J-Link / J-Trace User Guide

Software Version V4.96f

Manual Rev. 0

Date: February 4, 2015

Document: UM08001

SEGGER

A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

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

Copyright notice

Trademarks

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

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

Contact address

SEGGER Microcontroller GmbH & Co. KG

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	By	Explanation
V4.96f Rev. 0	150204	JL	Chapter "Related Software" * Section "GDB Server" Exit code description added.
V4.96 Rev. 0	141219	JL	Chapter "RTT" added. Chapter "Related Software" * Section "GDB Server" Command line option "-strict" added. Command line option "-timeout" added.
V4.90d Rev. 0	141112	NG	Chapter "Related Software" * Section "J-Link Remote Server" updated. * Section "J-Scope" updated.
V4.90c Rev. 0	140924	JL	Chapter "Related Software" * Section "JTAGLoad" updated.

Revision	Date	By	Explanation
V4.90b Rev. 1	140813	EL	Chapter "Working with J-Link and J-Trace" * Section "Connecting multiple J-Links / J-Traces to your PC" updated Chapter "J-Link software" * Section "J-Link Configurator" updated.
V4.90b Rev. 0	140813	NG	Chapter "Related Software" * Section "J-Scope" added.
V4.86 Rev. 2	140606	AG	Chapter "Device specifics" * Section "Silicon Labs - EFM32 series devices" added
V4.86 Rev. 1	140527	JL	Chapter "Related Software" * Section "GDB Server" Command line options -halt / -nohalt added. Description for GDB Server CL version added.
V4.86 Rev. 0	140519	AG	Chapter "Flash download" Section "Mentor Sourcery CodeBench" added.
V4.84 Rev. 0	140321	EL	Chapter "Working with J-Link" * Section "Virtual COM Port (VCOM) improved. Chapter "Target interfaces and adapters" * Section "Pinout for SWD + Virtual COM Port (VCOM) added."
V4.82 Rev. 1	140228	EL	Chapter "Related Software" * Section "Command line options" Extended command line option -speed. Chapter "J-Link software and documentation package" * Section "J-Link STR91x Commander" Added command line option parameter to specify a customized scan-chain. Chapter "Working with J-Link" * Section "Virtual COM Port (VCOM) added. Chapter "Setup" * Section "Getting started with J-Link and DS-5"
V4.82 Rev. 0	140218	JL	Chapter "Related Software" * Section "GDB Server" Command line option -notimout added.
V4.80f Rev. 0	140204	JL	Chapter "Related Software" * Section "GDB Server" Command line options and remote commands added.
V4.80 Rev. 1	131219	JL/ NG	Chapter "Related Software" * Section "GDB Server" Remote commands and command line options description improved. Several corrections.
V4.80 Rev. 0	131105	JL	Chapter "Related Software" * Section "GDB Server" SEGGER-specific GDB protocol extensions added.
V4.76 Rev. 3	130823	JL	Chapter "Flash Download" * Replaced references to GDB Server manual. Chapter "Working withc J-Link" * Replaced references to GDB Server manual.
V4.76 Rev. 2	130821	JL	Chapter "Related Software" * Section "GDB Server" Remote commands added.

Revision	Date	By	Explanation
V4.76 Rev. 1	130819	JL	Chapter "Related Software" * Section "SWO Viewer" Sample code updated.
V4.76 Rev. 0	130809	JL	Chapter "Related Software" * Sections reordered and updated. Chapter "Setup" * Section "Using JLinkARM.dll moved here.
V4.71b Rev. 0	130507	JL	Chapter "Related Software" * Section "SWO Viewer" Added new command line options.
V4.66 Rev. 0	130221	JL	Chapter "Introduction" * Section "Supported OS" Added Linux and Mac OSX
V4.62b Rev. 0	130219	EL	Chapter "Introduction" * Section "J-Link / J-Trace models" Clock rise and fall times updated.
V4.62 Rev. 0	130129	JL	Chapter "Introduction" * Section "J-Link / J-Trace models" Sub-section "J-link ULTRA" updated.
V4.62 Rev. 0	130124	EL	Chapter "Target interfaces and adapters" * Section "9-pin JTAG/SWD connector" Pinout description corrected.
V4.58 Rev. 1	121206	AG	Chapter "Intoduction" * Section "J-Link / J-Trace models" updated.
V4.58 Rev. 0	121126	JL	Chapter "Working with J-Link" * Section "J-Link script files" Sub-section "Executing J-Link script files" updated.
V4.56b Rev. 0	121112	JL	Chapter "Related Software" * Section "J-Link SWO Viewer" Added sub-section "Configure SWO output after device reset"
V4.56a Rev. 0	121106	JL	Chapter "Related Software" * Section "J-Link Commander" Renamed "Commander script files" to "Commander files" and "script mode" to "batch mode".
V4.56 Rev. 0	121022	AG	Renamed "J-Link TCP/IP Server" to "J-Link Remote Server".
V4.54 Rev. 1	121009	JL	Chapter "Related Software" * Section "TCP/IP Server", subsection "Tunneling Mode" added.
V4.54 Rev. 0	120913	EL	Chapter "Flash Breakpoints" * Section "Licensing" updated. Chapter "Device specifics" * Section "Freescale", subsection "Data flash support" added.
V4.53c Rev. 0	120904	EL	Chapter "Licensing" * Section "Device-based license" updated.
V4.51h Rev. 0	120717	EL	Chapter "Flash download" * Section "J-Link commander" updated. Chapter "Support and FAQs" * Section "Frequently asked questions" updated. Chapter "J-Link and J-Trace related software" * Section "J-Link Commander" updated.

Revision	Date	By	Explanation
V4.51e Rev. 1	120704	EL	Chapter "Working with J-Link" * Section "Reset strategies" updated and corrected. Added reset type 8.
V4.51e Rev. 0	120704	AG	Chapter "Device specifics" * Section "ST" updated and corrected.
V4.51b Rev. 0	120611	EL	Chapter "J-Link and J-Trace related software" * Section "SWO Viewer" added.
V4.51a Rev. 0	120606	EL	Chapter "Device specifics" * Section "ST", subsection "ETM init" for some STM32 devices added.. * Section "Texas Instruments" updated. 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.
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" * Section "J-Link / J-Trace models" corrected.
V4.26 Rev. 0	110427	KN	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.

Revision	Date	By	Explanation
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
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.

Revision	Date	By	Explanation
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.
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.

Revision	Date	By	Explanation
62	090211	AG	Chapter "Working with J-Link and J-Trace" * 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	TQ	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.
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.

Revision	Date	By	Explanation
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.
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.

Revision	Date	By	Explanation
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 "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	TQ	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.
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.
19	070226	SK	Chapter "J-Link / J-Trace related software" updated and reorganized. Chapter "Hardware" "List of OEM products" updated
18	070221	SK	Chapter "Device specifics" added Subchapter "Command strings" added

Revision	Date	By	Explanation
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	OO	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	TQ	Subchapter 4.7 "Using DCC for memory reads" added.
13	060711	OO	Subchapter 5.2.2: Corrected JTAG+Trace connector pinout table.
12	060628	OO	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
9	060324	OO	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	OO	Chapter "Related Software": Added JLinkARM.dll. Screenshots updated.
7	051208	OO	Chapter Working with J-Link: Sketch added.
6	051118	OO	Chapter Working with J-Link: "Connecting multiple J-Links to your PC" added. Chapter Working with J-Link: "Multi core debugging" added. Chapter Background information: "J-Link firmware" added.
5	051103	TQ	Chapter Setup: "JTAG Speed" added.
4	051025	OO	Chapter Background information: "Flash programming" added. Chapter Setup: "Scan chain configuration" added. Some smaller changes.
3	051021	TQ	Performance values updated.
2	051011	TQ	Chapter "Working with J-Link" added.
1	050818	TW	Initial version.

About this document

Assumptions

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

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

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

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

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

Table 1.1: Typographic conventions

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

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

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet 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 micro controllers, as well as J-Link, a JTAG emulator to assist in development, debugging and production, which has rapidly become the industry standard for debug access to ARM cores.

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

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

EMBEDDED SOFTWARE (Middleware)

emWin

Graphics software and GUI
emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display.

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.

embOS/IP

TCP/IP stack
embOS/IP a high-performance TCP/IP stack that has been optimized for speed, versatility and a small memory footprint.

emFile

File system
emFile is an embedded file system with FAT12, FAT16 and FAT32 support. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.

USB-Stack

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

SEGGER TOOLS

Flasher

Flash programmer
Flash Programming tool primarily for micro controllers.

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 1.1:

Table of Contents

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 23.

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 25.

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
- Windows 8
- Windows 8 x64
- Linux
- Mac OSX 10.5 and higher

