	B\$10	Rcc_getPlagstatus(u8) clk_Init() + 66 Rcc_getPlagstatus(u8) <t< td=""><td></td></t<>	
dex 22 368 22 403 22 407 22 442 22 481 22 481 22 481 22 524 22 524 22 524 22 525 22 524 22 525 22 524 22 525 22 602 22 602 22 662 22 662 22 662 22 662 22 676 22 676 22 759 22 754 22 754 22 757 22 836 22 836 22 883	Frame 002666 002721 002725 002764 002780 002780 002780 002838 002838 002950 002950 002950 002959 003076 003017 003015 003115 003154 003154 003130 003130 003130 003154 003130 003133 003133	0x0000544 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554	B510 2800 B510 B510 B510 B510 B510 B510 B510 B510	RcC_getPlaStatus(us) clk_Init() + 66 RcC_getPlaStatus(us) clk_	
dex 22368 32407 32442 32407 32442 324461 32481 32486 32481 32520 32520 32524 32520 32522 32602 32715 32735 3274 32755 32675 32685 32685 32695 32685 32695 32685 32695 32685 32695 32685 32695 32685 32695 32685 32695 32685 32695 32685 32695 32685 32695 32695 32685 32906 32906 32906 32906 32906 32906 32907 32906 32906 32907 32906 32907 32907 32906 32907 3007	Frame 002686 00271 00271 00271 00271 00271 00271 00271 00271 00271 002760 002761 002762 002763 002838 002842 002847 002946 002994 002994 002994 001072 001072 001114 001154 001203 001224 001224 001223 001224	0x0000544 0x0000554 0x0000554 0x0000554 0x000054 0x000554 0x000554 0x	B\$10	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getClastConfig(us2) Clk_Init() + 76 RcC_getDisconfig(us2) Clk_Init() + 76 RcLini() + 780 RCL_Ro	
dex 22368 22403 22407 22442 22407 22442 22485 22485 22520 02524 22559 02559 02559 02559 02559 02637 02641 02675 02641 02675 02671 02675 02671 02754 02779 02754 02675 02677 02675 027500 0000000000	Frame 001565 001721 002721 002721 002760 002760 002790 002842 002842 002790 002843 002843 00295 002950 002955 002998 001072 001075 001076 001150 001150 001150 0012921 0012201 001223 0012241 001243 001243	0x0000544 0x0000556 0x0000556 0x0000560 0x0000556 0x0000556 0x0000560 0x0000560 0x0000560 0x0000560 0x0000560 0x0000560 0x0000	B\$10	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_actLoonfigut2) Clk_Init() + 76 Rcc_actCoconfigut2) Clk_Init() + 76 Rcc_actLoonfigut2) Clk_Init() + 76 Rcc_actLoonfigut2) Clk_Init() + 76 Rcc_actLoonfigut2) Clk_Init() +	
dex 22368 22403 22407 22442 22485 22485 22485 22524 22524 22524 22525 22524 22525 22524 22525 22637 22637 22637 22637 22637 22637 22637 22637 22637 22637 22637 22637 22637 22719 22754 22754 22753 22832 22908 22908 22928 22942 29442 29442 2944 2944 2944	Frame 001565 001721 001721 001721 001721 001721 001721 001721 001721 0017260 002760 002831 0028431 0018427 001916 0019201 001950 001951 001072 001072 001071 001150 001150 001150 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224	0x0000544 0x0000550 0x0000550 0x0000500 0x0000500 0x0000500000000	B5.0	Rcc_getPlayStatus(us) clk_init() + 66 Rcc_getPlayStatus(us) <t< td=""><td></td></t<>	
dex 22 368 22 402 22 50 22 50 20 50 22 50 20 50 2	Frame 002585 002721 002721 002760 002760 002760 002780 002780 002780 002781 002780 002783 002831 002831 002831 002831 002955 002959 002959 002959 001037 001072 001015 001115 001150 001203 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001226 001277 001277 001277 001277 001277 001277	0x20005A4 0x20005E20 0x00005E4 0x0005E4 0x00	B510 2800 B510 2000 B510 2000 B510 2000 B510 2001 2010 2020	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 84 RcC_pecLuconfig(ul2) Clk_Init() + 84 RcC_pecLuconfig(ul2) Clk_Init() + 84 RcC_pecLuconfig(ul2) Clk_	
dex 22 368 22 402 22 52 22 52 22 52 22 52 22 52 22 52 22 52 22 52 22 53 22 53 22 75 22 75 22 83 22 83 22 83 22 83 22 83 22 802 22 803 22 803 22 803 22 906 22 902 22 942 22 942 22 942 22 942 22 942 22 945 22 985 22	Frame 001565 001721 002721 002760 002760 002770 002799 002831 002831 002831 002831 002831 002831 00295 00295 002959 002959 003072 003072 0031150 0031160 003203 003224 0032243 0032243 003227 003227 0032243 0032243 003227 003277 003203 0032243 0032243 0032243 003277 003277 003277 003203 003243 003277 003303 003303	0x20005A4 0x20005E20 0x20005A1 0x2005A1 0x	B 5 10 2 800 B 5 10 2 8 50 2 8 50 2 8 50 2 8 50 2 8 50 </td <td>RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_</td> <td></td>	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_	
dex 22 368 22 403 22 407 22 404 22 408 22 407 22 442 22 448 22 481 22 482 22 482 22 482 22 481 22 482 22 481 22 583 22 583 22 682 22 758 22 987 22 987 23 987 2	Frame 001656 001721 002721 002721 002760 002760 002760 002760 002760 002760 002843 002843 002843 002843 00290 00295 002959 002959 001072 001072 001074 001115 001116 001110 001224 001224 001224 001224 001224 001224 001226 001226 001226 001226 001201 001201 001226 001201 001201 001226 001201 001201 001226 001201 001201 001201 00120	0x20005A4 0x20005E20 0x20005A1 0x2005A1 0x2	B510 2800 B510 B510 B510 B510 B510 B510 B510 B510	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getClaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getPlaStatus(us) Clk_Init() + 76 RcC_getClastif(us) RcC_getClastif(us) Clk_Init() + 76 RcC_getC	
dex 22 368 22 402 22 402 22 402 22 402 22 442 22 442 22 442 22 445 22 481 22 481 22 481 22 481 22 481 22 481 22 481 22 481 22 52 42 52 22 632 22 632 22 632 22 632 22 735 22 735 22 735 22 737 22	Frame 002656 002721 002721 002750 002760 002760 002760 002780 002780 002780 002780 002780 00280 00280 00281 002916 002916 002917 002918 002916 002917 002916 002916 001072 001072 001115 001154 001224 001224 0012240 0012241 0012241 0012241 0012241 0012260 0012241 0012241 001305 001305	0x00005A4 0x00005E4 0x00005E4 0x00005E4 0x00005E4 0x00005E4 0x00005E4 0x00005E4 0x00005E4 0x00005A5 0x00055A4 0x00005E4 0x0005E	B5.0 2800	Rcc_getPlagstatus(us) clk_init() + 66 Rcc_getPlagstatus(us) <t< td=""><td></td></t<>	
dex 22 368 22 403 22 407 22 403 22 407 22 408 22 52 22 52 22 52 22 59 22 59 29 59	Frame 002656 002721 002751 002760 002760 002760 002760 002760 002760 002770 002802 002760 002802 002802 002802 002802 002915 002950 002950 002950 002950 002950 002950 002950 002950 002950 001037 003072 0031110 001031 002203 003224 003224 003224 003224 003227 003226 003277 003303 003139 003224 003277 003303 003303 003304 003304 003305 003304 003305 003354 003354 003354	0x20005A4 0x20005E20 0x00005E20 0x00005E20 0x00005E20 0x00005E20 0x00005E20 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005A2 0x00005E20 0x0005E20	B510 2800 B510 2000 B510 2000 B510 2000 B510 2001 2002 B510 2002 B510 2002 B510 2002 2002	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 86 RcC_getPlaStatus(us) Clk_Init() + 184 RcC_getPlaStatus(us) Clk_Init() + 184 RcC_getPlaStatus(us) Clk_Init() + 184 RcC_getPlaStatus(us) C	
dex 22 368 22 403 22 407 22 403 22 407 22 408 22 52 22 52 22 52 22 52 22 52 22 52 22 52 22 53 22 55 22 55 25 55	Frame 002585 002721 002721 002750 002760 002770 002780 002799 002831 002831 002831 002831 002831 00295 002959 002959 002959 002077 001077 001077 001071 001115 0012944 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001224 001277 001270 001270 001270 001270 001270 001270 001270 001	0x20005A4 0x20005E20 0x20005F40 0x2005F40 0x20000F40 0x20000F40 0x2000F40	B 5 10 2 800 2 801 2 802 2 802 2 802 2 802 2 802 <tr< td=""><td>RcC_getPlaStatus(us) Clk_Init() + 66 RcC_getPlaStatus(us) Clk_Init() + 66 RcC_usetPlaStatus(us) Clk_Init() + 66 RcC_usetPlaStatus(us) Clk_Init() + 66 RcC_usetPlaStatus(us) Clk_Init() + 66 RcC_usetLoconfig(us2) Clk_Init() + 66 RcC_usetLoconfig(us2) Clk_Init() + 100 RcC_usetLoconfig(us2) <</td><td></td></tr<>	RcC_getPlaStatus(us) Clk_Init() + 66 RcC_usetPlaStatus(us) Clk_Init() + 66 RcC_usetPlaStatus(us) Clk_Init() + 66 RcC_usetPlaStatus(us) Clk_Init() + 66 RcC_usetLoconfig(us2) Clk_Init() + 66 RcC_usetLoconfig(us2) Clk_Init() + 100 RcC_usetLoconfig(us2) <	
dex 22 368 22 403 22 407 22 408 22 407 22 408 22 509 22 509 22 509 22 509 22 509 22 607 22 906 22 906 22 925 22 944 22 967 33 001 33 031 33 033 30 55 40 40 40 40 40 40 40 40 40 40	Frame 002656 002721 002721 002760 002760 002760 002780 002780 002780 002780 002780 002781 002877 002881 002916 002990 002991 002994 001007 001007 001150 001291 001	0x20005A4 0x20005E4 0x20005E4 0x20005E4 0x20005E4 0x20005E4 0x20005E4 0x20005E4 0x20005A4 0x2005A4 0x2	B510	Rcc_getPlagStatus(u8) clk_init() + 66 Rcc_getPlagStatus(u8) <t< td=""><td></td></t<>	

CHAPTER 11

		6 陥 💼 🗠 🤉	si l	<u>Window Belp</u> ▼ ペ > > 2 ⊠ ⇒ ≁ 43 B+ 05 100 SK.	26 Ab Ab					
	en en:	5 25 27 X	* []		- -					
					* x	Disassembly				
glcd_l. 74	c stm32f10x	_nvic.c	FR-			Goto	-	Memory	• 🖻	
75	SCB->HFS SCB->DFS Function Descript	$ \begin{array}{l} \mathbf{R} &= \boldsymbol{0} \times \mathbf{FFFFFF} \\ \mathbf{R} &= \boldsymbol{0} \times \mathbf{FFFFFF} \end{array} $	FF;		Ξr	text 661				
22		*****				0800D842			LR	
79 ×	Function	Name : NVI ion : Con				DMA2_Channe1 DMA2_Channe1	1_IRQHandler 1_IRQHandler 4770 B×			
81 *	Output Return id NUIC_ /* Check	ion : Con and	subpriori	GroupConfig priority grouping: pre-emption priority tyGroup: specifies the priority grouping bits s parameter can be one of the following values: riorityGroup. This for pre-emption priority for subpriority for subpriority for subpriority for subpriority		0800D844	4770 B×		LR	
82 *	Input	: - N 1	ength. Thi	tyGroup: specifies the priority grouping bits s parameter can be one of the following values:		DMA2_Channel	2_IRQHandler 2_IRQHandler	:		
84 × 85 ×			- NUIC_P 4 bits	riorityGroup_0: 0 bits for pre-emption priority for subpriority		.text_68: 0800D846	4770 BX	•	LR	
86 × 87 ×			- NUIC_P 3 bits	riorityGroup_1: 1 bits for pre-emption priority for subpriority					LR	
88 × 89 ×			- NUIC_P	riorityGroup_2: 2 bits for pre-emption priority		DMA2_Channel	3_IRQHandler 3_IRQHandler 4770 BX			
90 ×			- NUIC_P	for subpriority for subpriority for subpriority for subpriority for subpriority riorityGroup_4: 4 bits for pre-emption priority for subpriority					LR	
92 ×			- NUIC_P	riorityGroup_4: 4 bits for pre-emption priority		DMA2_Channel	4_5_IRQHand1 4_5_IRQHand1	er: er:		
94 ×	Output	: Non : Non	e	for supprovity					18	
96 **	Keturn	*********	*********						LR NVIC_PriorityGroup)	
97 00	oid NUIC_			2 NUIC_PriorityGroup)		NVTC Priorit	yGroupConfig VGroupConfig	:		
100	/* Check assert_p	the paramet aram(IS_NUIC	ers */ _PRIORITY_	GROUP <nuic_prioritygroup>>;</nuic_prioritygroup>		.text_5: 0800D84C	ýGroup⊂onfiĝ 8510 PU:	SH	{R4,LR}	
				according to NUIC_PriorityGroup value */ K NUIC_PriorityGroup;		0800D84E assert par	am(IS_NVIC P	VS RIORITY G	(R4,LR) R4, R0 R0UP(NYTC_priorityGroup)): R4, #0X700 7?NVIC_priorityGroupConfig_0 R4, #0X600 22NVIC_priorityGroupConfig_0 0.	
	SCB->AIR	CR = AIRCR_U	ECTREY_MAS	K NVIC_PriorityGroup;		0800D850 0800D854	F5B46FE0 CM D00B BE	р 9	R4, #0×700 ??NVIC_PriorityGroupConfig 0	
104 / 105 106 /•			********	*****		0800D856 0800D85A	F5B46FC0 CM D008 RFI	P Q	??NVIC_PriorityGroupContig_0 R4, #0x600 ??NVIC_PriorityGroupContig_0 R4, #0x500 ??NVIC_PriorityGroupContig_0 R4, #0x400 ??NVIC_PriorityGroupContig_0 R4, #0x300	
107 *	Function	Name : NVI	C_Init	<pre>www.www.www.www.www.www.www.www.www. he NUC_initStruct. ruct: pointer to a NUC_initTypeDef structure as the configuration information for the UC peripheral.</pre>		0800D85A 0800D85C 0800D85C	F5B46FA0 CM D008 BEI F5B46FA0 CM D005 BEI F5B46F80 CM	P n	R4, #0×500 22NVIC PriorityGroupConfig 0	
109 🐱	Descript	ion : Ini par	tializes t. ameters in	he NVIC peripheral according to the specified the NVIC_InitStruct.		0800D862 0800D866	F5B46F80 CM D002 BE	P n	R4, #0x400 22NVIC PriorityGroupConfig 0	
110 × 111 ×	Input	: - N t	UIC_InitSt hat contai	ruct: pointer to a NUIC_InitTypeDef structure ns the configuration information for the				P F	R4, #0×300 ??NVIC_PriorityGroupConfig_1	
112 ×	Output	s : Non	pecified N	VIC peripheral.		??NVIC_Prior	F5847F40 CM D100 BN ityGroupConf E004 B ityGroupConf 2164 MD F8DF005C LD F7FEFCAB BL - ATPC CC	ig_0:	??NVIC_PriorityGroupConfig_2	
114 *	Output Return	: Non : Non	e			??NVIC_Prior	ityGroupConf	ig_1:		
115 ** 116 vc	id NUIC_			NVIC_InitStruct>		08000872	F8DF005C LD	v. R.W	R0, [PC, #0×5C]	
						SCB->AIRCR	= AIRCR_VEC	TKEY_MASK	R1, #0x64 R0, [PC, #0x5C] assert_failed (NVIC_PriorityGroup;	
119	u32 tmpp	re = 0, tmps	$ub = 0 \times 0F;$	= 0×00, tmpmask = 0×00;		??NVIC_Prior 0800D87A	TTYGROUPCONT F8DF0058 LDI 6800 LDI	1g_2: R.W	R0, [PC, #0×58] R0, [R0]	
121	∕* Check	the paramet	ers */		<u> </u>	0800D87E	6800 LD	R	R0, [R0]	100000
					<u> </u>	<u> </u>				
۹ 🍾	· 🔛 🕺									
lex	Frame	Address	Opcode	Trace Comment						
2368	002686 002721	0×0800B5A4 0×0800BEBE	8510 2800	RCC_GetFlagStatus(u8)						
2368 2403 2407	002686 002721 002725	0x0800B5A4 0x0800BEBE 0x0800B5A4	8510 2800 8510	RCC_GetFlagStatus(u8) Clk_Init() + 66 RCC_GetFlagStatus(u8)						
2368 2403 2407 2442 2446	002686 002721 002725 002760 002764	0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800BEBE 0x0800B5A4	8510 2800 8510 2800 8510	RCC_GetFlagStatus(u8) Clk_Init() + 66 RCC_GetFlagStatus(u8) Clk_Init() + 66 RCC_GetFlagStatus(u8)						
2368 2403 2407 2442 2446 2481	002686 002721 002725 002760 002764 002799	0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800BEBE 0x0800BEBE 0x0800B5A4 0x0800BEBE	8510 2800 8510 2800 8510 2800	RCC_GetFlagStatus(U0) Clk_Int() + 66 RCC_GetFlagStatus(U0) Clk_Int() + 66 RCC_GetFlagStatus(U0) Clk_Int() + 66						
2368 2403 2407 2442 2446 2481 2485 2485 2520	002686 002721 002725 002760 002764 002799 002803 002838	0x0800B5A4 0x0800BEBE 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5BE 0x0800B5A4	B510 2800 B510 2800 B510 2800 B510 2800 2800	RCC_setFlagStatus(U0) Clk_int() + s6						
2368 2403 2407 2442 2446 2481 2485 2520 2524	002686 002721 002725 002760 002764 002799 002803 002838 002838	0x0800B5A4 0x0800B5BE 0x0800B5BE 0x0800B5A4 0x0800B5B4 0x0800B5B4 0x0800B5B4 0x0800B5B4	8510 2800 8510 2800 8510 2800 8510 2800 8510 2800 8510	RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8)						
2368 2403 2407 2442 2446 2481 2485 2520 2524 2559 2563	002686 002721 002725 002760 002764 002799 002803 002838 002842 002877 002881	0x0800B5A4 0x0800B5B4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4 0x0800B5A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6)</pre>						
2368 2403 2407 2442 2446 2481 2485 2520 2524 2559 2563 2598	002686 002721 002725 002760 002764 002799 002803 002838 002842 002877 002881 002916	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	8510 2800 8510 2800 8510 2800 8510 2800 8510 2800 8510 2800	RCC_setPlagStatus(U0) Clk_Int() + s6 RCC_setPlagStatus(U0) Clk_Int() + s6 RCC_setPlagStatus(U0)						