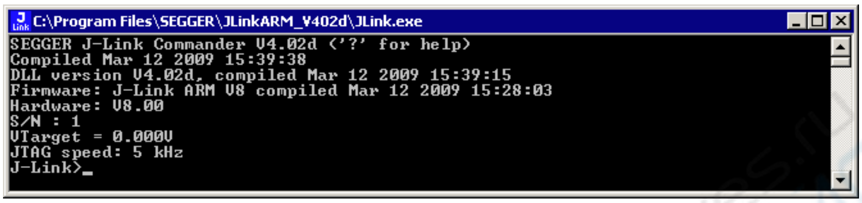
1.3 J-Link / J-Trace models

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
- J-Link PLUS
- J-Link PRO
- J-Link ULTRA
- J-Link ULTRA+
- 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 `JLink.exe`. As part of the initial message, the hardware version is displayed.



```
SEGGER J-Link Commander V4.02d ('?' 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: V8.00
S/N : 1
VTarget = 0.000V
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 Plus	J-Link ULTRA+	J-Link Pro	J-Trace for Cortex-M	J-Trace
USB	yes	yes	yes	yes	yes	yes
Ethernet	no	no	no	yes	no	no
Supported cores	ARM7/9/11, Cortex-A5/A8/A9/R4, Cortex-M0/M0+/M1/M3/M4, Renesas RX				Tracing: Cortex-M3/M4 No tracing: ARM7/9/11, Cortex-M0/M0+/M1 Cortex-A5/A8/A9/R4	ARM 7/9
JTAG	yes	yes	yes	yes	yes	yes
SWD	yes	yes	yes	yes	yes	no
SWO	yes	yes	yes	yes	yes	no
ETM Trace	no	no	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 PLUS	J-Link ULTRA+	J-Link Pro	J-Trace for Cortex-M	J-Trace
J-Flash	yes(opt)	yes	yes	yes	yes	yes
Flash breakpoints[2]	yes(opt)	yes	yes	yes	yes	yes
Flash download[1]	yes	yes	yes	yes	yes	yes
GDB Server	yes	yes	yes	yes	yes	yes
RDI	yes(opt)	yes	yes	yes	yes	yes

[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 191.

[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 205.

1.3.2 J-Link

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 23. 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 1 MBytes/second*
- Debug interface (JTAG/SWD/...) speed up to 15 MHz
- RDI interface available, which allows using J-Link with RDI compliant software

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

1.3.2.2 Specifications

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

General	
Supported OS	For a complete list of all operating systems which are supported, please refer to *Supported OS* on page 23.
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} <= 10% of V_{IF}
For the whole target voltage range (1.2V <= V_{IF} <= 5V)	
LOW level input voltage (V_{IL})	V_{IL} <= 40% of V_{IF}

Table 1.1: J-Link specifications

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} <= 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. 7.5 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} <= 3ns
Clock fall time (T_{fc})	T_{fc} <= 3ns

Table 1.1: J-Link specifications

1.3.2.3 Hardware versions

Versions 1-4 (Obsolete)

Obsolete.

Version 5.0 (Obsolete)

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 (Obsolete)

Identical to version 5.0 with the following exception:

- Target interface: RESET is open drain

Version 5.3 (Obsolete)

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 (Obsolete)

Identical to version 5.3 with the following exception:

- Supports 5V target interfaces.

Version 6.0 (Obsolete)

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 (Obsolete)

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.

Version 9.1

- New design based on STM32F205.

Version 9.2

Identical to version 9.1 with the following exception:

- Pin 1 (VTref) is used for measuring target reference voltage only. Buffers on J-Link side are no longer powered through this pin but via the J-Link internal voltage supplied via USB.

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
- Download speed up to 3 MByte/second*
- Debug interface (JTAG/SWD/...) speed up to 15 MHz
- Serial Wire Debug (SWD) supported
- Serial Wire Viewer (SWV) supported
- SWO sampling frequencies up to 100 MHz
- Serial Wire Output (SWO) supported
- Target power can be supplied
- Target power consumption can be measured with high accuracy.

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

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 *Supported OS* on page 23.
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	20-pin J-Link debug interface connector
JTAG/SWD Interface, Electrical	
Target interface voltage (V_{IF})	1.8V ... 5V
Target supply voltage	4.5V ... 5V
Target supply current	Max. 300mA

Table 1.2: J-link ULTRA specifications

Reset Type	Open drain. Can be pulled low or tristated.
Reset low level output voltage (V_{OL})	V_{OL} <= 10% of V_{IF}
For the whole target voltage range (1.8V <= V_{IF} <= 5V)	
LOW level input voltage (V_{IL})	V_{IL} <= 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} <= 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. 100 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} <= 3ns
Clock fall time (T_{fc})	T_{fc} <= 3ns
Analog power measurement interface	
Sampling frequency	50 kHz
Resolution	1 mA

Table 1.2: J-link ULTRA specifications

1.3.4 J-Link 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, Linux or Mac OS X. For a complete list of all operating systems which are supported, please refer to Supported OS on page 19. 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
- 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)
- Serial Wire Debug supported
- Serial Wire Viewer supported
- Download speed up to 3 MByte/second
- Comes with built-in licenses for: Unlimited number of breakpoints in flash (FlashBP), J-Link GDBServer, J-Link RDI and J-Flash (production programming software).
- Embedded Trace Buffer (ETB) support

- Galvanic isolation from host via Ethernet

1.3.4.2 Hardware versions

Version 1.1

Compatible to J-Link.

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

Version 4

- New design based on STM32F407 + FPGA (Cyclone IV)Version 4.3

Identical to version 4 with the following exception:

- Pin 1 (VTref) is used for measuring target reference voltage only. Buffers on J-Link side are no longer powered through this pin but via the J-Link internal voltage supplied via USB.

1.3.5 J-Link Lite ARM

J-Link Lite ARM is a fully functional OEM-version of J-Link. If you are selling evaluation-boards, J-Link Lite ARM 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
- Supports any ARM7/9/11, Cortex-A5/A8/A9, Cortex-M0/M0+/M1/M3/M4, Cortex-R4/R5 core
- 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)

1.3.5.2 Specifications

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

General	
Supported OS	For a complete list of all operating systems which are supported, please refer to *Supported OS* on page 23.
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
Mechanical	
USB interface	USB 2.0, full speed
Target interface	JTAG 20-pin (14-pin adapter available)
JTAG/SWD Interface, Electrical	

Table 1.3: J-Link Lite specifications

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 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 1.3: J-Link 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/M0+/M1/M3/M4 core supported
- JTAG clock up to 4 MHz
- SWD, SWO supported
- Flash download into supported MCUs
- Standard 9- or 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 systems which are supported, please refer to *Supported OS* on page 23.
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 Interface, Electrical	

Table 1.4: J-Link Lite Cortex-M specifications

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 1.4: J-Link Lite Cortex-M specifications

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 systems which are supported, please refer to *Supported OS* on page 23.
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

Table 1.5: J-Trace specifications

JTAG/SWD Interface, Electrical	
Power Supply	USB powered < 300mA
Supported Target interface voltage	3.0 - 3.6 V (5V adapter available)

Table 1.5: 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 1.6: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 334.

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 systems 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 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
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. 3ns

Table 1.7: J-Trace for Cortex-M3 specifications

Clock fall time (T_{fc})	Max. 3ns
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_{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

Table 1.7: 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 1.8: 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

1.3.9 Flasher ARM

Flasher ARM is a programming tool for microcontrollers with on-chip 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.

General	
Supported OS	For a complete list of all operating systems 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_{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 1.9: 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 load of 10 kOhm	Max. 0.5V
HIGH level output voltage (V_{OH}) with a load of 10 kOhm	Min. 2.85V

Table 1.9: 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
- Remote 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.

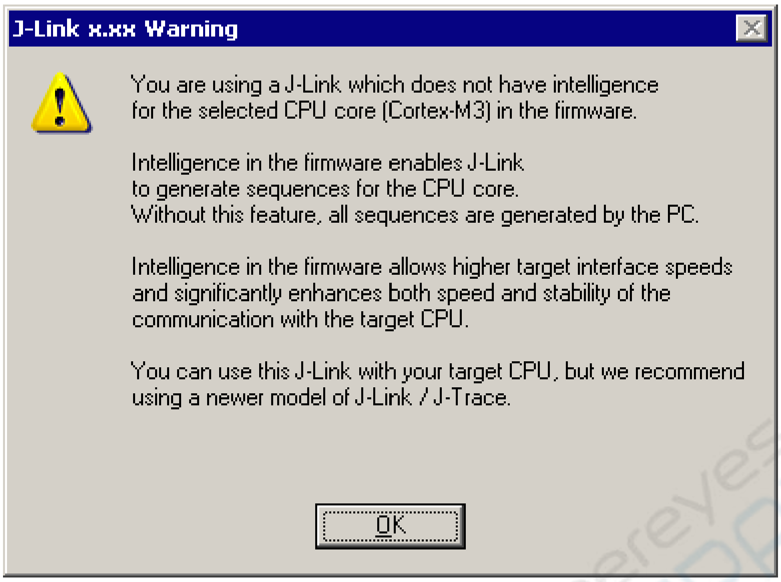
- ARM7TDMI (Rev 1)
- ARM7TDMI (Rev 3)
- ARM7TDMI-S (Rev 4)
- ARM720T
- ARM920T
- ARM922T
- ARM926EJ-S
- ARM946E-S
- ARM966E-S
- ARM1136JF-S
- ARM1136J-S
- ARM1156T2-S
- ARM1156T2F-S
- ARM1176JZ-S
- ARM1176JZF
- ARM1176JZF-S
- Cortex-A5
- Cortex-A8
- Cortex-A9
- Cortex-M0
- Cortex-M1
- Cortex-M3
- Cortex-M4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

1.6 Built-in intelligence for supported CPU-cores

In general, there are 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.



You are using a J-Link which does not have intelligence for the selected CPU core (Cortex-M3) in the firmware.

Intelligence in the firmware enables J-Link to generate sequences for the CPU core. Without this feature, all sequences are generated by the PC.

Intelligence in the firmware allows higher target interface speeds and significantly enhances both speed and stability of the communication with the target CPU.

You can use this J-Link with your target CPU, but we recommend using a newer model of J-Link / J-Trace.

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 DLL-side: 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	Cortex-M		Renesas RX600
		JTAG	JTAG	JTAG	JTAG	SWD	JTAG
J-Link	8	✓	✓	✓	✓	✓	✓
J-Link Pro	3	✓	✓	✓	✓	✓	✓
J-link ULTRA	1	✓	✓	✓	✓	✓	✓
J-Link Lite ARM	8	✓	✓	✓	✓	✓	✗
J-Link Lite Cortex-M	8	✗	✗	✗	✓	✓	✗
J-Link Lite RX	8	✗	✗	✗	✗	✗	✓
J-Trace ARM	1	✓	✗	✗	✗	✗	✗
J-Trace for Cortex-M	3	✗	✗	✗	✓	✓	✗

Table 1.10: 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	Cortex-M		Renesas RX600
		JTAG	JTAG	JTAG	JTAG	SWD	JTAG
J-Link	3	✗	✗	✗	✗	not supported	✗
J-Link	4	✗	✗	✗	✗	not supported	✗
J-Link	5	✓	✗	✗	✗	not supported	✗
J-Link	6	✓	✗	✗	✗	✓	✗
J-Link	7	✓	✗	✗	✗	✓	✗
J-Link Pro	1	✓	✓	✓	✓	✓	✓
J-Trace for Cortex-M	1	✗	✗	✓	✓	✓	✗

Table 1.11: 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 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 Components requiring a license

The following programs/features require a full-featured J-Link (PLUS, ULTRA+, PRO, J-Trace) or an additional license for the J-Link base model:

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

2.2 License types

For each of the software components which require an additional license, there are different types of licenses which are explained in the following.

Built-in License

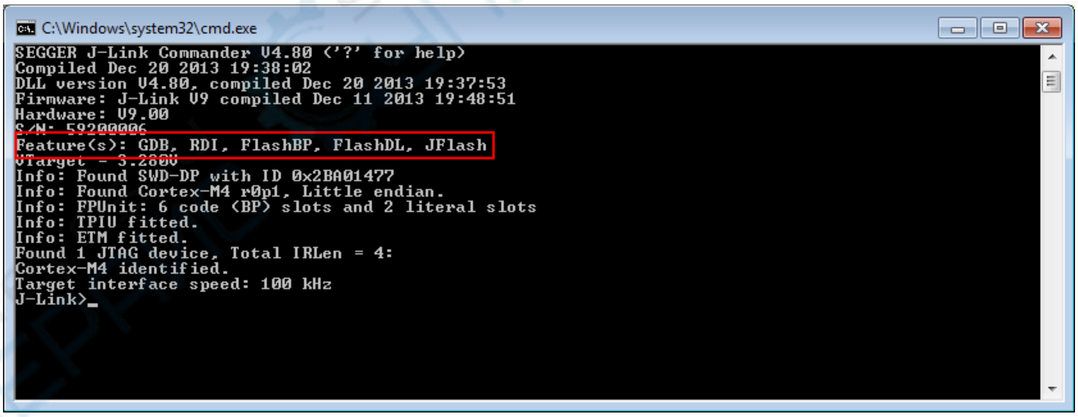
This type of license is easiest to use. The customer does not need to deal with a license key. The software automatically finds out that the connected J-Link contains the built-in license(s). The license is burned into the J-Link debug probe and can be used on any computer, the J-Link is connected to. This type of license applies to the J-Link PLUS, J-Link ULTRA+ and J-Link Pro.

Key-based license

This type of license is used if you already have a J-Link, but order a license for a J-Link software component at a later time. 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 207. 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. This means that for example if you want to use the unlimited breakpoints in flash memory feature with different J-Links which do not come with a built-in license, a license is needed for each of these J-Links.

2.2.1 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.



```
SEGGER J-Link Commander V4.80 ('?' for help)
Compiled Dec 20 2013 19:38:02
DLL version V4.80, compiled Dec 20 2013 19:37:53
Firmware: J-Link V9 compiled Dec 11 2013 19:48:51
Hardware: V9.00
S/N: 59200006
Feature(s): GDB, RDI, FlashBP, FlashDL, JFlash
VTarget = 3.280V
Info: Found SWD-DP with ID 0x2BA01477
Info: Found Cortex-M4 r0p1, Little endian.
Info: FPUnit: 6 code (BP) slots and 2 literal slots
Info: TPIU fitted.
Info: ETM fitted.
Found 1 JTAG device, Total IRLen = 4:
Cortex-M4 identified.
Target interface speed: 100 kHz
J-Link>_
```

The J-Link PLUS in the example above contains licenses for all features. Note that GDB and FlashDL feature are no longer required.

2.2.2 Key-based license

When using a key-based license, a license key is required in order to enable the feature. License keys can be added via the J-Link License Manager. How to enter a license via the license manager is described in *Licensing* on page 207. 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 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.

2.3.1 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.

2.4 Original SEGGER products

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

2.4.1 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 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 flash download.

2.4.2 J-Link PLUS

J-Link PLUS is a USB powered JTAG emulator supporting a large number of CPU cores.

Based on a 32-bit RISC CPU, it can communicate at high speed with the supported target CPUs. J-Link is used around the world in tens of thousand places for development and production (flash programming) purposes.

J-Link is supported by all major IDEs such as IAR EWARM, Keil MDK, Rowley CrossWorks, Atollic TrueSTUDIO, IAR EWRX, Renesas HEW, Renesas e2studio, and many others.

Licenses

Comes with liicenses for: Unlimited breakpoints in flash memory (Unlimited Flash Breakpoints), RDI and J-Flash

2.4.3 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, Windows XP, Windows 2003, Windows Vista or Windows 7.
J-link ULTRA has a built-in 20-pin JTAG/SWD connector.

Licenses

Unlimited breakpoints in flash memory (Unlimited Flash Breakpoints), RDI and J-Flash

2.4.4 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 FlashDL, FlashBP, RDI, J-Link GDB Server and J-Flash.

2.4.5 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.

2.4.6 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.

2.4.7 Flasher ARM

Flasher ARM is a programming tool for microcontrollers with on-chip 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.

Licenses

Comes with built-in licenses for flash download and J-Flash.

2.4.8 Flasher RX

Flasher RX is a programming tool for Renesas RX600 series microcontrollers with on-chip or external flash memory and Renesas RX core. Flasher RX is designed for programming flash targets with the J-Flash software or stand-alone. In addition to that Flasher RX has all of the J-Link RX functionality. Flasher RX connects via Ethernet, USB or via RS232 interface to a PC, running Microsoft Windows 2000, Windows XP, Windows 2003, Windows Vista or Windows 7.

Flasher RX itself has a built-in 20-pin JTAG connector but is shipped with an 14-pin adapter for Renesas RX devices.

Licenses

Comes with built-in licenses for flash download and J-Flash.

2.4.9 Flasher PPC

Flasher ARM is a programming tool for microcontrollers with on-chip 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.

Licenses

Comes with built-in licenses for flash download and J-Flash.

2.5 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.

2.5.1 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 FlashDL and FlashBP are included. Other licenses can be added.

2.5.2 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.

2.5.3 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.

2.5.4 IAR: J-Link / J-Link KS

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.

2.5.5 IAR: J-Link Lite

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.

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

2.5.6 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.

2.5.7 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.

2.5.8 SEGGER: J-Link Lite ARM

J-Link Lite ARM is a fully functional OEM-version of SEGGER J-Link. If you are selling evaluation-boards, J-Link Lite ARM 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 Lite ARM 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.

2.6 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.

2.7 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 software and documentation package

This chapter describes the contents of the J-Link software and documentation package which can be downloaded from *www.segger.com*.

3.1 Software overview

The J-Link software and documentation package, which is available for download from *http://www.segger.com/jlink-software.html* includes some applications to be used with J-Link. It also comes with USB-drivers for J-Link and documentations in pdf format.

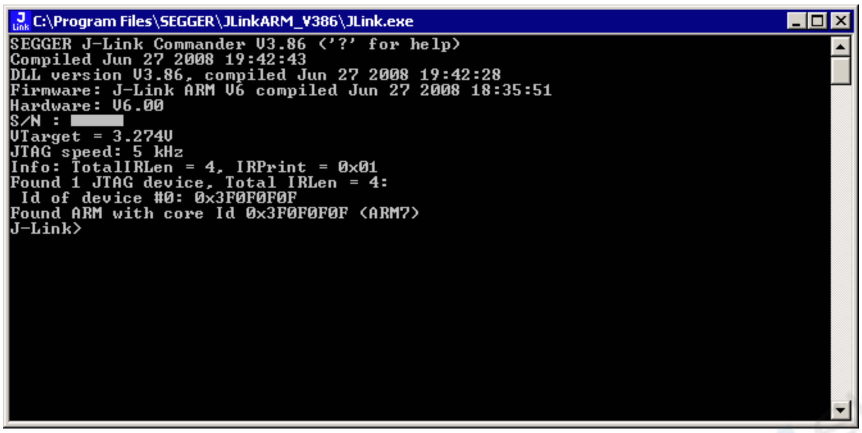
Software	Description
JLink Commander	Command-line tool with basic functionality for target analysis.
J-Link GDB Server	The J-Link GDB Server is a server connecting to the GNU Debugger (GDB) via TCP/IP. It is required for toolchains using the GDB protocol to connect to J-Link.
J-Link GDB Server command line version	Command line version of the J-Link GDB Server. Same functionality as the GUI version.
J-Link Remote Server	Utility which provides the possibility to use J-Link / J-Trace remotely via TCP/IP.
J-Mem Memory Viewer	Target memory viewer. Shows the memory content of a running target and allows editing as well.
J-Flash[a]	Stand-alone flash programming application. For more information about J-Flash please refer to *J-Flash ARM User's Guide (UM08003)*.
J-Link SWO Viewer	Free-of-charge utility for J-Link. Displays the terminal output of the target using the SWO pin. Can be used in parallel with a debugger or stand-alone.
J-Scope	J-Scope is a free-of-charge software to analyze and visualize data on a microcontroller in real-time, while the target is running
J-Link SWO Analyzer	Command line tool that analyzes SWO RAW output and stores it into a file.
JTAGLoad	Command line tool that opens an `svf` file and sends the data in it via J-Link / J-Trace to the target.
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 *J-Link Configurator* on page 131.
RDI support[a]	Provides Remote Debug Interface (RDI) support. This allows the user to use J-Link with any RDI-compliant debugger.
Processor specific tools	Free command-line tools for handling specific processors. Included are: STR9 Commander and STM32 Unlock.

Table 3.1: J-Link / J-Trace related software

[a] Full-featured J-Link (PLUS, PRO, ULTRA+) or an additional license for J-Link base model required.

3.2 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 target CPU, as well as for simple analysis of the target system. It permits some simple commands, such as memory dump, halt, step, go etc. to verify the target connection.