2368 2403 2407 2442 2446 2481 2485 2520 2524 2559 2563 2598 2602 2637	002686 002721 002725 002760 002764 002799 002803 002838 002842 002877 002881 002916 002920 002955	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800	<pre>RCC_setPlagStatus(U0) Clk_int() + 66 RCC_setPlagStatus(U0) Clk_int() + 66 RCC_setPlagStatus(U0) Clk_int() + 66 RCC_setPlagStatus(U0) Clk_int() + 66 RCC_setPlagStatus(U0) Clk_int() + 66 RCC_setPlagStatus(U0) Clk_int() + 66 RCC_setPlagStatus(U0) Clk_int() + 66</pre>						
2368 2403 2407 2442 2446 2481 2485 2520 2524 2559 2563 2598 2602	002686 002721 002725 002764 002799 002803 002838 002842 002877 002881 002842 002877	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510	<pre>RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8)</pre>						
2368 2403 2407 2442 2446 2441 2445 2520 2524 2520 2524 2559 2563 2598 2602 2637 2641 2676 2680	002686 002721 002725 002760 002764 002799 002803 002842 002842 002842 002841 002916 002920 002955 002959 002994	0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4 0x080055A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510	<pre>RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8)</pre>						
2368 2403 2407 2442 2446 2481 2485 2520 2524 2559 2563 2598 2563 2598 2602 2602 2602 2602	002686 002721 002725 002760 002764 002799 002803 002838 002842 002877 002881 002842 002877 002881 002916 002920 002955 002955 002954	0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4 0x080085A4	B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800 B510 2800	RCC_GetPlagStatus(U6) Clk_Init() + 66						
2368 2403 2407 2442 2445 2485 2520 2524 2559 2563 2598 2598 2598 2598 2598 2602 2598 2602 2641 2676 2641 2676 2641 2675 2719 2754	002686 002721 002725 002755 002764 002784 002803 002842 002877 0028842 002877 0028842 002916 002916 002955 002955 002959 002959 002959 002959 002959 002959 002959 002959 002959 002959 002959	0x0800554 0x0800554	B510 2800	RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) RCC_GETPlagStatus(U6) RCC_GETPl						
2368 2403 2447 2442 2446 2481 2485 2520 2525 2520 2525 2559 2563 2598 2602 2637 2641 2676 2680 2715 2719 2758	002686 002721 002725 002750 002760 002760 002760 002780 002838 002838 002838 002838 002887 002887 002838 00282946 002920 002955 002955 002959 002998 002033 003076 003072	0x08008544 0x08008545 0x0800855 0x0800855 0x0800855 0x0800855 0x0800855 0x0800855 0x0800855 0x0800855 0x080085 0x08085 0x08000085 0x08000085 0x0	B510 2800	<pre>RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66</pre>						
2368 2403 2407 2442 2446 2446 2485 2524 2524 2524 2524 2524 2524 2529 2563 2592 2602 2602 2602 2602 2602 2602 2602 26	002686 002721 002725 002750 002760 002760 002760 002764 002799 002803 002881 002881 00281 002895 002881 002955 002959 002995 002995 002994 002995 002994 002995 003075 003076	0x08008544 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544 0x08008544	B510 2800	RCC_GetPlagStatus(U6) Clk_Init() + 66 RCL_GetPlagStatus(U6) RL(Linit() + 66 RCL_GetPlagStatus(U6) RL(Linit() + 66 RCL_GetPlagStatus(U6) RL(Linit() + 66 RCL_GetPlagStatus(U6)						
2368 2403 2442 2442 2442 2445 2445 2445 2520 2524 2553 2598 2602 26641 26641 2676 26641 2676 2675 2719 2758 2719 2758 2793 2797 2836	002666 002721 002725 002725 002760 002760 002760 002760 002842 002838 002842 002842 00292816 00292816 00292816 0029298 002959 002959 002959 002994 002998 002998 002998 0020355 002994 002095 0020355 002055 00000000	0x08008544 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008584 0x080858 0x0808 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x0808 0x0808 0x080858 0x0808 0x0808 0x0808 0x0808 0x0808 0x0808 0x0808 0x08	B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 8510 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 2800 2800 2800 2800 2800 2800 28	RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) RCC						
2368 2407 2442 2444 2442 2444 2485 2524 2524 2524	022665 002725 002725 002760 002760 002760 002764 002799 002838 002838 002838 002842 002877 002838 002946 002955 002959 002994 002994 002994 002994 002994 002994 002955 002955 002955 002955 002954 002955 002954 002955 002954 002955 002955 002954 002955 002954 002955 002955 002955 002954 002955 002055 000055 000000	0x08008544 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x08008545 0x0800854 0x08000	B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800	<pre>RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) RCC_HCL() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatus(U8) RCC_GETFlagStatu</pre>						
2368 2403 2442 2442 2442 2445 2445 2445 2520 2524 2553 2598 2602 26641 26641 2676 26641 2676 2675 2719 2758 2719 2758 2793 2797 2836	002666 002721 002725 002725 002760 002760 002760 002760 002842 002838 002842 002842 00292816 00292816 00292816 0029298 002959 002959 002959 002994 002998 002998 002998 0020355 002994 002095 0020355 002055 00000000	0x08008544 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008586 0x08008584 0x080858 0x0808 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x080858 0x0808 0x0808 0x080858 0x0808 0x0808 0x0808 0x0808 0x0808 0x0808 0x0808 0x08	B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 8510 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 2800 2800 2800 2800 2800 2800 28	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) RCC_GETFlagStatus(U6) RCC_GETFlagStatus(U6) RCC_GETFlagStatus(U6) RCC_GETFlagStatus(U6) R</pre>						
2368 2403 2447 24442 2446 2481 2524 2524 2559 2559 2559 2559 2559 2559	022665 002721 002725 002764 002764 002803 002842 002877 002887 002887 002887 002887 002894 002994 002995 002994 002995 002994 002995 002994 003037 003035 003111 003115 0031150 003183 003183 003201	0x00005A4 0x000025A 0x000025A 0x00005A4 0x00005A4 0x00005A4 0x00005A4 0x00005A4 0x00005A4 0x00005A4 0x00005A4 0x00005A6 0x00005A4 0x0000	B\$10 B\$10	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetClagStatus(U6) Clk_Init() + 66 RCC_GetClagStatus(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTATUS(U6) RCC_GETCLAGSTAT</pre>						