```
SEGGER J-Link Commander V3.86 ('?' for help)
Compiled Jun 27 2008 19:42:43
DLL version V3.86, compiled Jun 27 2008 19:42:28
Firmware: J-Link ARM V6 compiled Jun 27 2008 18:35:51
Hardware: V6.00
S/N : [redacted]
VTarget = 3.274V
JTAG speed: 5 kHz
Info: TotalIRLen = 4, IRPrint = 0x01
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x3F0F0F0F
Found ARM with core Id 0x3F0F0F0F (ARM7)
J-Link>
```

3.2.1 Command line options

J-Link Commander can be started with different command line options for test and automation purposes. In the following, the command line options which are available for J-Link Commander are explained.

-CommandFile

Selects a command file and starts J-Link Commander in batch mode. The batch mode of J-Link Commander is similar to the execution of a batch file. The command file is parsed line by line and one command is executed at a time.

Syntax

```
-CommanderScript <CommandFilePath>
```

Example

See *Using command files* on page 64

-Device

Pre-selects the device J-Link Commander shall connect to. For some devices, J-Link already needs to know the device at the time of connecting, since special handling is required for some of them. For a list of all supported device names, please refer to *http://www.segger.com/jlink_supported_devices.html.*

Syntax

```
-Device <DeviceName>
```

Example

```
JLink.exe -Device STM32F103ZE
```

-If

Selects the target interface J-Link shall use to connect to the target. By default, J-Link Commander first tries to connect to the target using the target interface which is currently selected in the J-Link firmware. If connecting fails, J-Link Commander goes through all target interfaces supported by the connected J-Link and tries to connect to the device.

Syntax

```
-If <TargetInterface>
```

Example

```
JLink.exe -If SWD
```

Additional information

Currently, the following target interfaces are supported:

- JTAG
- SWD

-IP

Selects IP as host interface to connect to J-Link. Default host interface is USB.

Syntax

```
-IP <IPAddr>
```

Example

```
JLink.exe -IP 192.168.1.17
```

Additional information

To select from a list of all available emulators on Ethernet, please use `*` as `<IPAddr>`.

-SelectEmuBySN

Connect to a J-Link with a specific serial number via USB. Useful if multiple J-Links are connected to the same PC and multiple instances of J-Link Commander shall run and each connects to another J-Link.

Syntax

```
-SelectEmuBySN <SerialNo>
```

Example

```
JLink.exe -SelectEmuBySN 580011111
```

-Speed

Starts J-Link Commander with a given initial speed. Available parameters are "adaptive", "auto" or a freely selectable integer value in kHz. It is recommended to use either a fixed speed or, if it is available on the target, adaptive speeds. Default interface speed is 100kHz.

Syntax

```
-Speed <Speed_kHz>
```

Example

```
JLink.exe -Speed 4000
```

3.2.2 Using command files

J-Link commander can also be used in batch mode which allows the user to use J-Link commander for batch processing and without user interaction. Please do not confuse command file with J-Link script files (please refer to *J-Link script files* on page 169 for more information about J-Link script files). When using J-Link commander in batch mode, the path to a command file is passed to it. The syntax in the command file is the same as when using regular commands in J-Link commander (one line per command). SEGGER recommends to always pass the device name via command line option due some devices need special handling on connect/reset in order to guarantee proper function.

Example

```
JLink.exe -device STM32F103ZE -CommanderScript C:\CommandFile.jlink
```

Contents of CommandFile.jlink:

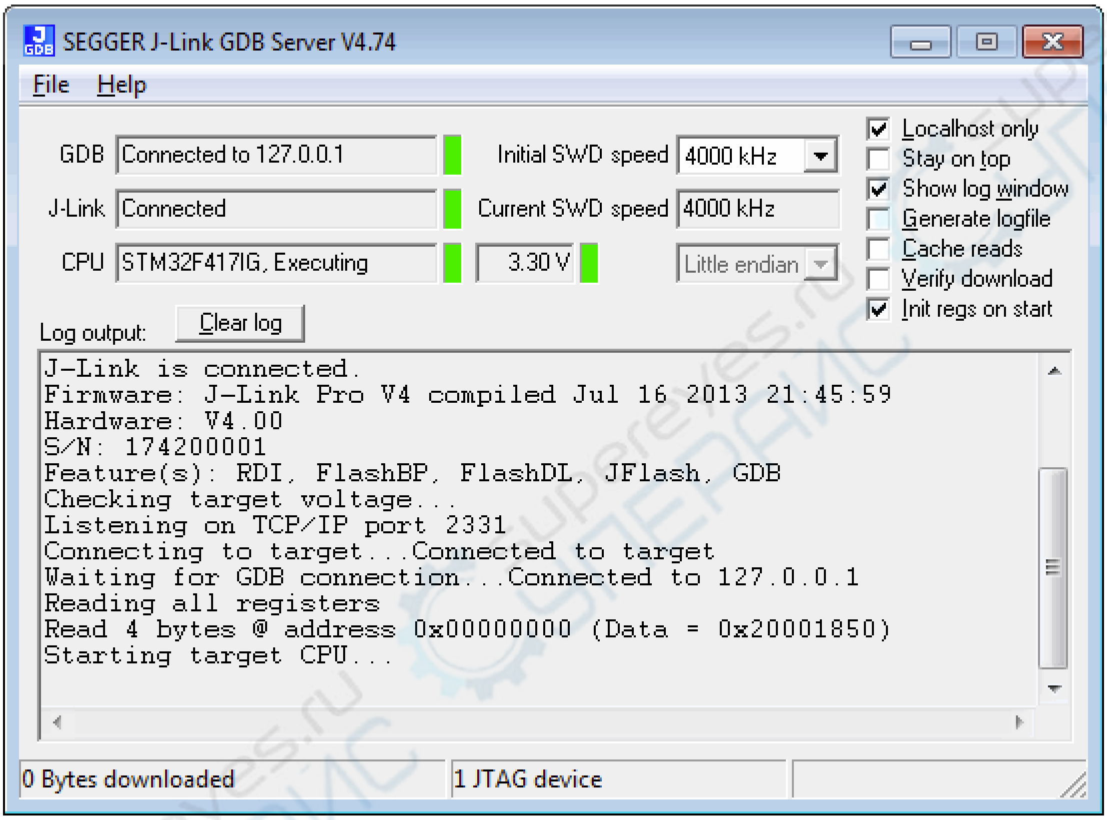
```
si 1
speed 4000
r
h
loadbin C:\firmware.bin,0x08000000
```

3.3 J-Link GDB Server

The GNU Project Debugger (GDB) is a freely available and open source debugger. It can be used in command line mode, but is also integrated in many IDEs like emIDE or Eclipse.

J-Link GDB Server is a remote server for GDB making it possible for GDB to connect to and communicate with the target device via J-Link. GDB Server and GDB communicate via a TCP/IP connection, using the standard GDB remote protocol. GDB Server receives the GDB commands, does the J-Link communication and replies with the answer to GDB.

With J-Link GDB Server debugging in ROM and Flash of the target device is possible and the Unlimited Flash Breakpoints can be used.
It also comes with some functionality not directly implemented in the GDB. These can be accessed via monitor commands, sent directly via GDB, too.



GDB: Connected to 127.0.0.1 🟩
J-Link: Connected 🟩
CPU: STM32F417IG, Executing 🟩 3.30 V 🟩
Initial SWD speed: 4000 kHz
Current SWD speed: 4000 kHz
Little endian

☑ Localhost only
□ Stay on top
☑ Show log window
□ Generate logfile
□ Cache reads
□ Verify download
☑ Init regs on start

```
J-Link is connected.
Firmware: J-Link Pro V4 compiled Jul 16 2013 21:45:59
Hardware: V4.00
S/N: 174200001
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...Connected to 127.0.0.1
Reading all registers
Read 4 bytes @ address 0x00000000 (Data = 0x20001850)
Starting target CPU...
```

0 Bytes downloaded | 1 JTAG device

The GNU Project Debugger (GDB) is a freely available debugger, distributed under the terms of the GPL. 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.

3.3.1 J-Link GDB Server CL (Windows, Lunux, Mac)

J-Link GDB Server CL is a commandline-only version of the GDB Server.
The command line version is part of the Software and Documentation Package and also included in the Linux and MAC versions.

Except for the missing GUI, J-Link GDB Server CL is identical to the normal version.
All sub-chapters apply to the command line version, too.

3.3.2 Debugging with J-Link GDB Server

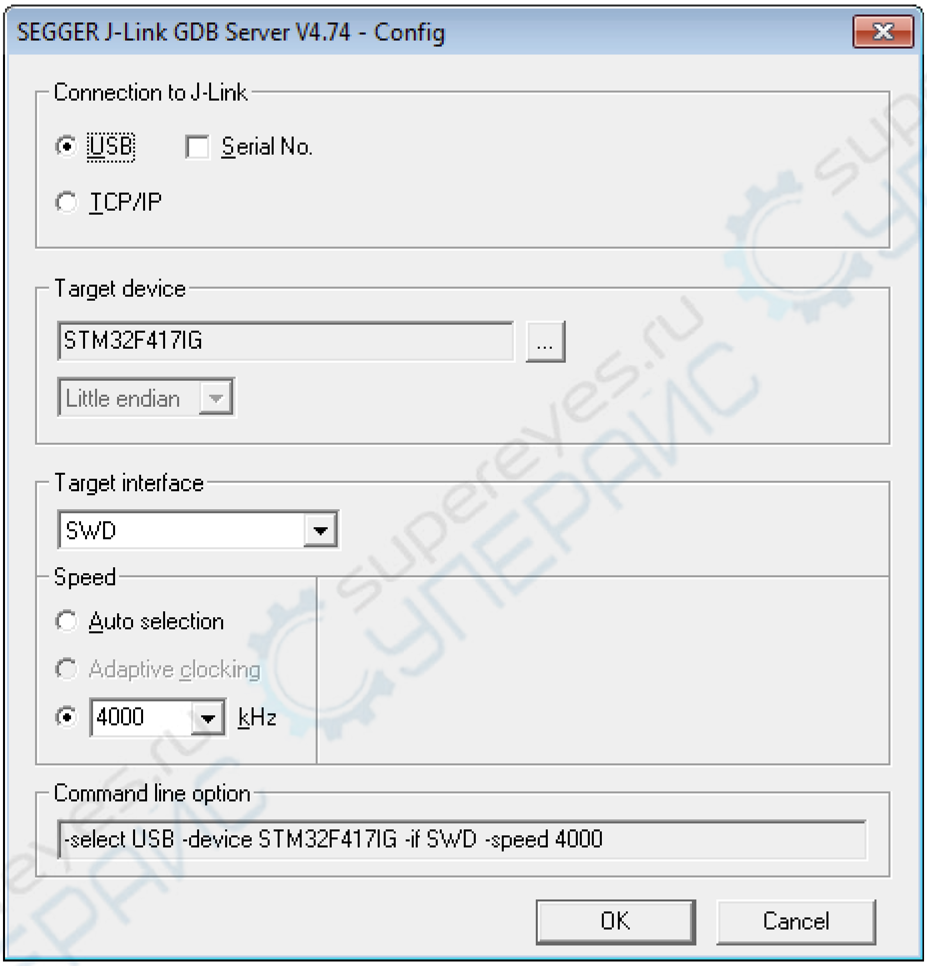
With J-Link GDB Server programs can be debugged via GDB directly on the target device like a normal application. The application can be loaded into RAM or flash of the device.

Before starting GDB Server make sure a J-Link and the target device are connected.

3.3.2.1 Setting up GDB Server GUI version

The GUI version of GDB Server is part of the Windows J-Link Software Package (JLinkGDBServer.exe).

When starting GDB Server a configuration dialog pops up letting you select the needed configurations to connect to J-Link and the target.



SEGGER J-Link GDB Server V4.74 - Config

Connection to J-Link

● USB □ Serial No.

○ TCP/IP

Target device: STM32F417IG

Little endian

Target interface: SWD

Speed:

○ Auto selection

○ Adaptive clocking

● 4000 kHz

Command line option: -select USB -device STM32F417IG -if SWD -speed 4000

All configurations can optionally be given in the command line options.

Note: To make sure the connection to the target device can be established correctly, the device, as well as the interface and interface speed have to be given on start of GDB Server, either via command line options or the configuration dialog.
If the target device option (`-device`) is given, the configuration dialog will not pop up.

3.3.2.2 Setting up GDB Server CL version

The command line version of GDB Server is part of the J-Link Software Package for all supported platforms.
On Windows its name is JLinkGDBServerCL.exe, on Linux and Mac it is JLinkGDBServer.

Starting GDB Server on Windows

To start GDB Server CL on Windows, open the 'Run' prompt (Windows-R) or a command terminal (cmd) and enter <PathToJLinkSoftware>\JLinkGDBServerCL.exe <CommandLineOptions>.

Starting GDB Server on Linux / Mac

To start GDB Server CL on Linux / Mac, open a terminal and call JLinkGDBServer <CommandLineOptions>

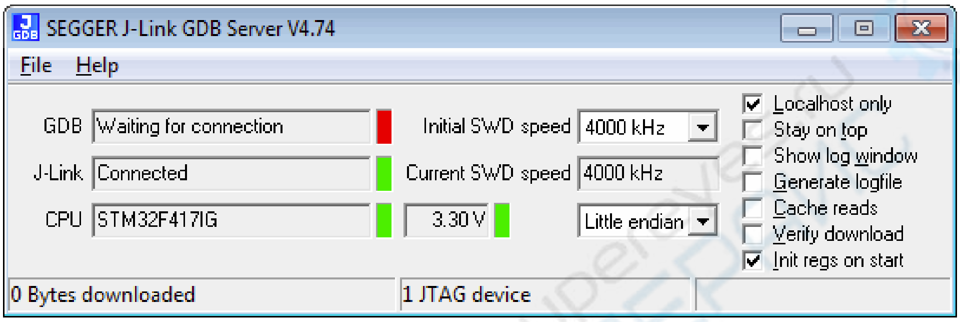
Command Line Options

When using GDB Server CL, at least the mandatory command line options have to be given. Additional command line options can be given to change the default behavior of GDB Server.

For more information about the available command line options, please refer to *Command line options* on page 87.

3.3.2.3 GDB Server user interface

The J-Link GDB Server's user interface shows information about the debugging process and the target and allows some in progress configurations.



SEGGER J-Link GDB Server V4.74

File Help

GDB: Waiting for connection 🟥

J-Link: Connected 🟩

CPU: STM32F417IG 🟩 3.30 V 🟩

Initial SWD speed: 4000 kHz

Current SWD speed: 4000 kHz

Little endian

☑ Localhost only
☐ Stay on top
☐ Show log window
☐ Generate logfile
☐ Cache reads
☐ Verify download
☑ Init regs on start

0 Bytes downloaded | 1 JTAG device

It shows following information:

- IP address of host running debugger.
- Connection status of J-Link.
- Information about the target core.
- Measured target voltage.
- Bytes that have been downloaded
- Status of target.
- Log output of the GDB Server (optional, if `Show log window` is checked).
- Initial and current target interface speed.
- Target endianess.

These configurations can be made from inside GDB Server:

- Localhost only: If checked only connections from 127.0.0.1 are accepted.
- Stay on top
- Show log window
- Generate logfile: If checked, a log file with the GDB <-> GDB Server <-> J-Link communication will be created.
- Verify download: If checked, the memory on the target will be verified after download.
- Init regs on start: If checked, the register values of the target will be set to a reasonable value before on start of GDB Server.

3.3.2.4 Running GDB from different programs

We assume that you already have a solid knowledge of the software tools used for building your application (assembler, linker, C compiler) and especially the debugger and the debugger frontend of your choice. We do not answer questions about how to install and use the chosen toolchain.

GDB is included in many IDEs and most commonly used in connection with the GCC compiler toolchain. This chapter shows how to configure some programs to use GDB and connect to GDB Server. For more information about any program using GDB, please refer to its user manual.

emIDE

emIDE is a full-featured, free and open source IDE for embedded development including support for debugging with J-Link.

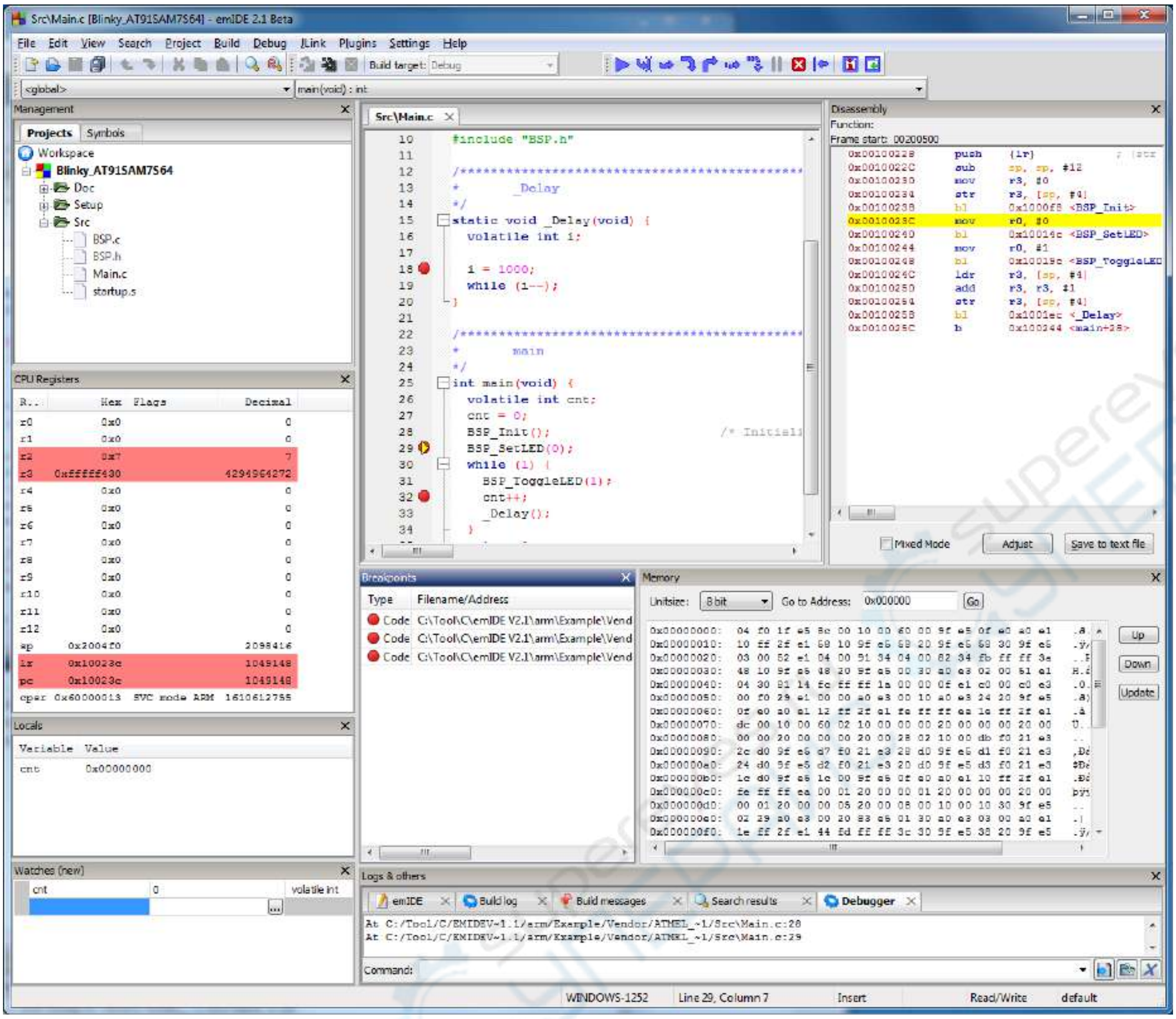
To connect to GDB Server with emIDE, the GDB Server configurations need to be set in the project options at Project -> Properties... -> Debugger.
Select the target device you are using, the target connection, endianess and speed and enter the additional GDB start commands.
The typically required GDB commands are:

```
#Initially reset the target
monitor reset
#Load the application
load
```

Other commands to set up the target (e.g. Set PC to RAM, initialize external flashes) can be entered here, too.

emIDE will automatically start GDB Server on start of the debug session. If it does not, or an older version of GDB Server start, in emIDE click on JLink -> Run the JLink-plugin configuration.

The screenshot below shows an debug session in IDE. For download and more information about emIDE, please refer to *http://emide.org*.



```
#include "BSP.h"

/*************************************************
*       _Delay
*/
static void _Delay(void) {
  volatile int i;

  i = 1000;
  while (i--);
}

/*************************************************
*       main
*/
int main(void) {
  volatile int cnt;
  cnt = 0;
  BSP_Init();                          /* Initiali
  BSP_SetLED(0);
  while (1) {
    BSP_ToggleLED(1);
    cnt++;
    _Delay();
  }
```

CPU Registers

```
R..        Hex  Flags               Decimal
r0         0x0                            0
r1         0x0                            0
r2         0x7                            7
r3  0xffffff430                  4294964272
r4         0x0                            0
r5         0x0                            0
r6         0x0                            0
r7         0x0                            0
r8         0x0                            0
r9         0x0                            0
r10        0x0                            0
r11        0x0                            0
r12        0x0                            0
sp    0x2004f0                      2098416
lr    0x10023c                      1049148
pc    0x10023c                      1049148
cpsr 0x60000013  SVC mode ARM    1610612755
```

Locals

```
Variable  Value
cnt       0x00000000
```

Watches (new)

```
cnt      0      volatile int
```

Disassembly
Function:
Frame start: 00200500

```
0x00100228    push    {lr}                    ; (str
0x0010022C    sub     sp, sp, #12
0x00100230    mov     r3, #0
0x00100234    str     r3, [sp, #4]
0x00100238    bl      0x1000f8 <BSP_Init>
0x0010023C    mov     r0, #0
0x00100240    bl      0x10014c <BSP_SetLED>
0x00100244    mov     r0, #1
0x00100248    bl      0x10019c <BSP_ToggleLED
0x0010024C    ldr     r3, [sp, #4]
0x00100250    add     r3, r3, #1
0x00100254    str     r3, [sp, #4]
0x00100258    bl      0x1001ec <_Delay>
0x0010025C    b       0x100244 <main+28>
```

Breakpoints

Type	Filename/Address
● Code	C:\Tool\C\emIDE V2.1\arm\Example\Vend
● Code	C:\Tool\C\emIDE V2.1\arm\Example\Vend
● Code	C:\Tool\C\emIDE V2.1\arm\Example\Vend

Memory
Unitsize: 8 bit   Go to Address: 0x000000   Go

```
0x00000000:  04 f0 1f e5 8c 00 10 00 60 00 9f e5 0f e0 a0 e1
0x00000010:  10 ff 2f e1 68 10 9f e5 68 20 9f e5 68 30 9f e5
0x00000020:  03 00 52 e1 04 00 91 34 04 00 82 34 fb ff ff 3a
0x00000030:  48 10 9f e5 48 20 9f e5 00 30 a0 e3 02 00 51 e1
0x00000040:  04 30 81 14 fc ff ff 1a 00 00 0f e1 c0 00 c0 e3
0x00000050:  00 f0 29 e1 00 00 a0 e3 00 10 a0 e3 24 20 9f e5
0x00000060:  0f e0 a0 e1 12 ff 2f e1 fe ff ff ea 1e ff 2f e1
0x00000070:  dc 00 10 00 60 02 10 00 00 00 20 00 00 00 20 00
0x00000080:  00 00 20 00 00 00 20 00 28 02 10 00 db f0 21 e3
0x00000090:  2c d0 9f e5 d7 f0 21 e3 28 d0 9f e5 d1 f0 21 e3
0x000000a0:  24 d0 9f e5 d2 f0 21 e3 20 d0 9f e5 d3 f0 21 e3
0x000000b0:  1c d0 9f e5 1c 00 9f e5 0f e0 a0 e1 10 ff 2f e1
0x000000c0:  fe ff ff ea 00 01 20 00 00 01 20 00 00 00 20 00
0x000000d0:  00 01 20 00 00 05 20 00 08 00 10 00 10 30 9f e5
0x000000e0:  02 29 a0 e3 00 20 83 e5 01 30 a0 e3 03 00 a0 e1
0x000000f0:  1e ff 2f e1 44 fd ff ff 3c 30 9f e5 38 20 9f e5
```

Debugger

```
At C:/Tool/C/EMIDEV~1.1/arm/Example/Vendor/ATMEL_~1/Src\Main.c:28
At C:/Tool/C/EMIDEV~1.1/arm/Example/Vendor/ATMEL_~1/Src\Main.c:29
```

Command:

Console

GDB can be used stand-alone as a console application.

To connect GDB to GDB Server enter `target remote localhost:2331` into the running GDB. Within GDB all GDB commands and the remote monitor commands are available. For more information about debugging with GDB refer to its online manual available at *http://sourceware.org/gdb/current/onlinedocs/gdb/*.

A typical startup of a debugging session can be like:

```
(gdb) file C:/temp/Blinky.elf
 Reading symbols from C:/temp/Blinky.elf...done.
(gdb) target remote localhost:2331
 Remote debugging using localhost:2331
 0x00000000 in ?? ()
(gdb) monitor reset
 Resetting target
(gdb) load
 Loading section .isr_vector, size 0x188 lma 0x8000000
 Loading section .text, size 0x568 lma 0x8000188
 Loading section .init_array, size 0x8 lma 0x80006f0
 Loading section .fini_array, size 0x4 lma 0x80006f8
 Loading section .data, size 0x428 lma 0x80006fc
 Start address 0x8000485, load size 2852
 Transfer rate: 146 KB/sec, 570 bytes/write.
(gdb) break main
 Breakpoint 1 at 0x800037a: file Src\main.c, line 38.
(gdb) continue
 Continuing.

 Breakpoint 1, main () at Src\main.c:38
 38        Cnt = 0;
(gdb)
```

Eclipse (CDT)

Eclipse is an open source platform-independent software framework, which has typically been used to develop integrated development environment (IDE). Therefore Eclipse can be used as C/C++ IDE, if you extend it with the CDT plug-in (*http://www.eclipse.org/cdt/*).

CDT means "C/C++ Development Tooling" project and is designed to use the GDB as default debugger and works without any problems with the GDB Server.

Refer to *http://www.eclipse.org* for detailed information about Eclipse.

Note: We only support problems directly related to the GDB Server. Problems and questions related to your remaining toolchain have to be solved on your own.

3.3.3 Supported remote (monitor) commands

J-Link GDB Server comes with some functionality which are not part of the standard GDB. These functions can be called either via a gdbinit file passed to GDB Server or via monitor commands passed directly to GDB, forwarding them to GDB Server.

To indicate to GDB to forward the command to GDB Server 'monitor' has to be prepended to the call. For example a reset can be triggered in the gdbinit file with "reset" or via GDB with "monitor reset".

Following remote commands are available:

Remote command	Explanation
clrbp	Removes an instruction breakpoint.
cp15	Reads or writes from/to cp15 register.
device	Select the specified target device.
DisableChecks	Do not check if an abort occurred after memory read (ARM7/9 only).
EnableChecks	Check if an abort occurred after memory read (ARM7/9 only).
flash breakpoints	Enables/Disables flash breakpoints.
getargs	Get the arguments for the application.
go	Starts the target CPU.
halt	Halts the target CPU.
jtagconf	Configures a JTAG scan chain with multiple devices on it.
memU8	Reads or writes a byte from/to given address.
memU16	Reads or writes a halfword from/to given address.
memU32	Reads or writes a word from/to given address.
reg	Reads or writes from/to given register.
regs	Reads and displays all CPU registers.
reset	Resets and halts the target CPU.
semihosting breakOnError	Enable or disable halting the target on semihosting error.
semihosting enable	Enables semihosting.
semihosting IOClient	Set semihosting I/O to be handled via Telnet port or GDB.
semihosting ARMSWI	Sets the SWI number used for semihosting in ARM mode.
semihosting ThumbSWI	Sets the SWI number used for semihosting in thumb mode.
setargs	Set the arguments for the application.
setbp	Sets an instruction breakpoint at a given address.
sleep	Sleeps for a given time period.
speed	Sets the JTAG speed of J-Link / J-Trace.
step	Performs one or more single instruction steps.
SWO DisableTarget	Undo target configuration for SWO and disable it in J-Link.
SWO EnableTarget	Configure target for SWO and enable it in J-Link.
SWO GetMaxSpeed	Prints the maximum supported SWO speed for J-Link and Target CPU.
SWO GetSpeedInfo	Prints the available SWO speed and its minimum divider.
waithalt	Waits for target to halt code execution.
wice	Writes to given IceBreaker register.

Table 3.2: GDB remote commands

Following remote commands are deprecated and only available for backward compatibility:

Remote command	Explanation
device	Select the specified target device. **Note:** Use command line option -device instead.
interface	Select the target interface. **Note:** Use command line option -if instead.
speed	Sets the JTAG speed of J-Link / J-Trace. **Note:** For the initial connection speed, use command line option -speed instead.

Table 3.3: GDB remote commands

Note: The remote commands are case-insensitive.

Note: Optional parameters are set into square brackets.

Note: The examples are described as follows:
Lines starting with '#' are comments and not used in GDB / GDB Server.
Lines starting with '>' are input commands from the GDB.
Lines starting with '<' is the output from GDB Server as printed in GDB.

3.3.3.1 clrbp

Syntax

```
ClrBP [<BPHandle>]
```

or

```
ci [<BPHandle>]
```

Description

Removes an instruction breakpoint, where `<BPHandle>` is the handle of breakpoint to be removed. If no handle is specified this command removes all pending breakpoints.

Example

```
> monitor clrbp 1
```

or

```
> monitor ci 1
```

3.3.3.2 cp15

Syntax

```
cp15 <CRn>, <CRm>, <op1>, <op2> [= <data>]
```

Description

Reads or writes from/to cp15 register. If `<data>` is specified, this command writes the data to the cp15 register. Otherwise this command reads from the cp15 register. For further information please refer to the ARM reference manual.

Example

```
#Read:
> monitor cp15 1, 2, 6, 7
< Reading CP15 register (1,2,6,7 = 0x0460B77D)

#Write:
> monitor cp15 1, 2, 6, 7 = 0xFFFFFFFF
```

3.3.3.3 device

Note: Deprecated. Use command line option `-device` instead.

Syntax

```
device <DeviceName>
```

Description

Selects the specified target device. This is necessary for the connection and some special handling of the device.

Note: The device should be selected via commandline option `-device` when starting GDB Server.

Example

```
> monitor device STM32F417IG
< Selecting device: STM32F417IG
```

3.3.3.4 DisableChecks

Syntax

```
DisableChecks
```

Description

Disables checking if a memory read caused an abort (ARM7/9 devices only). On some CPUs during the init sequence for enabling access to the internal memory (for example on the TMS470) some dummy reads of memory are required which will cause an abort as long as the access-init is not completed.

3.3.3.5 EnableChecks

Syntax

```
EnableChecks
```

Description

Enables checking if a memory read caused an abort (ARM7/9 devices only). On some CPUs during the init sequence for enabling access to the internal memory (for example on the TMS470) some dummy reads of memory are required which will cause an abort as long as the access-init is not completed. The default state is: Checks enabled.

3.3.3.6 flash breakpoints

Syntax