2368 2403 2442 2444 2444 2559 2552 2552 2552 2552	002666 002721 002725 002764 002764 002801 002800 002800 002842 002871 002887 002887 002887 002894 002994 002995 002994 002995 002994 003037 003072 003072 003072 003072 003115 003115 003150 003153 003224 003224	0x00005A4 0x00005E4 0x00005A1 0x0000	B\$10 B\$10 2800 B\$10	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetClagStatus(U6) Clk_Init() + 66 RCC_GetClagStatus(U6) Clk_Init() + 66 RCC_GetClagStatus(U6) Clk_Init() + 66 RCC_GetClagStatus(U6) Clk_Init() + 64 RCC_GetClagStatus(U6) Clk_Init() + 64 RCC_GetClagStatus(U6) Clk_Init() + 64 RCC_GetClagStatus(U6) Clk_Init() + 64 RCC_GetClagStatus(U6) Clk_Init() + 64 RCC_GEtClagStatus(U6) Clk_Init() + 64 RCC_GETCLAGSTAGUS(U6) Clk_Init() + 64 RCC_GETCLAGSTAGUS(U6) R</pre>						
2368 2403 2442 2444 2481 2485 2552 2553 2552 2553 2553 2553 2553 25	002666 002721 002725 002750 002764 002764 002790 002803 0028842 002842 002842 002892 002870 002950 002050 002950 002050 00000000	0x0000544 0x0000540 0x0000540 0x0000540 0x0000540 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000540 0x0000540 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000540 0x000055400 0x00005540 0x00005540 0x00005540 0x00005540 0x00005540 0x00005540 0x00005540 0x00005540 0x00005540 0x000055400 0x000055400 0x0000055400 0x00005540000000000	B\$10 2800 B\$10 2810 810 810 </th <th><pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 70 RCC_RCC[RCCConfig(U2) Clk_Init() + 70 RCC_RCCCCNCCONFig(U2) Clk_Init() + 70 RCC_RCCCNCCNFig(U2) RCC_RCCCCNCCNFig(U2) Clk_Init() + 70 RCC_RCCCNCCNFig(U2) RCC_RCCCNCCNFig(U2) RCC_RCCCNCNFig(U2) RCC_RCCNCNFig(U2) RCCNCNCNCNFig(U2) RCCNCNCNFig(U2) RCCNCNCNCNFig(U2) RCC</pre></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th>	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 70 RCC_RCC[RCCConfig(U2) Clk_Init() + 70 RCC_RCCCCNCCONFig(U2) Clk_Init() + 70 RCC_RCCCNCCNFig(U2) RCC_RCCCCNCCNFig(U2) Clk_Init() + 70 RCC_RCCCNCCNFig(U2) RCC_RCCCNCCNFig(U2) RCC_RCCCNCNFig(U2) RCC_RCCNCNFig(U2) RCCNCNCNCNFig(U2) RCCNCNCNFig(U2) RCCNCNCNCNFig(U2) RCC</pre>						
2368 2403 2442 2446 2446 2446 2446 2448 2559 2554 2559 2559 2559 2559 2559 2559	002686 002721 002725 002725 002764 002780 002803 002880 002842 002842 002842 002946 00295 00205 00000 00000 00000 00000000	0x0000544 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x000556 0x0000556 0x0000000000	B510 2800 B510 2810 2810 2810 2810 2810 2810 2810 2810 2810 2810 2810 2810 2810 2810 2000 844 844 844 844	<pre>RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) Clk_Init() + 84 RCC_GetPlagStatus(U6) Clk_Init() + 84 RCC_CCLCONTIG(U2) Clk_Init() + 84 RCC_CCLCONTIG(U2) Clk_Init() + 84 RCC_CCLCONTIG(U2) Clk_Init() + 89 RCC_CCLCONTIG(U2) RCC_RCLUCONTIG(U2) RC</pre>						
2368 2403 2442 2446 2446 2520 2524 2520 2524 2558 2602 2524 2558 2602 2558 2602 2558 2602 2637 2641 2758 2759 2758 2759 2758 2759 2758 2759 2758 2759 2758 2758 2759 2832 2835 2835 2835 2835 2835 2835 2906 2908	002666 002721 002725 002725 002764 002764 002780 002883 002842 0028842 0028842 002936 00107 00107 001076 0010776 0010776 0010776 0000000000	0x00005A4 0x000055A 0x00055A 0x00055C 0x00055A 0x00055C 0x00055A 0x00055C 0x00055A 0x00055C 0x00055A 0x00055C 0x0000	B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2810 B\$10 2810 2810 2810 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10 2800 B\$10	<pre>RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 60 RCC_GetFlagStatus(U8) Clk_Init() + 60 RCC_GetFlagStatus(U8) Clk_Init() + 80 RCC_ACCUSCLCONTig(U22) Clk_Init() + 88 RCC_ACCUSCLCONTig(U22) Clk_Init() + 88 RCC_ACCUSCLCONTig(U22) Clk_Init() + 88 RCC_ACCUSCLCONTig(U22) Clk_Init() + 88 RCC_ACCUSCLCONTig(U22) Clk_Init() + 88 RCC_ACCUSCLCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) RCC_ACCUSCNCONTig(U22) Clk_Init() + 88 RCC_ACCUSCNCONTig(U22) RCC_ACUSCNCONTig(U22) RCC_ACUSCNCONTig(U22) RCC_ACUSCNCONTig(U22) RCC_ACUSCNCONTig(U22) RCC_ACUSCNCONTig(U22) RCC_ACUSCNCONTig(U2</pre>						
2368 2403 2407 2442 2446 2446 2446 2446 2446 2446 2524 2559 2559 2559 2559 2559 2559 2559	002666 002713 002725 002750 002764 002764 002769 002803 002842 002842 002842 002895 002994 002995 002995 002995 002995 002995 002995 002995 002995 002950 003037 003037 003037 003035 003154 003154 003154 003154 0031260 003226 003226 003226 0032277	0x0000544 0x0000540 0x0000000000	B510 2800 B510	<pre>RCC_GetPlagStatus(U6) Clk_Tit() + 66 RCC_GetPlagStatus(U6) Clk_Tit() + 66 RCC_GetPlagStatus(U6) RCC_GetPlagStatus(U6) RCC_GetPlagStatus(U6) Clk_Tit() + 66 RCC_GetPlagStatus(U6) Clk_Tit() + 84 RCC_RCC_VCONTq(U22) Clk_Tit() + 84 RCC_RCC_RCUNCONTq(U22) Clk_Tit() + 84 RCC_RCC_RCUNCONTq(U22) Clk_Tit() + 84 RCC_RCC_VCONTq(U22) Clk_Tit() + 84 RCC_RCC_RCUNCONTq(U22) Clk_Tit() + 84 RCC_RCC_RCUNCONTq(U22) RCC_RCC_RCUNCONTq(U22) RCC_RCC_RCUNCONTq(U22) RCC_RCUNCONTQ(U22)</pre>						
2268 2403 2407 2442 2440 2524 2446 2524 2520 2524 2529 2559 2559 2559 26641 2559 26641 2715 26637 2719 2758 2709 2908 2908 2908 2908 2908 2908 2908 29	002666 002721 002725 002725 002764 002764 002769 002603 002877 0028877 0028877 0028877 0028877 0028877 002959 002959 002959 002959 002959 002959 003077 003076 003115 003150 003159 003024 003224 003224 003224 003224 003224	0x00005A4 0x000055A 0x00055A 0x00	B510 2800 B510 2810 P44F B510 2000 B510 2000 S10 S200 S200 S200 S200 S200 S00 S00 S00 S200	<pre>RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) Clk_Init() + 64 RCC_GetPlagStatus(U6) RCC_GetPlagStatus(U6) RCC_GetPlagStatus(U6) RCC_GetPlagStatus(U6) RCC_GetPlagStat</pre>						
2368 2403 2407 2442 2446 2446 2446 2446 2446 2446 2524 2559 2559 2559 2559 2559 2559 2559	002666 002713 002725 002750 002764 002764 002769 002803 002842 002842 002842 002895 002994 002995 002995 002995 002995 002995 002995 002995 002995 002950 003037 003037 003037 003035 003154 003154 003154 003154 0031260 003226 003226 003226 0032277	0x00005A4 0x00005E4 0x00005A1 0x0000	B510 2800 B510	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_RCUCONT[gU22] Clk_Init() + 90 RCC_RCUCONT[gU22] RCUCONT[gU22] RCUCONT[gU22] RCUCONT[gU22] RCUCONT[gU22]</pre>						