```
monitor flash breakpoints = <Value>
```

Description

This command enables/disables the Flash Breakpoints feature.

By default Flash Breapkoints are enabled and can be used for evaluation.

Example

```
#Disable Flash Breakpoints:
> monitor flash breakpoints = 0
< Flash breakpoints disabled

#Enable Flash Breakpoins:
> monitor flash breakpoints = 1
< Flash breakpoints enabled
```

3.3.3.7 getargs

Syntax

```
getargs
```

Description

Get the currently set argument list which will be given to the application when calling semihosting command SYS_GET_CMDLINE (0x15). The argument list is given as one string.

Example

```
#No arguments set via setargs:
> monitor getargs
< No arguments.
#Arguments set via setargs:
> monitor getargs
< Arguments: test 0 1 2 arg0=4
```

3.3.3.8 go

Syntax

```
go
```

Description

Starts the target CPU.

Example

```
> monitor go
```

3.3.3.9 halt

Syntax

```
halt
```

Description

Halts the target CPU.

Example

```
> monitor halt
```

3.3.3.10 interface

Note: Deprecated. Use command line option -if instead.

Syntax

```
interface <InterfaceIdentifier>
```

Description

Selects the target interface used by J-Link / J-Trace.

3.3.3.11 jtagconf

Syntax

```
jtagconf <IRPre> <DRPre>
```

Description

Configures a JTAG scan chain with multiple devices on it. <IRPre> is the sum of IRLens of all devices closer to TDI, where IRLen is the number of bits in the IR (Instruction Register) of one device. <DRPre> is the number of devices closer to TDI. For more detailed information of how to configure a scan chain with multiple devices please refer to See “Determining values for scan chain configuration” on page 147..

Note: To make sure the connection to the device can be established correctly, it is recommended to configure the JTAG scan chain via command line options at start of GDB Server.

Example

```
#Select the second device, where there is 1 device in front with IRLen 4
> monitor jtagconf 4 1
```

3.3.3.12 memU8

Syntax

```
MemU8 <address> [= <value>]
```

Description

Reads or writes a byte from/to a given address. If <value> is specified, this command writes the value to the given address. Otherwise this command reads from the given address.

Example

```
#Read:
> monitor memU8 0x50000000
< Reading from address 0x50000000 (Data = 0x04)

#Write:
> monitor memU8 0x50000000 = 0xFF
< Writing 0xFF @ address 0x50000000
```

3.3.3.13 memU16

Syntax

`memU16 <address> [= <value>]`

Description

Reads or writes a halfword from/to a given address. If `<value>` is specified, this command writes the value to the given address. Otherwise this command reads from the given address.

Example

```
#Read:
> monitor memU16 0x50000000
< Reading from address 0x50000000 (Data = 0x3004)

#Write:
> monitor memU16 0x50000000 = 0xFF00
< Writing 0xFF00 @ address 0x50000000
```

3.3.3.14 memU32

Syntax

`MemU32 <address> [= <value>]`

Description

Reads or writes a word from/to a given address. If `<value>` is specified, this command writes the value to the given address. Otherwise this command reads from the given address. This command is similar to the long command.

Example

```
#Read:
> monitor memU32 0x50000000
< Reading from address 0x50000000 (Data = 0x10023004)

#Write:
> monitor memU32 0x50000000 = 0x10023004
< Writing 0x10023004 @ address 0x50000000
```

3.3.3.15 reg

Syntax

`reg <RegName> [= <value>]`

or

`reg <RegName> [= (<address>)]`

Description

Reads or writes from/to given register. If `<value>` is specified, this command writes the value into the given register. If `<address>` is specified, this command writes the memory content at address `<address>` to register `<RegName>`. Otherwise this command reads the given register.

Example

```
#Write value to register:
> monitor reg pc   = 0x00100230
< Writing register (PC = 0x00100230)

#Write value from address to register:
> monitor reg r0   = (0x00000040)
< Writing register (R0 = 0x14813004)

#Read register value:
> monitor reg PC
< Reading register (PC = 0x00100230)
```

3.3.3.16 regs

Syntax

```
regs
```

Description

Reads all CPU registers.

Example

```
> monitor regs
< PC = 00100230, CPSR = 20000013 (SVC mode, ARM)
  R0 = 14813004, R1 = 00000001, R2 = 00000001, R3 = 000003B5
  R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 00000000
  USR: R8 =00000000, R9 =00000000, R10=00000000, R11 =00000000, R12 =00000000
       R13=00000000, R14=00000000
  FIQ: R8 =00000000, R9 =00000000, R10=00000000, R11 =00000000, R12 =00000000
       R13=00200000, R14=00000000, SPSR=00000010
  SVC: R13=002004E8, R14=0010025C, SPSR=00000010
  ABT: R13=00200100, R14=00000000, SPSR=00000010
  IRQ: R13=00200100, R14=00000000, SPSR=00000010
  UND: R13=00200100, R14=00000000, SPSR=00000010
```

3.3.3.17 reset

Syntax

```
reset
```

Description

Resets and halts the target CPU. Make sure the device is selected prior to using this command to make use of the correct reset strategy.

Add. information

There are different reset strategies for different CPUs. Moreover, the reset strategies which are available differ from CPU core to CPU core. J-Link can perform various reset strategies and always selects the best fitting strategy for the selected device.

Example

```
> monitor reset
< Resetting target
```

3.3.3.18 semihosting breakOnError

Syntax

```
semihosting breakOnerror <Value>
```

Description

Enables or disables halting the target at the semihosting breakpoint / in SVC handler if an error occurred during a semihosting command, for example a bad file handle for SYS_WRITE. The GDB Server log window always shows a warning in these cases. breakOnError is disabled by default.

Example

```
#Enable breakOnError:
> monitor semihosting breakOnError 1
```

3.3.3.19 semihosting enable

Syntax

```
semihosting enable [<VectorAddr>]
```

Description

Enables semihosting with the specified vector address. If no vector address is specified, the SWI vector (at address 0x8) will be used. GDBServer will output semihosting terminal data from the target via a separate connection on port 2333. Some IDEs already establish a connection automatically on this port and show terminal data in a specific window in the IDE.
For IDEs which do not support semihosting terminal output directly, the easiest way to view semihosting output is to open a telnet connection to the GDBServer on port 2333. The connection on this port can be opened all the time as soon as GDBServer is started, even before this remote command is executed.

Example

```
> monitor semihosting enable
< Semihosting enabled (VectorAddr = 0x08)
```

3.3.3.20 semihosting IOClient

Syntax

```
semihosting IOClient <ClientMask>
```

Description

GDB itself can handle (file) I/O operations, too. With this command it is selected wheter to print output via TELNET port (2333), GDB, or both.
<ClientMask> is

- 1 for TELNET Client (Standard port 2333) (Default)
- 2 for GDB Client
- or 3 for both (Input via GDB Client)

Example

```
#Select TELNET port as output source
> monitor semihosting ioclient 1
< Semihosting I/O set to TELNET Client

#Select GDB as output source
> monitor semihosting ioclient 2
< Semihosting I/O set to GDB Client

#Select TELNET port and GDB as output source
> monitor semihosting ioclient 3
< Semihosting I/O set to TELNET and GDB Client
```

3.3.3.21 semihosting ARMSWI

Syntax

```
semihosting ARMSWI <Value>
```

Description

Sets the SWI number used for semihosting in ARM mode. The default value for the ARMSWI is 0x123456.

Example

```
> monitor semihosting ARMSWI 0x123456
< Semihosting ARM SWI number set to 0x123456
```

3.3.3.22 semihosting ThumbSWI

Syntax

```
semihosting ThumbSWI <Value>
```

Description

Sets the SWI number used for semihosting in thumb mode. The default value for the ThumbSWI is 0xAB

Example

```
> monitor semihosting ThumbSWI 0xAB
< Semihosting Thumb SWI number set to 0xAB
```

3.3.3.23 setargs

Syntax

```
setargs <ArgumentString>
```

Description

Set arguments for the application, where all arguments are in one <ArgumentString> separated by whitespaces.

The argument string can be gotten by the application via semihosting command SYS_GET_CMDLINE (0x15).

Semihosting has to be enabled for getting the argumentstring (`semihosting enable`). "monitor setargs" can be used before enabeling semihosting.

The maximum length for <ArgumentString> is 512 characters.

Example

```
> monitor setargs test 0 1 2 arg0=4
< Arguments: test 0 1 2 arg0=4
```

3.3.3.24 setbp

Syntax

```
setbp <Addr> [<Mask>]
```

Description

Sets an instruction breakpoint at the given address, where `<Mask>` can be 0x03 for ARM instruction breakpoints (Instruction width 4 Byte, mask out lower 2 bits) or 0x01 for THUMB instruction breakpoints (Instruction width 2 Byte, mask out lower bit). If no mask is given, an ARM instruction breakpoint will be set.

Example

```
#Set a breakpoint (implicit for ARM instructions)
> monitor setbp 0x00000000

#Set a breakpoint on a THUMB instruction
> monitor setbp 0x00000100 0x01
```

3.3.3.25 sleep

Syntax

```
sleep <Delay>
```

Description

Sleeps for a given time, where `<Delay>` is the time period in milliseconds to delay. While sleeping any communication is blocked until the command returns after the given period.

Example

```
> monitor sleep 1000
< Sleep 1000ms
```

3.3.3.26 speed

Note: Deprecated. For setting the initial connection speed, use command line option `-speed` instead.

Syntax

```
speed <kHz>|auto|adaptive
```

Description

Sets the JTAG speed of J-Link / J-Trace. Speed can be either fixed (in kHz), automatic recognition or adaptive. In general, Adaptive is recommended if the target has an RTCK signal which is connected to the corresponding RTCK pin of the device (S-cores only). For detailed information about the different modes, refer to *JTAG Speed* on page 148.

The speed has to be set after selecting the interface, to change it from its default value.

Example

```
> monitor speed auto
< Select auto target interface speed (8000 kHz)

> monitor speed 4000
< Target interface speed set to 4000 kHz

> monitor speed adaptive
< Select adaptive clocking instead of fixed JTAG speed
```

3.3.3.27 step

Syntax

```
step [<NumSteps>]
```

or

```
si [<NumSteps>]
```

Description

Performs one or more single instruction steps, where <NumSteps> is the number of instruction steps to perform. If `<NumSteps>` is not specified only one instruction step will be performed.

Example

```
> monitor step 3
```

3.3.3.28 SWO DisableTarget

Syntax

```
SWO DisableTarget <PortMask[0x01-0xFFFFFFFF]>
```

Description

Disables the output of SWO data on the target (Undoes changes from SWO EnableTarget) and stops J-Link to capture it.

Example

```
#Disable captureing SWO from stimulus ports 0 and 1
> monitor SWO DisableTarget 3
< SWO disabled succesfully.
```

3.3.3.29 SWO EnableTarget

Syntax

```
SWO EnableTarget <CPUFreq[Hz]> <SWOFreq[Hz]> <PortMask[0x01-0xFFFFFFFF]
<Mode[0]>
```

Description

Configures the target to be able to output SWO data and starts J-Link to capture it.
CPU and SWO frequency can be 0 for auto-detection.
If CPUFreq is 0, J-Link will measure the current CPU speed.
If SWOFreq is 0, J-Link will use the highest available SWO speed for the selected / measured CPU speed.

Note: CPUFreq has to be the speed at which the target will be running when doing SWO.
If the speed is different from the current speed when issuing CPU speed auto-detection, getting SWO data might fail.
SWOFreq has to be a quotient of the CPU and SWO speeds and their prescalers. To get available speed, use SWO GetSpeedInfo.
PortMask can be a decimal or hexadecimal Value. Values starting with the Prefix "0x" are handled hexadecimal.

Example

```
#Configure SWO for stimulus port 0, measure CPU frequency and calculate SWO frequency
> monitor SWO EnableTarget 0 0 1 0
< SWO enabled succesfully.

#Configure SWO for stimulus ports 0-2, fixed SWO frequency and measure CPU frequency
> monitor SWO EnableTarget 0 1200000 5 0
< SWO enabled succesfully.

#Configure SWO for stimulus ports 0-255, fixed CPU and SWO frequency
> monitor SWO EnableTarget 72000000 6000000 0xFF 0
< SWO enabled succesfully.
```

3.3.3.30 SWO GetMaxSpeed

Syntax

SWO GetMaxSpeed <CPUFrequency [Hz]>

Description

Prints the maximum SWO speed supported by and matching both, J-Link and the target CPU frequency.

Example

```
#Get SWO speed for 72MHz CPU speed
> monitor SWO GetMaxSpeed 72000000
< Maximum supported SWO speed is 6000000 Hz.
```

3.3.3.31 SWO GetSpeedInfo

Syntax

`SWO GetSpeedInfo`

Description

Prints the base frequency and the minimum divider of the connected J-Link. With this, the available SWO speeds for J-Link can be calculated and the matching one for the target CPU frequency can be selected.

Example

```
> monitor SWO GetSpeedInfo
< Base frequency: 60000000Hz, MinDiv: 8
# Available SWO speeds for J-Link are: 7.5MHz, 6.66MHz, 6MHz, ...
```

3.3.3.32 waithalt

Syntax

`waithalt <Timeout>`

or

`wh <Timeout>`

Description

Waits for target to halt code execution, where `<Timeout>` is the maximum time period in milliseconds to wait.

Example

```
#Wait for halt with a timeout of 2 seconds
> monitor waithalt 2000
```

3.3.3.33 wice

Syntax

`wice <RegIndex> <value>`

or

`rmib <RegIndex> <value>`

Description

Writes to given IceBreaker register, where `<value>` is the data to write.

Example

```
> monitor wice 0x0C 0x100
```

3.3.4 SEGGER-specific GDB protocol extensions

J-Link GDB Server implements some functionality which are not part of the standard GDB remote protocol in general query packets. These SEGGER-specific general query packets can be sent to GDB Server on the low-level of GDB, via maintanace commands, or with a custom client connected to GDB Server.

General query packets start with a ’q’. SEGGER-specific general queries are followed by the identifier ’Segger’ plus the command group, the actual command and its parameters.

Following SEGGER-specific general query packets are available:

Query Packet	Explanation
qSeggerSTRACE:config	Configure STRACE for usage.
qSeggerSTRACE:start	Start STRACE.
qSeggerSTRACE:stop	Stop STRACE.
qSeggerSTRACE:read	Read STRACE data.

Table 3.4: General Queries

3.3.4.1 qSeggerSTRACE:config

Syntax

```
qSeggerSTRACE:config:<ConfigString>
```

Parameter

ConfigString: String containing the configuration data separating settings by ’;’.

Description

Configures STRACE for usage. Configuration for example includes specification of the trace port width to be used for tracing (1-bit, 2-bit, 4-bit (default) PortWidth=%Var%.

Note: For more information please refer to *UM08002* (J-Link SDK user guide), chapter *STRACE*.

Response

```
<ReturnValue>
```

ReturnValue is a 4 Byte signed integer.

>= 0	O.K.
< 0	Error.

Note: ReturnValue is hex-encoded.
Return value 0 is "00000000", return value -1 is "FFFFFFFF".

3.3.4.2 qSeggerSTRACE:start

Syntax