2368 2403 2407 2442 2447 2442 2481 2550 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2553 2552 2554 2559 2554 2657 2715 2676 2676 2676 2676 2676 2675 2675 267	002666 002721 002725 002750 002764 002764 002760 002764 002838 002838 0028438 002847 002847 002847 002847 002847 002847 002946 002946 002946 003946 003076 003159 003150 003150 003150 003150 003260 003243 003244 003243 003244 003243 003243 003244 0	0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0000554 0x0005554 0x000556 0x000554 0x000556 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x00056 0x000056 0x00056 0x00056	B510 2800 B510 2800 B510 2800 B510 2810 B510 2800 B510	RCC_GetPlagStatus(U6) Clk_Init() + 66 RCC_GetPlagStatus(U6) Clk_Init() + 76 RCC_GETUGPlagStatus(U6) Clk_Init() + 76 RCC_RCLCoonfig(U2) Clk_Init() + 76 RCC_RCLCLCoonfig(U2) Clk_Init() + 76 RCC_RCLCLCONFIg(U2) Clk_Init() + 76 RCC_RCLCLCONFIg(U2)						
2368 2403 2407 2442 2440 2442 2440 2459 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2554 2559 2755 2755 2755 2755 2755 2755 2755	002666 002721 002725 002725 002764 002764 002764 002877 0028877 0028877 0028877 0028877 0028877 0028877 0028877 0029916 0029916 0029916 0029916 0029916 0029916 0029916 0029916 0029916 0029916 00292916 003077 003077 003077 003077 003077 003077 003077 003077 003077 003224 0032279 0032375 0032375 003279 003279 003279 003279 0033279	0x0000544 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000544 0x0000	B510 2800 B510 B510 2810 2810 2810 2810 2810 2810 2810 2810 2810 2000 B510 2000 B510 2000 B510	<pre>RCC_GetFlagStatus(U8) Clk_Init() + 66 RCC_GetFlagStatus(U8) Clk_Init() + 60 RCC_GetFlagStatus(U8) Clk_Init() + 89 RCC_GetFlagStatus(U8) Clk_Init() + 80 RCC_GetFlagStatus(U8) Clk_Init() + 88 RCC_GetFlagStatus(U8) Clk_Init() + 104 RCC_GetFlagStatus(U8) Clk_Init() + 104 RCC_GETLESCONT[0](22) Clk_Init() + 104 RCC_GETLESCONT[0](22) Clk_Init(</pre>						
2368 24403 24407 24407 24442 24407 24412 2485 2520 2556 2556 25563 25563 25563 25563 25563 25563 25563 26647 26647 26647 26647 26647 26647 26647 26647 26647 26754 26352 2715 26647 2715 26647 2715 26647 27554 26647 26	002666 002711 002725 002725 002764 002764 0026838 0028838 0028838 0028877 0028838 002842 002877 0028838 002994 002995 002994 002995 002995 002995 002995 002995 002995 002995 002995 002995 002995 002995 002995 002995 002995 003115 003150 003115 003124 003226 003226 003226 003227 003226 003226 003227 003226 003277 003226 003277 003226 003277 003226 003277 003226 003277 003226 003277 003277 00326 003277 00326 003277 003377 003277 003277 003377 003277 003377 003277 003377 003777 003377	0x0000544 0x0000544 0x0000541 0x0000541 0x0000541 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000544 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000540 0x0000554 0x0000555 0x000555 0x00	B510 2800 B510 2000 B510 2000 2000 2000 2000 2000 2010 2010 2010 2010 2010 2010 2010 2010 2010 2010	RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 76 RCC_GETLAgStatus(U6) Clk_Init() + 76 RCC_ACCLCONTG(U22) Clk_Init() + 76 RCC_ACCLCONTG(U22) Clk_Init() + 104 RCC_ACLCLCONTG(U22) Clk_Init() + 104 RCA_RCLACCLCONTG(U22) Clk_Init() + 104 RLASLALATACONTG(U22)						
2368 2403 2407 2442 2440 2442 2440 2459 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2563 2552 2554 2559 2755 2755 2755 2755 2755 2755 2755	002666 002721 002725 002750 002764 002764 002780 0028838 0028838 0028877 0028838 0028877 0028877 0028877 0028877 0028877 0028877 0028916 002920 002994 002998 002998 002998 003976 003076 003076 003076 003076 003153 003016 003153 003201 003150 003150 0032243 00322420 0032243 0032243 0032243 0032243 003245 0022442 00224	0x0000544 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000544 0x000544 0x0000544 0x0	B510 2800 B510 B510 2810 2810 2810 2810 2810 2810 2810 2810 2810 2000 B510 2000 B510 2000 B510	RCC_GetPlastatus(U6) Clk_Init() + 66 RCC_RCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCL_RCLCLCONTIGU22) Clk_Ini						
2368 2403 2407 2442 2407 2442 2446 2446 2446 2559 2556 2556 2556 2556 2556 2556 255	002666 002721 002725 002725 002764 002764 002769 0026035 0026035 0026035 0028877 0028877 0028877 0028877 0028877 002895 002995 002995 002995 002995 002995 002995 002995 002995 002995 003995 003076 003995 003076 003115 00315 00315 00315 003224 003224 003224 003224 003224 003224 003224 003224 003224 003224 0032279 0032279 0032279 0032279 003305 003325 00335 003325 003325 003325 003325 003325 003325 003325 003325 003325 003325 003325 003325 003325 003325 00335 0035 00035 000505 000505 000505 000500000000	0x0000544 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000E84 0x0000544 0x0000544 0x0000544 0x0000540 0x000554 0x0005554 0x000555 0x000554 0x000555	B510 2800 B510	RCC_GetPlastatus(U6) Clk_Init() + 66 RCC_RCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCC_RCLCLCONTigU22) Clk_Init() + 100 RCL_RCLCLCONTIGU22) Clk_Ini						
2368 2403 2407 2407 2442 2407 2442 2407 2442 2407 2407	002666 002711 002725 002726 002764 002764 002786 0028838 0028838 0028838 0028838 002842 0028877 0028838 002946 002994 002995 002994 002995 002994 002995 002994 002993 003027 003155 003150 003150 003150 003150 003150 003243 003327 003372	0x0000544 0x0000540 0x0000540 0x0000541 0x0000541 0x00005541 0x00005541 0x00005541 0x00005540 0x00005550 0x00005540 0x00005550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x0000550 0x00000500 0x00000500 0x00000000	B510 2800 B510 B510 B510 B510 B510 B510 B510 B510 B510 2000 2100 2100 2100 2100 2100 2100 2100 2100 2100 2100 2100 2100	<pre>RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_GetFlagStatus(U6) Clk_Init() + 66 RCC_CGetFlagStatus(U6) Clk_Init() + 66 RCC_CGEtChConfig(U2) Clk_Init() + 106 RCC_FCLCCOnfig(U2) Clk_Init() + 106 RCC_FCLCCCOn</pre>						

11.3 Embedded Trace Buffer (ETB)

The ETB is a small, circular on-chip memory area where trace information is stored during capture. It contains the data which is normally exported immediately after it has been captured from the ETM. The buffer can be read out through the JTAG port of the device once capture has been completed. No additional special trace port is required, so that the ETB can be read via J-Link. The trace functionality via J-Link is limited by the size of the ETB. While capturing runs, the trace information in the buffer will be overwritten every time the buffer size has been reached.

J-Link ABM
SEGGER J-Link Commander U3.72c ('?' for help)
Compiled Jul 4 2007 20:17:14
DLL version V3.72c, compiled Jul 4 2007 20:17:09
Firmware: J-Link compiled Jun 14 2007 14:36:33 ARM Rev.5
Hardware: V5.30
S/N : 1
Feature(s) : RDI, FlashBP, FlashDL, JFlash, GDB
$\mathbf{UTarget} = 3.1190$
JTAG speed: 30 kHz
Info: CP15.0.0: 0x41069264: ARM, Architecure 5TEJ Info: CP15.0.1: 0x1D192192: ICache: 32kB (4*256*32), DCache: 32kB (4*256*32)
Found 2 JTAG devices, Total IRLen = 8:
Id of device #0: 0x18900F0F
Id of device #1: 0x17900F0F
Found ARM with core Id 0x17900F0F (ARM9)
ETM V1.3: 8 pairs addr.comp, 8 data comp, 16 MM decs, 4 counters, sequencer
ETB V1.0: 2048×24 bit RAM
J-Link>etb
ETB is present.
$ID_{register} (ETB[0\times00]) : 1B900F0F$
$\operatorname{RAM} \operatorname{depth} \qquad \langle \operatorname{ETB}[0 \times 01] \rangle = 00000800$
RAM width $\langle ETB[0\times02]\rangle = 00000018$
Status (ETB[0x03]) : 00000008 RAM data (ETB[0x04]) : 00CBB1B7
RAM read pointer (ETBI0X041) - 000001000
RAM write pointer (EIBI0X051) - 00000000 RAM write pointer (EIBI0X061) - 00000000
Trigger counter (ETBL0x001) = 00000000
$\begin{bmatrix} 1 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
J-Link>

The result of the limited buffer size is that not more data can be traced than the buffer can hold. Through this limitation is an ETB not in every case an fully-fledged alternative to the direct access to an ETM via J-Trace.

11.4 Flash programming

J-Link / J-Trace 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.

11.4.1 How does flash programming via J-Link / J-Trace 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. This program is called RAM code and "knows" how to program the flash; it contains an implementation of the flash programming algorithm for the particular flash. Different flash chips 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 (for example 1 * 8 bits, 1 * 16 bits, 2 * 16 bits or 32 bits). The RAM code requires data to be programmed into the flash memory. There are 2 ways of supplying this data: Data download to RAM or data download via DCC.

11.4.2 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 Ram code, the CPU is started, executing the RAM code. The RAM code, 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.

11.4.3 Data download via DCC

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

11.4.4 Available options for flash programming

There are different solutions available to program internal or external flashes connected to ARM cores using J-Link / J-Trace. The different solutions have different fields of application, but of course also some overlap.

11.4.4.1 J-Flash - Complete flash programming solution

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

11.4.4.2 RDI flash loader: Allows flash download from any RDI-compliant tool chain

RDI, (Remote debug interface) is a standard for "debug transfer agents" such as J-Link. It allows using J-Link from any RDI compliant debugger. RDI by itself does not include download to flash. To debug in flash, you need to somehow program your application program (debuggee) into the flash. You can use J-Flash for this purpose, use the flash loader supplied by the debugger company (if they supply a matching flash loader) or use the flash loader integrated in the J-Link RDI software. The RDI software as well as the RDI flash loader require licenses from SEGGER.

11.4.4.3 Flash loader of compiler / debugger vendor such as IAR

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

11.4.4.4 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 knowl-edge of the flash programming algorithm used as well as of the target system.

11.5 J-Link / J-Trace firmware

The heart of J-Link / J-Trace is a microcontroller. The firmware is the software executed by the microcontroller inside of the J-Link / J-Trace. The J-Link / J-Trace firmware sometimes needs to be updated. This firmware update is performed automatically as necessary by the JLinkARM.dll.

11.5.1 Firmware update

Every time you connect to J-Link / J-Trace, JLinkARM.dll checks if its embedded firmware is newer than the one used the J-Link / J-Trace. The DLL will then update the firmware automatically. This process takes less than 3 seconds and does not require a reboot.

It is recommended that you always use the latest version of JLinkARM.dll.