```
qSeggerSTRACE:start
```

Description

Starts capturing of STRACE data.

Note: For more information please refer to *UM08002* (J-Link SDK user guide), chapter *STRACE*.

Response

```
<ReturnValue>
```

ReturnValue is a 4 Byte signed integer.

>= 0	O.K.
< 0	Error.

Note: ReturnValue is hex-encoded.
Return value 0 is "00000000", return value -1 is "FFFFFFFF".

3.3.4.3 qSeggerSTRACE:stop

Syntax

```
qSeggerSTRACE:stop
```

Description

Stops capturing of STRACE data.

Note: For more information please refer to *UM08002* (J-Link SDK user guide), chapter *STRACE*.

Response

```
<ReturnValue>
```

ReturnValue is a 4 Byte signed integer.

>= 0	O.K.
< 0	Error.

Note: ReturnValue is hex-encoded.
Return value 0 is "00000000", return value -1 is "FFFFFFFF".

3.3.4.4 qSeggerSTRACE:read

Syntax

```
qSeggerSTRACE:read:<NumItems>
```

Parameter

NumItems: Maximum number of trace data (addresses) to be read. Hexadecimal.

Description

Read the last recently called instruction addresses. The addresses are returned LIFO, meaning the last recent address is returned first.

Note: For more information please refer to *UM08002* (J-Link SDK user guide), chapter *STRACE*.

Response

```
<ReturnValue>[<Item0><Item1>...]
```

ReturnValue is a 4 Byte signed integer.

>= 0	Number of items read.
< 0	Error.

ItemN is a 4 Byte unsigned integer.

0x00000000 - 0xFFFFFFFF Address of the executed instruction

Note: ReturnValue and ItemN are hex-encoded.
e.g. 3 Items read: 0x08000010, 0x08000014, 0x08000018
Response will be: 00000003080000100800001408000018

3.3.5 Command line options

There are several command line options available for the GDB Server which allow configuration of the GDB Server before any connection to a J-Link is attempted or any connection from a GDB client is accepted.

Note: Using GDB Server CL, device, interface, endian and speed are mandatory options to correctly connect to the target, and should be given before connection via GDB. Using GDB Server GUI the mandatory options can also be sleceted in the configuration dialog.

Command line option	Explanation
`-device`	Select the connected target device.
`-endian`	Select the device endianness.
`-if`	Select the interface to connect to the target.
`-speed`	Select the target communication speed.

Table 3.5: Mandatory command line options

Note: Using multiple instances of GDB Server, setting custom values for port, SWOPort and TelnetPort is necessary.

Command line option	Explanation
`-port`	Select the port to listen for GDB clients.
`-swoport`	Select the port to listen for clients for SWO RAW output.
`-telnetport`	Select the port to listen for clients for printf output.

Table 3.6: Port selection command line options

The GDB Server GUI version uses persistent settings which are saved across different instances and sessions of GDB Server. These settings can be toggled via the checkboxes in the GUI.

Note: GDB Server CL always starts with the settings marked as default.

For GUI and CL, the settings can be changed with following command line options. For all persistent settings there is a pair of options to enable or disable the feature.

Command line option	Explanation
`-ir` `-noir`	Initialize the CPU registers on start of GDB Server. (Default) Do not initialize CPU registers on start of GDB Server.
`-localhostonly` `-nolocalhostonly`	Allow only localhost connections (Windows default) Allow connections from outside localhost (Linux default)
`-logtofile` `-nologtofile`	Generate a GDB Server log file. Do not generate a GDB Server log file. (Default)
`-halt` `-nohalt`	Halt the target on start of GDB Server. Do not halt the target on start of GDB Server. (Default)
`-silent` `-nosilent`	Do not show log output. Show log output. (Default)
`-stayontop` `-nostayontop`	Set the GDB Server GUI to be the topmost window. Do not be the topmost window. (Default)
`-timeout` `-notimeout`	Set the time after which the target has to be connected. Set infinite timeout for target connection.
`-vd` `-novd`	Verify after downloading. Do not verify after downloading. (Default)

Table 3.7: Persistent command line options

Following additional command line options are available. These options are temporary for each start of GDB Server.

Command line option	Explanation
`-jtagconf`	Configures a JTAG scan chain with multiple devices on it.
`-log`	Log the GDB Server communication to a specific file.
`-singlerun`	Start GDB Server in single run mode.
`-scriptfile`	Use a J-Link scriptfile.
`-select`	Select the interface to connect to J-Link (USB/IP).
`-settingsfile`	Select the J-Link Settings File.
`-strict`	Starts GDB Server in strict mode.
`-x`	Execute a gdb file on first connection.
`-xc`	Execute a gdb file on every connection.
`-cpu`	Select the CPU core. Deprecated, use `-device` instead.

Table 3.8: General command line options

3.3.5.1 -cpu

Description

Pre-select the CPU core of the connected device, so the GDB Server already knows the register set, even before having established a connection to the CPU.

Note: Deprecated, please use -device instead. Anyhow, does not hurt if this option is set, too.

Syntax

```
-CPU <CPUCore>
```

Example

```
jlinkgdbserver -CPU ARM7_9
```

Add. information

The following table lists all valid values for <CPUCore>:

<CPUCore>	Supported CPU cores
CPU_FAMILY_ARM7_9	Pre-select ARM7 and ARM9 as CPU cores.
CPU_FAMILY_CORTEX_A_R	Pre-select Cortex-A and Cortex-R as CPU cores.
CPU_FAMILY_CORTEX_M	Pre-select Cortex-M as CPU core.
CPU_FAMILY_RX600	Pre-select Renesas RX600 as CPU core.

Table 3.9: GDB allowed values for CPUCore

3.3.5.2 -device

Description

Tells GDBServer to which device J-Link is connected before the connect sequence is actually performed. It is recommended to use the command line option to select the device instead of using the remote command since for some devices J-Link already needs to know the device at the time of connecting to it since some devices need special connect sequences (e.g. devices with TI ICEPick modules). In such cases, it is not possible to select the device via remote commands since they are configured after the GDB client already connected to GDBServer and requested the target registers which already requires a connection to the target.

Note: Using GDB Server CL this option is mandatory to correctly connect to the target, and should be given before connection via GDB.

Syntax

```
-device <DeviceName>
```

Example

```
jlinkgdbserver -device AT91SAM7SE256
```

Add. information

For a list of all valid values for <DeviceName>, please refer to *http://www.segger.com/jlink_supported_devices.html*.

3.3.5.3 -endian

Description

Sets the endianess of the target where endianess can either be "little" or "big".

Note: Using GDB Server CL this option is mandatory to correctly connect to the target, and should be given before connection via GDB.

Syntax

```
-endian <endianess>
```

Example

```
jlinkgdbserver -endian little
```

3.3.5.4 -if

Description

Selects the target interface which is used by J-Link to connect to the device. The default value is JTAG.

Note: Using GDB Server CL this option is mandatory to correctly connect to the target, and should be given before connection via GDB.

Syntax

```
-if <Interface>
```

Example

```
jlinkgdbserver -if SWD
```

Add. information

Currently, the following values are accepted for `<Interface>`:

- JTAG
- SWD
- FINE
- 2-wire-JTAG-PIC32

3.3.5.5 -ir

Description

Initializes the CPU register with default values on startup.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via -noir or the GUI.

Example

```
jlinkgdbserver -ir
```

3.3.5.6 -jtagconf

Syntax

```
-jtagconf <IRPre>,<DRPre>
```

Description

Configures a JTAG scan chain with multiple devices on it. `<IRPre>` is the sum of IRLens of all devices closer to TDI, where IRLen is the number of bits in the IR (Instruction Register) of one device. `<DRPre>` is the number of devices closer to TDI. For more detailed information of how to configure a scan chain with multiple devices please refer to See “Determining values for scan chain configuration” on page 147..

Example

```
#Select the second device, where there is 1 device in front with IRLen 4
jlinkgdbserver -jtagconf 4,1
```

3.3.5.7 -localhostonly

Description

Starts the GDB Server with the option to listen on localhost only (This means that only TCP/IP connections from localhost are accepted) or on any IP address. To allow remote debugging (connecting to GDBServer from another PC), deactivate this option.

If no parameter is given, it will be set to 1 (active).

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-LocalhostOnly <State>
```

Example

```
jlinkgdbserver -LocalhostOnly 0 //Listen on any IP address (Linux/MAC default)

jlinkgdbserver -LocalhostOnly 1 //Listen on localhost only (Windows default)
```

3.3.5.8 -log

Description

Starts the GDB Server with the option to write the output into a given log file.
The file will be created if it does not exist. If it exists the previous content will be removed. Paths including spaces need to be set between quotes.

Syntax

```
-log <LogFilePath>
```

Example

```
jlinkgdbserver -log “C:\my path\to\file.log”
```

3.3.5.9 -logtofile

Description

Starts the GDB Server with the option to write the output into a log file.

If no file is given via `-log`, the log file will be created in the GDB Server application directory.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via -nologtofile or the GUI.

Syntax

```
-logtofile
```

Example

```
jlinkgdbserver -logtofile

jlinkgdbserver -logtofile -log “C:\my path\to\file.log”
```

3.3.5.10 -halt

Description

Halts the target after connecting to it on start og GDB Server.

For most IDEs this option is mandatory since they rely on the target to be halted after connecting to GDB Server.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via -nohalt or the GUI.

Syntax

```
-halt
```

Example

```
jlinkgdbserver -halt
```

3.3.5.11 -noir

Description

Do not initialize the CPU registers on startup.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via -ir or the GUI.

Syntax

```
-noir
```

3.3.5.12 -nolocalhostonly

Description

Starts GDB Server with the option to allow remote connections (from outside localhost).

Same as `-localhostonly 0`

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-nolocalhostonly
```

3.3.5.13 -nologtofile

Description

Starts the GDB Server with the option to not write the output into a log file.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via -nologtofile or the GUI.

Note: When this option is used after -log, no log file will be generated, when -log is used after this option, a log file will be generated and this setting will be overridden.

Syntax

```
-nologtofile
```

Example

```
jlinkgdbserver -nologtofile                              // Will not generate a log file

jlinkgdbserver -nologtofile -log “C:\pathto\file.log” // Will generate a log file

jlinkgdbserver -log “C:\pathto\file.log” -nologtofile // Will not generate a log file
```

3.3.5.14 -nohalt

Description

When connecting to the target after starting GDB Server, the target is not explicitly halted and the CPU registers will not be inited.
After closing all GDB connections the target is started again and continues running.

Some IDEs rely on the target to be halted after connect. In this case do not use -nohalt, but -halt.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via -halt or the GUI.

Syntax

```
-nohalt
```

Example

```
jlinkgdbserver -nohalt
```

3.3.5.15 -nosilent

Description

Starts the GDB Server in non-silent mode. All log window messages will be shown.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-nosilent
```

3.3.5.16 -nostayontop

Description

Starts the GDB Server in non-topmost mode. All windows can be placed above it.

Note: For the CL version this setting has no effect.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-nostayontop
```

3.3.5.17 -notimeout

Description

GDB Server automatically closes after a timout of 5 seconds when no target voltage can be measured or connection to target fails.

This command line option prevents GDB Server from closing, to allow connecting a target after starting GDB Server.

Note: The recommended order is to power the target, connect it to J-Link and then start GDB Server.

Syntax

```
-notimeout
```

3.3.5.18 -novd

Description

Do not explicitly verify downloaded data.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-vd
```

3.3.5.19 -port

Description

Starts GDB Server listening on a specified port. This option overrides the default listening port of the GDB Server. The default port is 2331.

Note: Using multiple instances of GDB Server, setting custom values for this option is necessary.

Syntax

```
-port <Port>
```

Example

```
jlinkgdbserver -port 2345
```

3.3.5.20 -scriptfile

Description

Passes the path of a J-Link script file to the GDB Server. This scriptfile is executed before the GDB Server starts the debugging / identifying communication with the target. J-Link scriptfiles are mainly used to connect to targets which need a special connection sequence before communication with the core is possible. For more information about J-Link script files, please refer to *J-Link script files* on page 169.

Syntax

```
-scriptfile <ScriptFilePath>
```

Example

```
-scriptfile “C:\My Projects\Default.JLinkScript“
```

3.3.5.21 -select

Description

Specifies the host interface to be used to connect to J-Link. Currently, USB and TCP/IP are available.

Syntax

```
-select <Interface>=<SerialNo>/<IPAddr>
```

Example

```
jlinkgdbserver -select usb=580011111
jlinkgdbserver -select ip=192.168.1.10
```

Additional information

For backward compatibility, when USB is used as interface serial numbers from 0-3 are accepted as USB=0-3 to support the old method of connecting multiple J-Links to a PC. This method is no longer recommended to be used. Please use the “connect via emulator serial number‟ method instead.

3.3.5.22 -settingsfile

Description

Select a J-Link settings file to be used for the target device. The settings fail can contain all configurable options of the Settings tab in J-Link Control panel.

Syntax

```
-SettingsFile <PathToFile>
```

Example

```
jlinkgdbserver -SettingsFile "C:\Temp\GDB Server.jlink"
```

3.3.5.23 -silent

Description

Starts the GDB Server in silent mode. No log window messages will be shown.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-silent
```

3.3.5.24 -singlerun

Description

Starts GDB Server in single run mode. When active, GDB Server will close when all client connections are closed.

In normal run mode GDB Server will stay open and wait for new connections.

When started in single run mode GDB Server will close immediately when connecting to the target fails. Make sure it is powered and connected to J-Link before starting GDB Server.

Syntax

```
-s
-singlerun
```

3.3.5.25 -speed

Description

Starts GDB Server with a given initial speed.
Available parameters are “adaptive”, “auto” or a freely selectable integer value in kHz. It is recommended to use either a fixed speed or, if it is available on the target, adaptive speeds.

Note: Using GDB Server CL this option is mandatory to correctly connect to the target, and should be given before connection via GDB.

Syntax

```
-speed <Speed_kHz>
```

Example

```
jlinkgdbserver -speed 2000
```

3.3.5.26 -stayontop

Description

Starts the GDB Server in topmost mode. It will be placed above all non-topmost windows and maintains it position even when it is deactivated.

Note: For the CL version this setting has no effect.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-stayontop
```

3.3.5.27 -timeout

Description

Set the timeout after which the target connection has to be established. If no connection could be established GDB Server will close.

The default timeout is 5 seconds for the GUI version and 0 for the command line version.

Note: The recommended order is to power the target, connect it to J-Link and then start GDB Server.

Syntax

```
-timeout <Timeout[ms]>
```

Example

Allow target connection within 10 seconds.

```
jlinkgdbserver -timeout 10000
```

3.3.5.28 -strict

Description

Starts GDB Server in sctrict mode. When strict mode is active GDB Server checks the correctness of settings and exits in case of a failure.

Currently the device name is checked. If no device name is given or the device is unknown to the J-Link, GDB Server exits instead of selecting "Unspecified" as device or showing the device selection dialog.