Ilink.exe	
SEGGER J-Link Commander V2.68.01. '?' for help.	
Compiled 14:02:49 on Oct 25 2005	
Updating firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5	
Replacing firmware: J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5	
Firmware update successful. CRC=5EF3	
Waiting for new firmware to boot	
DLL version V2.70a, compiled Oct 25 2005 14:02:40 Dispersed Viela version V2.604 000 14:02:40	
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5 Hardware: V5.00	
3/0 = 0.000U	
Speed set to 30 kHz	
J-Link>	
	-

In the screenshot:

- The red box identifies the new firmware.
- The green box identifies the old firmware which has been replaced.

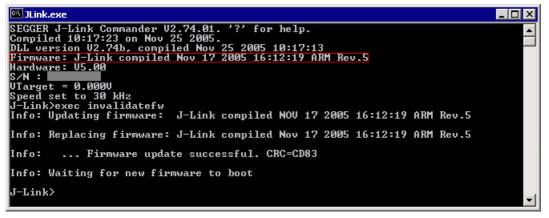
11.5.2 Invalidating the firmware

Downdating J-Link / J-Trace is not performed automatically through an old JLinkARM.dll. J-Link / J-Trace will continue using its current, newer firmware when using older versions of the JLinkARM.dll.

Note: Downdating J-Link / J-Trace is not recommended, you do it at your own risk!

Note: Note also the firmware embedded in older versions of JLinkARM.dll might not execute properly with newer hardware versions.

To downdate J-Link / J-Trace, you need to invalidate the current J-Link / J-Trace firmware, using the command <code>exec InvalidateFW</code>.



In the screenshot, the red box contains information about the formerly used J-Link / J-Trace firmware version.

Use an application (for example $_{JLink.exe}$) which uses the desired version of JLinkARM.dll. This automatically replaces the invalidated firmware with its embedded firmware.

🖾 JLink.exe	<u>- 🗆 ×</u>
SEGGER J-Link Commander V2.68.01. '?' for help.	
Compiled 14:02:49 on Oct 25 2005. Updating firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5	
Replacing firmware: J-Link compiled NOU 17 2005 16:12:19 ARM Rev.5	
Firmware update successful. CRC=5EF3 Waiting for new firmware to boot	
DLL version V2.70a, compiled Oct 25 2005 14:02:40 Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5	
Hardware: V5.00	
S∕N : VTarget = 0.000V	
Speed set to 30 kHz	
J-Link>	_

In the screenshot:

- The red box identifies the new firmware.
- The green box identifies the old firmware which has been replaced.

Chapter 12 Designing the target board for trace

This chapter describes the hardware requirements which have to be met by the target board.

12.1 Overview of high-speed board design

Failure to observe high-speed design rules when designing a target system containing an ARM Embedded Trace Macrocell (ETM) trace port can result in incorrect data being captured by J-Trace.You must give serious consideration to high-speed signals when designing the target system.

The signals coming from an ARM ETM trace port can have very fast rise and fall times, even at relatively low frequencies.

Note: These principles apply to all of the trace port signals (TRACEPKT[0:15], PIPESTAT[0:2], TRACESYNC), but special care must be taken with TRACECLK.

12.1.1 Avoiding stubs

Stubs are short pieces of track that tee off from the main track carrying the signal to, for example, a test point or a connection to an intermediate device. Stubs cause impedance discontinuities that affect signal quality and must be avoided.

Special care must therefore be taken when ETM signals are multiplexed with other pin functions and where the PCB is designed to support both functions with differing tracking requirements.

12.1.2 Minimizing Signal Skew (Balancing PCB Track Lengths)

You must attempt to match the lengths of the PCB tracks carrying all of TRACECLK, PIPESTAT, TRACESYNC, and TRACEPKT from the ASIC to the mictor connector to within approximately 0.5 inches (12.5mm) of each other. Any greater differences directly impact the setup and hold time requirements.

12.1.3 Minimizing Crosstalk

Normal high-speed design rules must be observed. For example, do not run dynamic signals parallel to each other for any significant distance, keep them spaced well apart, and use a ground plane and so forth. Particular attention must be paid to the TRACECLK signal. If in any doubt, place grounds or static signals between the TRACECLK and any other dynamic signals.

12.1.4 Using impedance matching and termination

Termination is almost certainly necessary, but there are some circumstances where it is not required. The decision is related to track length between the ASIC and the JTAG+Trace connector, see *Terminating the trace signal* on page 255 for further reference.

12.2 Terminating the trace signal

To terminate the trace signal, you can choose between three termination options:

- Matched impedance
- Series (source) termination
- DC parallel termination.

Matched impedance

Where available, the best termination scheme is to have the ASIC manufacturer match the output impedance of the driver to the impedance of the PCB track on your board. This produces the best possible signal.

Series (source) termination

This method requires a resistor fitted in series with signal. The resistor value plus the output impedance of the driver must be equal to the PCB track impedance.

DC parallel termination

This requires either a single resistor to ground, or a pull-up/pull-down combination of resistors (Thevenin termination), fitted at the end of each signal and as close as possible to the JTAG+Trace connector. If a single resistor is used, its value must be set equal to the PCB track impedance. If the pull-up/pull-down combination is used, their resistance values must be selected so that their parallel combination equals the PCB track impedance.

Caution:

At lower frequencies, parallel termination requires considerably more drive capability from the ASIC than series termination and so, in practice, DC parallel termination is rarely used.

12.2.1 Rules for series terminators

Series (source) termination is the most commonly used method. The basic rules are:

- 1. The series resistor must be placed as close as possible to the ASIC pin (less than 0.5 inches).
- 2. The value of the resistor must equal the impedance of the track minus the output impedance of the output driver. So for example, a 50 PCB track driven by an output with a 17 impedance, requires a resistor value of 33.
- 3. A source terminated signal is only valid at the end of the signal path. At any point between the source and the end of the track, the signal appears distorted because of reflections. Any device connected between the source and the end of the signal path therefore sees the distorted signal and might not operate correctly. Care must be taken not to connect devices in this way, unless the distortion does not affect device operation.

12.3 Signal requirements

The table below lists the specifications that apply to the signals as seen at the JTAG+Trace connector.

Signal	Value
Fmax	200MHz
Ts setup time (min.)	2.0ns
Th hold time (min.)	1.0ns
TRACECLK high pulse width (min.)	1.5ns
TRACECLK high pulse width (min.)	1.5ns

Table 12.1: Signal requirements

Chapter 13 Support and FAQs

This chapter contains troubleshooting tips together with solutions for common problems which might occur when using J-Link / J-Trace. There are several steps you can take before contacting support. Performing these steps can solve many problems and often eliminates the need for assistance. This chapter also contains a collection of frequently asked questions (FAQs) with answers.

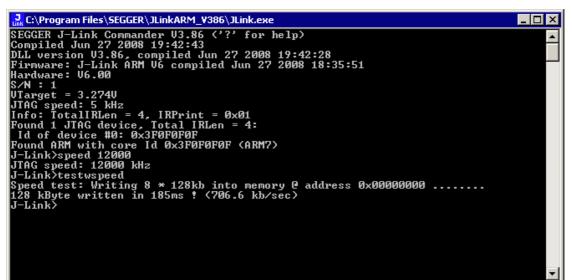
13.1 Measuring download speed

13.1.1 Test environment

JLink.exe has been used for measurement performance. The hardware consisted of:

- PC with 2.6 GHz Pentium 4, running Win2K
- USB 2.0 port
- USB 2.0 hub
- J-Link
- Target with ARM7 running at 50MHz.

Below is a screenshot of ${\tt JLink.exe}$ after the measurement has been performed.



13.2 Troubleshooting

13.2.1 General procedure

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

- 1. Close all running applications on your host system.
- 2. Disconnect the J-Link / J-Trace device from USB.
- 3. Disable power supply on the target.
- 4. Re-connect J-Link / J-Trace with the host system (attach USB cable).
- 5. Enable power supply on the target.
- 6. Try your target application again. If the problem remains continue the following procedure.
- 7. Close all running applications on your host system again.
- 8. Disconnect the J-Link / J-Trace device from USB.
- 9. Disable power supply on the target.
- 10. Re-connect J-Link / J-Trace with the host system (attach the USB cable).
- 11. Enable power supply on the target.
- 12. Start JLink.exe.
- 13. If JLink.exe displays the J-Link / J-Trace serial number and the target processor's core ID, the J-Link / J-Trace is working properly and cannot be the cause of your problem.
- 14. If the problem persists and you own an original product (not an OEM version), see section *Contacting support* on page 261.

13.2.2 Typical problem scenarios

J-Link / J-Trace LED is off

Meaning:

The USB connection does not work.

Remedy:

Check the USB connection. Try to re-initialize J-Link / J-Trace by disconnecting and reconnecting it. Make sure that the connectors are firmly attached. Check the cable connections on your J-Link / J-Trace and the host computer. If this does not solve the problem, check if your cable is defect. If the USB cable is ok, try a different host computer.

J-Link / J-Trace LED is flashing at a high frequency

Meaning:

J-Link / J-Trace could not be enumerated by the USB controller.

Most likely reasons:

- a.) Another program is already using J-Link / J-Trace.
- b.) The J-Link USB driver does not work correctly.

Remedy:

a.) Close all running applications and try to reinitialize J-Link / J-Trace by disconnecting and reconnecting it.

b.) If the LED blinks permanently, check the correct installation of the J-Link USB driver. Deinstall and reinstall the driver as shown in chapter *Setup* on page 81.

J-Link/J-Trace does not get any connection to the target

Most likely reasons:

a.) The JTAG cable is defective.

b.) The target hardware is defective.

Remedy:

Follow the steps described in *General procedure* on page 259.

13.3 Contacting support

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section *General procedure* on page 259. You may also try your J-Link / J-Trace 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 / J-Trace.

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

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

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

13.4 Frequently Asked Questions

Supported CPUs

- Q: Which CPUs are supported?
- A: J-Link / J-Trace should work with any ARM7/9 and Cortex-M3 core. For a list of supported cores, see section *Supported CPU cores* on page 39.

Using J-Link in my application

- Q: I want to write my own application and use J-Link / J-Trace. Is this possible?
- A: Yes. We offer a dedicated Software Developer Kit (SDK). See section *J-Link Software Developer Kit (SDK)* on page 78 for further information.

Using DCC with J-Link

- Q: Can I use J-Link / J-Trace to communicate with a running target via DCC?
- A: Yes. The DLL includes functions to communicate via DCC on cores which support DCC, such as ARM7/9/11, Cortex A/R series.

Read status of JTAG pins

- Q: Can J-Link / J-Trace read back the status of the JTAG pins?
- A: Yes, the status of all pins can be read. This includes the outputs of J-Link / J-Trace as well as the supply voltage, which can be useful to detect hardware problems on the target system.

J-Link support of ETM

- Q: Does J-Link support the Embedded Trace Macrocell (ETM)?
- A: No. ETM requires another connection to the ARM chip and a CPU with built-in ETM. Most current ARM7 / ARM9 chips do not have ETM built-in.

J-Link support of ETB

- Q: Does J-Link support the Embedded Trace Buffer (ETB)?
- A: Yes. J-Link supports ETB. Most current ARM7 / ARM9 chips do not have ETB builtin.

Registers on ARM 7 / ARM 9 targets

- Q: I'm running J-Link.exe in parallel to my debugger, on an ARM 7 target. I can read memory okay, but the processor registers are different. Is this normal?
- A: If memory on an ARM 7/9 target is read or written the processor registers are modified. When memory read or write operations are performed, J-Link preserves the register values before they are modified. The register values shown in the debugger's register window are the preserved ones. If now a second instance, in this case J-Link.exe, reads the processor registers, it reads the values from the hardware, which are the modified ones. This is why it shows different register values.

Chapter 14 Glossary

This chapter describes important terms used throughout this manual.

Adaptive clocking

A technique in which a clock signal is sent out by J-Link / J-Trace. J-Link / J-Trace waits for the returned clock before generating the next clock pulse. The technique allows the J-Link / J-Trace interface unit to adapt to differing signal drive capabilities and differing cable lengths.

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 is referred to as dirty data. Only write-back caches can have dirty data because a write-through cache writes data to the cache and to main memory simultaneously. See also 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.

Embedded Trace Macrocell (ETM)

ETM is additional hardware provided by debuggable ARM processors to aid debugging with trace functionality.

Embedded Trace Buffer (ETB)

ETB is a small, circular on-chip memory area where trace information is stored during capture.

EmbeddedICE

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

Halfword

A 16-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

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. Obtaining memory coherency is 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.

RESET

Abbreviation of System Reset. The electronic signal which causes the target system other than the TAP controller to be reset. This signal is also known as "nSRST" "nSYSRST", "nRST", or "nRESET" 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 state. Often, therefore, also referred to as Processor Status Register.

Also referred to as Current PSR (CPSR), to emphasize the distinction to 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 make RAM 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.

RTCK

Returned TCK. The signal which enables Adaptive Clocking.

RTOS

Real Time Operating System.

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 is running.

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 J-Link / J-Trace 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 J-Link / J-Trace 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.

Chapter 15

Literature and references

This chapter lists documents, which we think may be useful to gain deeper understanding of technical details.

Reference	Title	Comments
[ETM]	Embedded Trace Macrocell™ Architecture Specification, ARM IHI 0014J	This document defines the ETM standard, including signal protocol and physical interface. It is publicly available from ARM (www.arm.com).
[RVI]	RealView® ICE and RealView Trace User Guide, ARM DUI 0155C	This document describes ARM's realview ice emulator and requirements on the target side. It is publicly available from ARM (<i>www.arm.com</i>).

Table 15.1: Literature and References

Index

A

Adaptive clocking		
Application Program	Interface	

В

Big-endian		264
------------	--	-----

С

Cache cleaning	
Coprocessor	

D

Dirty data264	
Dynamic Linked Library (DLL)264	

Ε

Embedded Trace Buffer (ETB)	247, 264
Embedded Trace Macrocell (ETM)	243, 264
EmbeddedICE	

Η

Halfword2	264
Host2	264

Ι

ICache	. 264
ICE Extension Unit	. 264
ID	. 265
IEEE 1149.1	. 265
Image	
In-Circuit Emulator	. 265
Instruction Register	. 265
IR	. 265

J

J-Flash ARM72	2
J-Link	
Adapters237	7
Developer Pack DLL78	3
Supported chips146-147, 158-159)

J-Link ARM Flash DLL 78	3
J-Link Commander 66	5
J-Link GDB Server74	1
J-Link RDI	3
J-Link STR9 Commander 67	7
J-Link TCP/IP Server 70)
J-Mem Memory Viewer71	L
Joint Test Action Group (JTAG)265	
JTAG240	
TAP controller241	L
JTAGLoad78	3

L

Little-endian	

Μ

Memory coherency	265
Memory management unit (MMU)	265
Memory Protection Unit (MPU)	
Multi-ICE	265

Ν

nTRST	 224,	265

0

```
Open collector ......265
```

Ρ

Processor Core266 Program Status Register (PSR)266

R

RDI Support	73
Remapping	.266
Remote Debug Interface (RDI)	.266
RESET	
RTCK	.266
RTOS	.266

272

Т

-	
Tabs	. 113
TAP Controller	. 266
Target	. 266
TCK	
TDI	
TDO	, 267
Test Access Port (TAP)	. 267
Transistor-transistor logic (TTL)	. 267

W

Watchpoint	267
Word	267