Syntax

```
- strict
```

Example

Following executions of GDB Server (CL) will cause exit of GDB Server.

```
jlinkgdbserver -strict -device UnknownDeviceName
```

```
jlinkgdbservercl -strict
```

Following execution of GDB Server will show the device selction dialog under Windows or select "Unspecified" directly under Linux / OS X.

```
jlinkgdbserver -device UnknownDeviceName
```

3.3.5.29 -swoport

Description

Set up port on which GDB Server should listen for an incomming connection that reads the SWO data from GDB Server. Default port is 2332.

Note: Using multiple instances of GDB Server, setting custom values for this option is necessary.

Syntax

```
-SWOPort <Port>
```

Example

```
jlinkgdbserver -SWOPort 2553
```

3.3.5.30 -telnetport

Description

Set up port on which GDB Server should listen for an incoming connection that gets target’s printf data (Semihosting and anylized SWO data). Default port is 2333.

Note: Using multiple instances of GDB Server, setting custom values for this option is necessary.

Syntax

```
-TelnetPort <Port>
```

Example

```
jlinkgdbserver -TelnetPort 2554
```

3.3.5.31 -vd

Description

Verifys the data after downloading it.

Note: For the GUI version, this setting is persistent for following uses of GDB Server until changed via command line option or the GUI.

Syntax

```
-vd
```

3.3.5.32 -x

Description

Starts the GDB Server with a gdbinit (configuration) file. In contrast to the `-xc` command line option the GDB Server runs the commands in the gdbinit file once only direct after the first connection of a client.

Syntax

```
-x <ConfigurationFilePath>
```

Example

```
jlinkgdbserver -x C:\MyProject\Sample.gdb
```

3.3.5.33 -xc

Description

Starts the GDB Server with a gdbinit (configuration) file. GDB Server executes the commands specified in the gdbinit file with every connection of a client / start of a debugging session.

Syntax

```
-xc <ConfigurationFilePath>
```

Example

```
jlinkgdbserver -xc C:\MyProject\Sample.gdb
```

3.3.6 Program termination

J-Link GDB Server is normally terminated by a close or Ctrl-C event. When the single run mode is active it will also close when an error occurred during start or after all connections to GDB Server are closed.

On termination GDB Server will close all connections and disconnect from the target device, letting it run.

3.3.6.1 Exit codes

J-Link GDB Server terminates with an exit code indicating an error by a non-zero exit code.

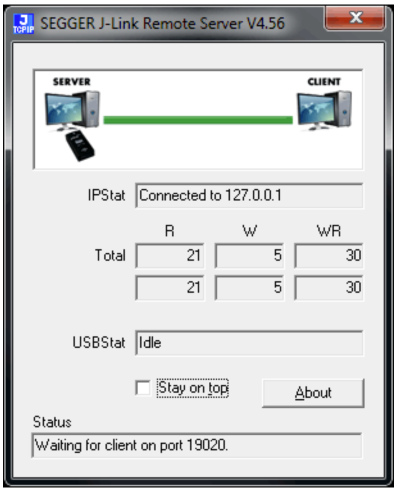
The following table describes the defined exit codes of GDB Server.

Exit code	Description
0	No error. GDB Server closed normally.
-1	Unknown error. Should not happen.
-2	Failed to open listener port (Default: 2331)
-3	Could not connect to target. No target voltage detected or connection failed.
-4	Failed to accept a connection from GDB.
-5	Failed to parse the command line options, wrong or missing command line parameter.
-6	Unknown or no device name set.
-7	Failed to connect to J-Link.

Table 3.10: GDB Server exit codes

3.4 J-Link Remote Server

J-Link Remote 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.



IPStat: Connected to 127.0.0.1

	R	W	WR
Total	21	5	30
	21	5	30

USBStat: Idle

□ Stay on top

Status: Waiting for client on port 19020.

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

3.4.1 List of available commands

The table below lists the commands accepted by the J-Link Remote Server

Command	Description
port	Selects the IP port on which the J-Link Remote Server is listening.
SelectEmuBySN	Select the J-Link to connect to by its Serial Number.

Table 3.11: Available commands

3.4.1.1 port

Syntax

```
-port <Portno.>
```

Example

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

```
-port 19021
```

3.4.1.2 SelectEmuBySN

Syntax

```
-SelectEmuBySN <S/N>
```

Example

To select the emulator with Serial Number 268000000 the command should look as follows:

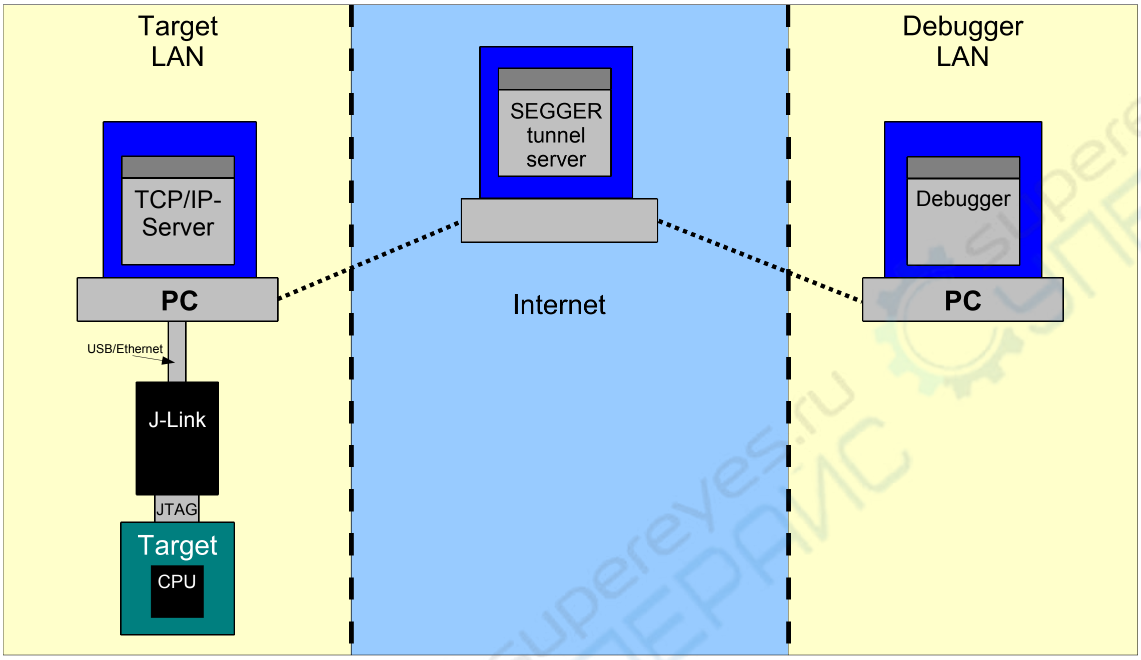
```
-SelectEmuBySN 268000000
```

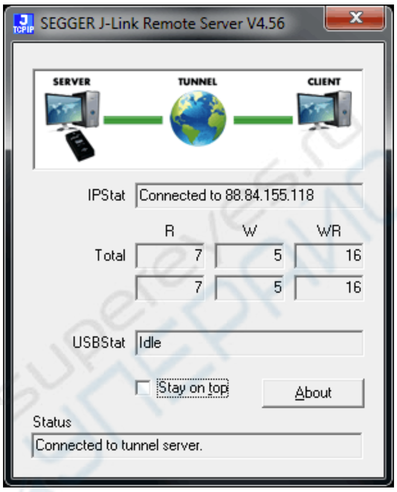
3.4.2 Tunneling mode

The Remote server provides a tunneling mode which allows remote connection to a J-Link / J-Trace from any computer, even from outside the local network.

To give access to a J-Link neither a remote desktop or vpn connection nor changing some difficult firewall settings are necessary.

When started in tunneling mode the Remote server connects to the SEGGER tunnel server via port 19020 and registers with its serial number. To connect to the J-Link from the remote computer an also simple connection to tunnel:<SerialNo> can be established and the debugger is connected to the J-Link.





IPStat Connected to 88.84.155.118

	R	W	WR
Total	7	5	16
	7	5	16

USBStat Idle

□ Stay on top   About

Status
Connected to tunnel server.

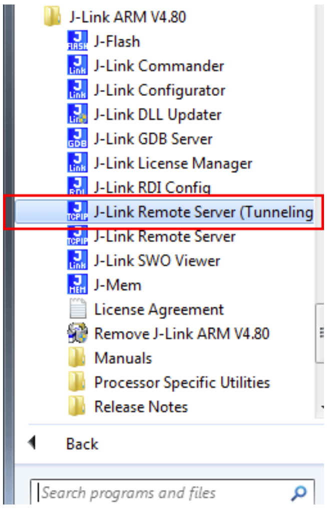
Example scenario

A device vendor is developing a new device which shall be supported by J-Link. Because there is only one prototype, a shipment to SEGGER is not possible.

Instead the vendor can connect the device via J-Link to a local computer and start the Remote server in tunneling mode. He then gives the serial number of the J-Link to an engineer at SEGGER.

The engineer at SEGGER can use J-Link Commander or a debugger to test and debug the new device without the need to have the device on his desk.

Start J-Link Remote Server in tunneling mode



Connect to the J-Link / J-Trace via J-Link commander

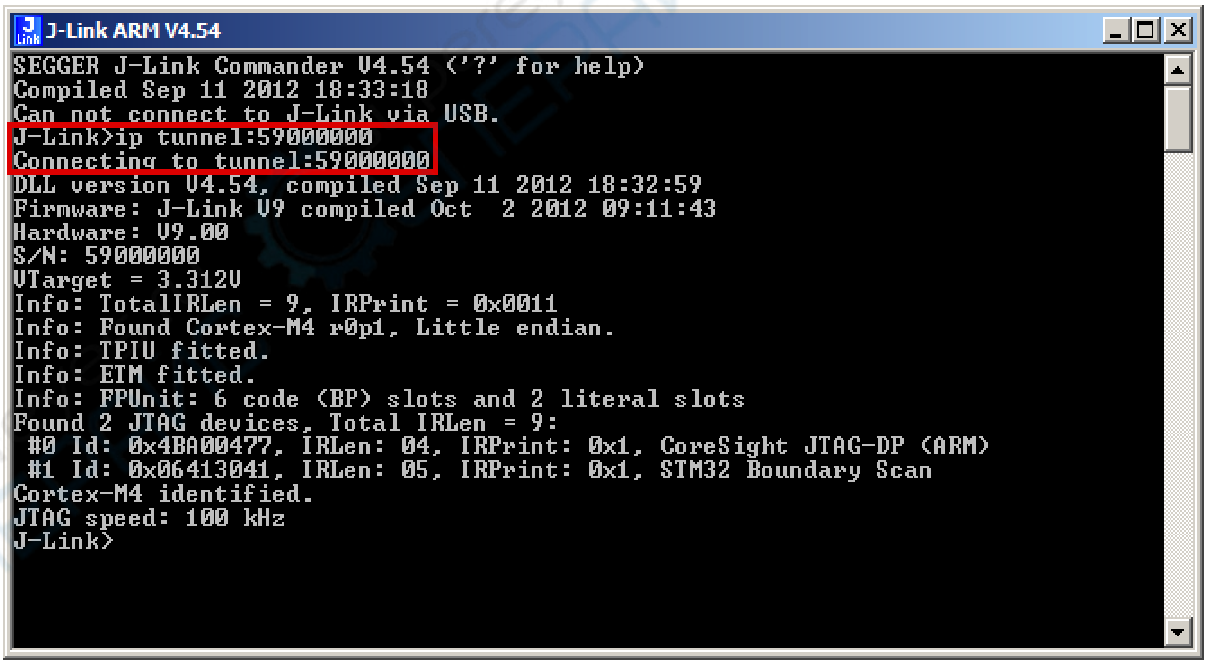
J-Link Commander can be used to verify a connection to the J-Link can be established as follows:

Start J-Link Commander

From within J-Link Commander enter

```
ip tunnel:<SerialNo>
```

If the connection was successful it should look like in this screenshot.



```
SEGGER J-Link Commander V4.54 ('?' for help)
Compiled Sep 11 2012 18:33:18
Can not connect to J-Link via USB.
J-Link>ip tunnel:59000000
Connecting to tunnel:59000000
DLL version V4.54, compiled Sep 11 2012 18:32:59
Firmware: J-Link V9 compiled Oct  2 2012 09:11:43
Hardware: V9.00
S/N: 59000000
VTarget = 3.312V
Info: TotalIRLen = 9, IRPrint = 0x0011
Info: Found Cortex-M4 r0p1, Little endian.
Info: TPIU fitted.
Info: ETM fitted.
Info: FPUnit: 6 code (BP) slots and 2 literal slots
Found 2 JTAG devices, Total IRLen = 9:
 #0 Id: 0x4BA00477, IRLen: 04, IRPrint: 0x1, CoreSight JTAG-DP (ARM)
 #1 Id: 0x06413041, IRLen: 05, IRPrint: 0x1, STM32 Boundary Scan
Cortex-M4 identified.
JTAG speed: 100 kHz
J-Link>
```

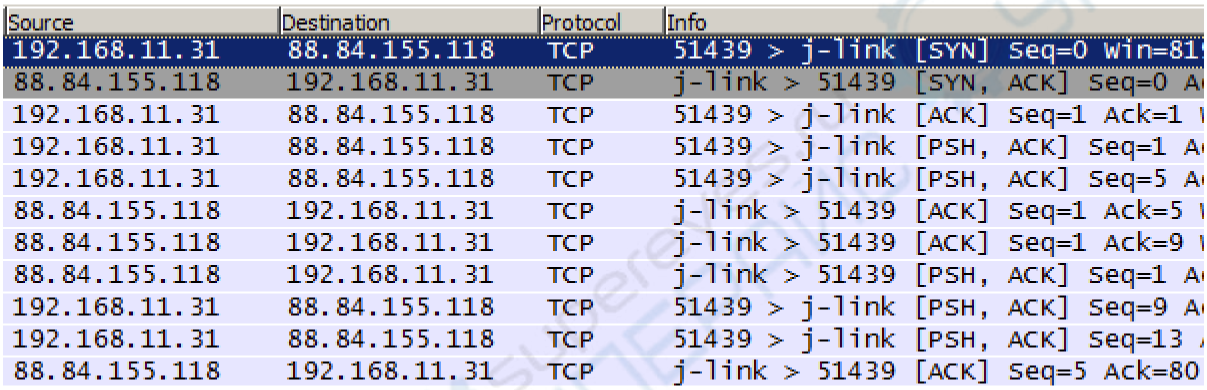
Troubleshooting

Problem	Solution
Remote server cannot connect to tunnel server	1. Make sure the Remote server is not blocked by any firewall. 2. Make sure port 19020 is not blocked by any firewall. 3. Contact network admin.
J-Link Commander cannot connect to tunnel server	1. Make sure Remote server is started correctly. 2. Make sure the entered serial number is correct. 3. Make sure port 19020 is not blocked by any firewall. Contact network admin.

Table 3.12:

To test whether a connection to the tunnel server can be established or not a network protocol analyzer like Wireshark can help.

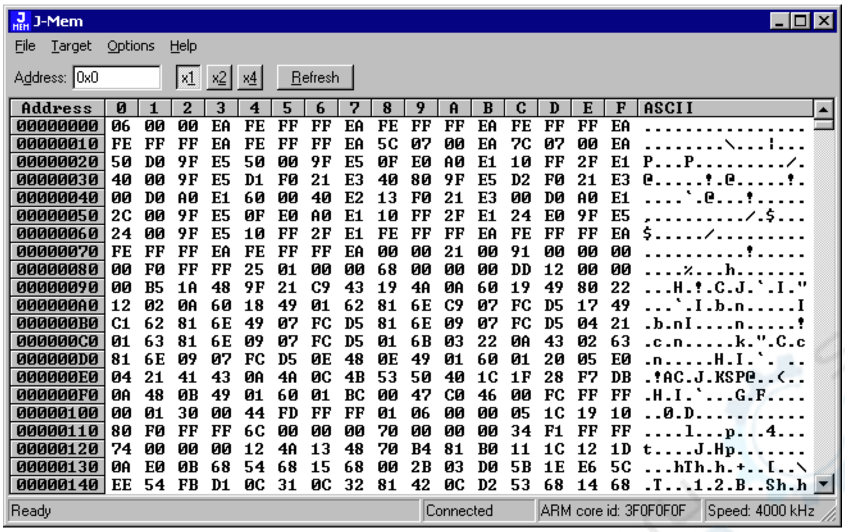
The network transfer of a successful connection should look like:



Source	Destination	Protocol	Info
192.168.11.31	88.84.155.118	TCP	51439 > j-link [SYN] Seq=0 Win=81
88.84.155.118	192.168.11.31	TCP	j-link > 51439 [SYN, ACK] Seq=0 A
192.168.11.31	88.84.155.118	TCP	51439 > j-link [ACK] Seq=1 Ack=1
192.168.11.31	88.84.155.118	TCP	51439 > j-link [PSH, ACK] Seq=1 A
192.168.11.31	88.84.155.118	TCP	51439 > j-link [PSH, ACK] Seq=5 A
88.84.155.118	192.168.11.31	TCP	j-link > 51439 [ACK] Seq=1 Ack=5
88.84.155.118	192.168.11.31	TCP	j-link > 51439 [ACK] Seq=1 Ack=9
88.84.155.118	192.168.11.31	TCP	j-link > 51439 [PSH, ACK] Seq=1 A
192.168.11.31	88.84.155.118	TCP	51439 > j-link [PSH, ACK] Seq=9 A
192.168.11.31	88.84.155.118	TCP	51439 > j-link [PSH, ACK] Seq=13
88.84.155.118	192.168.11.31	TCP	j-link > 51439 [ACK] Seq=5 Ack=80

3.5 J-Mem Memory Viewer

J-Mem displays memory contents of target 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 a target system 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.



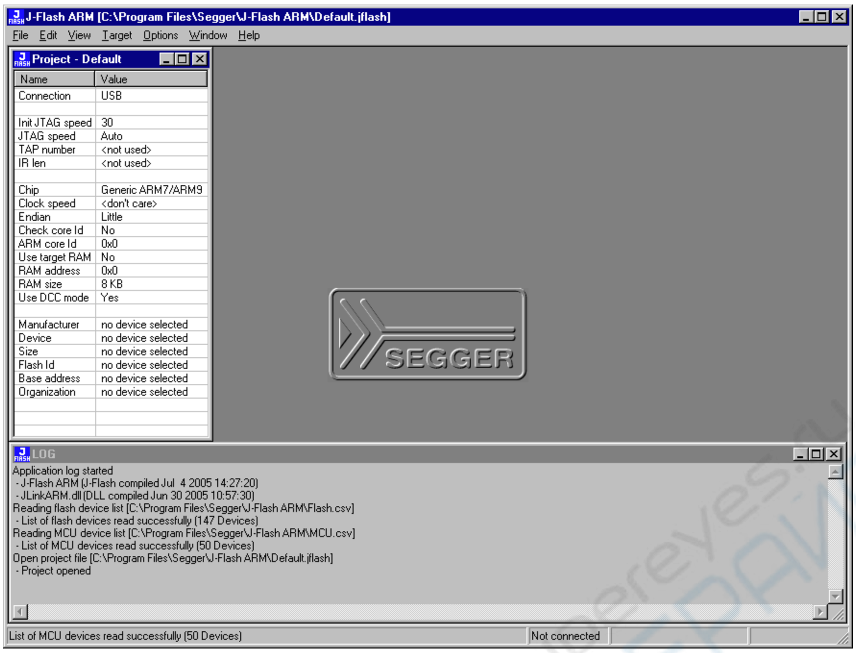
```
Address   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  ASCII
00000000 06 00 00 EA FE FF FF EA FE FF FF EA FE FF FF EA ................
00000010 FE FF FF EA FE FF FF EA 5C 07 00 EA 7C 07 00 EA ........\...|...
00000020 50 D0 9F E5 50 00 9F E5 0F E0 A0 E1 10 FF 2F E1 P...P........./.
00000030 40 00 9F E5 D1 F0 21 E3 40 80 9F E5 D2 F0 21 E3 @.....!.@.....!.
00000040 00 D0 A0 E1 60 00 40 E2 13 F0 21 E3 00 D0 A0 E1 ....`.@...!.....
00000050 2C 00 9F E5 0F E0 A0 E1 10 FF 2F E1 24 E0 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 F0 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 4A 0A 60 19 49 80 22 ...H.!.C.J.`.I."
000000A0 12 02 0A 60 18 49 01 62 81 6E C9 07 FC D5 17 49 ...`.I.b.n.....I
000000B0 C1 62 81 6E 49 07 FC D5 81 6E 09 07 FC D5 04 21 .b.nI....n.....!
000000C0 01 63 81 6E 09 07 FC D5 01 6B 03 22 0A 43 02 63 .c.n.....k.".C.c
000000D0 81 6E 09 07 FC D5 0E 48 0E 49 01 60 01 20 05 E0 .n.....H.I.`. ..
000000E0 04 21 41 43 0A 4A 0C 4B 53 50 40 1C 1F 28 F7 DB .!AC.J.KSP@..(..
000000F0 0A 48 0B 49 01 60 01 BC 00 47 C0 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 F0 FF FF 6C 00 00 00 70 00 00 00 34 F1 FF FF ....l...p...4...
00000120 74 00 00 00 12 4A 13 48 70 B4 81 B0 11 1C 12 1D t....J.Hp.......
00000130 0A E0 0B 68 54 68 15 68 00 2B 03 D0 5B 1E E6 5C ...hTh.h.+..[..\
00000140 EE 54 FB D1 0C 31 0C 32 81 42 0C D2 53 68 14 68 .T...1.2.B..Sh.h
```

Ready | Connected | ARM core id: 3F0F0F0F | Speed: 4000 kHz

3.6 J-Flash

J-Flash is an application to program data images to the flash of a target device. With J-Flash the internal flash of all J-Link supported devices can be programmed, as well as common external flashes connected to the device. Beside flash programming all other flash operations like erase, blank check and flash content verification can be done.

J-Flash requires an additional license from SEGGER to enable programming. For license keys, as well as evaluation licenses got to *http://www.segger.com* or contact us directly.



J-Flash ARM [C:\Program Files\Segger\J-Flash ARM\Default.jflash]
File Edit View Target Options Window Help
Project - Default
SEGGER
LOG

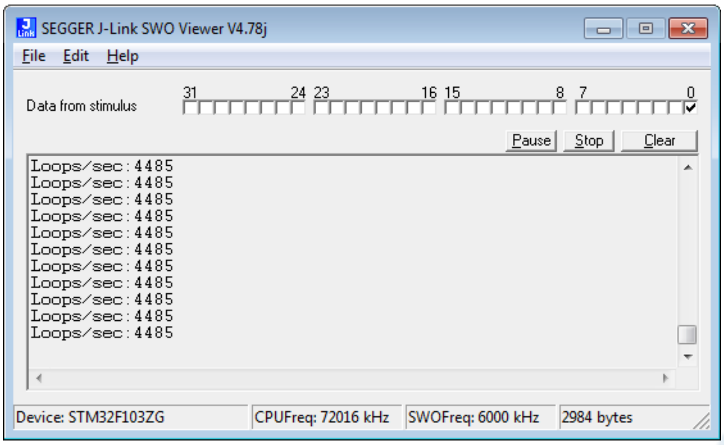
Name	Value
Connection	USB
Init JTAG speed	30
JTAG speed	Auto
TAP number	<not used>
IR len	<not used>
Chip	Generic ARM7/ARM9
Clock speed	<don't care>
Endian	Little
Check core Id	No
ARM core Id	0x0
Use target RAM	No
RAM address	0x0
RAM size	8 KB
Use DCC mode	Yes
Manufacturer	no device selected
Device	no device selected
Size	no device selected
Flash Id	no device selected
Base address	no device selected
Organization	no device selected

```
Application log started
 - J-Flash ARM (J-Flash compiled Jul  4 2005 14:27:20)
 - JLinkARM.dll (DLL compiled Jun 30 2005 10:57:30)
Reading flash device list [C:\Program Files\Segger\J-Flash ARM\Flash.csv]
 - List of flash devices read successfully (147 Devices)
Reading MCU device list [C:\Program Files\Segger\J-Flash ARM\MCU.csv]
 - List of MCU devices read successfully (50 Devices)
Open project file [C:\Program Files\Segger\J-Flash ARM\Default.jflash]
 - Project opened
```

List of MCU devices read successfully (50 Devices) | Not connected

3.7 J-Link SWO Viewer

Free-of-charge utility for J-Link. Displays the terminal output of the target using the SWO pin. The stimulus port(s) from which SWO data is received can be chosen by using the port checkboxes 0 to 31. Can be used in parallel with a debugger or stand-alone. This is especially useful when using debuggers which do not come with built-in support for SWO such as most GDB / GDB+Eclipse based debug environments.

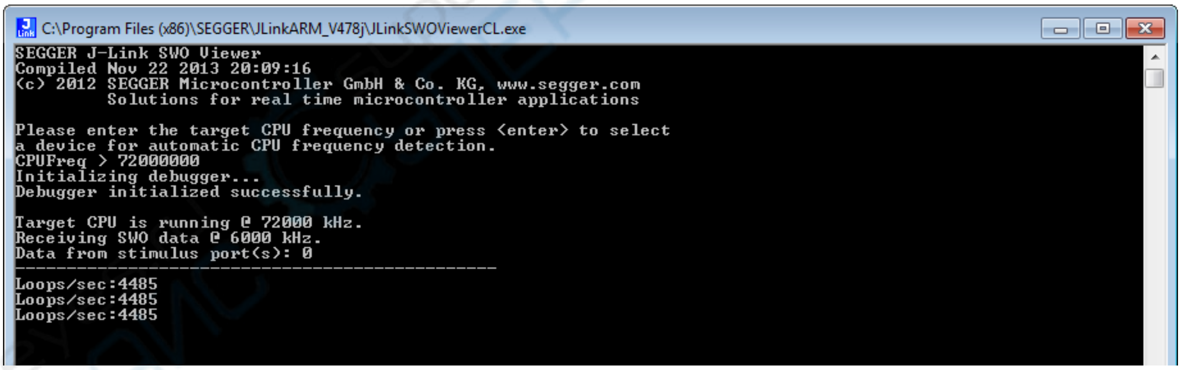


```
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
```

Device: STM32F103ZG | CPUFreq: 72016 kHz | SWOFreq: 6000 kHz | 2984 bytes

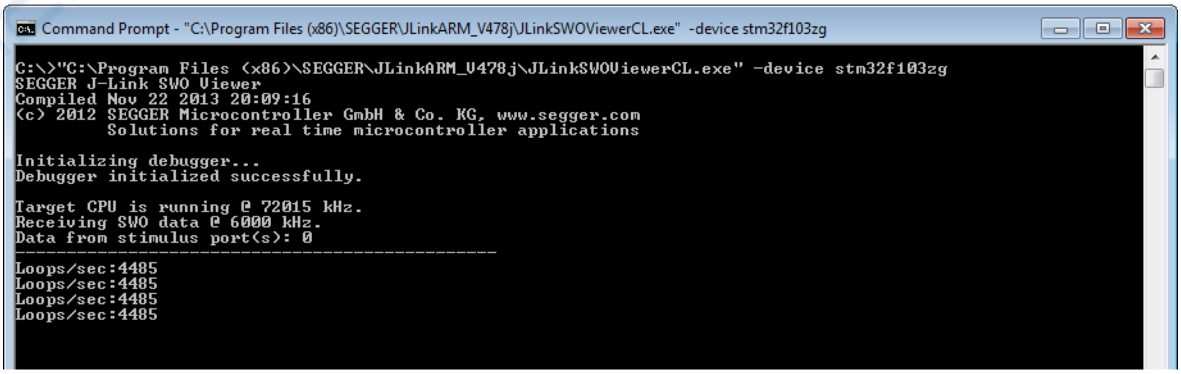
3.7.0.1 J-Link SWO Viewer CL

Commandline-only version of SWO Viewer. All commands avaible for J-Link SWO Viewer can be used with J-Link SWO Viewer Cl. Similar to the GUI Version,J-Link SWO Viewer Cl asks for a device name or CPU clock speed at startup to be able to calculate the correct SWO speed or to connect to a running J-Link GDB Server



```
SEGGER J-Link SWO Viewer
Compiled Nov 22 2013 20:09:16
(c) 2012 SEGGER Microcontroller GmbH & Co. KG, www.segger.com
         Solutions for real time microcontroller applications

Please enter the target CPU frequency or press <enter> to select
a device for automatic CPU frequency detection.
CPUFreq > 72000000
Initializing debugger...
Debugger initialized successfully.

Target CPU is running @ 72000 kHz.
Receiving SWO data @ 6000 kHz.
Data from stimulus port(s): 0
-----------------------------------------------
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
```

Using the syntax given below(List of available command line options), you can directly start J-Link SWO Viewer Cl with parameters.

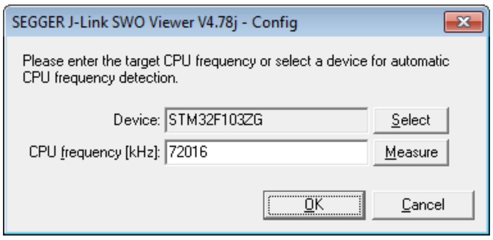

Command Prompt - "C:\Program Files (x86)\SEGGER\JLinkARM_V478j\JLinkSWOViewerCL.exe" -device stm32f103zg

```
C:\>"C:\Program Files (x86)\SEGGER\JLinkARM_V478j\JLinkSWOViewerCL.exe" -device stm32f103zg
SEGGER J-Link SWO Viewer
Compiled Nov 22 2013 20:09:16
(c) 2012 SEGGER Microcontroller GmbH & Co. KG, www.segger.com
         Solutions for real time microcontroller applications

Initializing debugger...
Debugger initialized successfully.

Target CPU is running @ 72015 kHz.
Receiving SWO data @ 6000 kHz.
Data from stimulus port(s): 0
-----------------------------------------------
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
Loops/sec:4485
```

3.7.1 Usage

J-Link SWO Viewer is available via the start menu.

It asks for a device name or CPU clock speed at startup to be able to calculate the correct SWO speed or to connect to a running J-Link GDB Server.



SEGGER J-Link SWO Viewer V4.78j - Config

Please enter the target CPU frequency or select a device for automatic CPU frequency detection.

Device: STM32F103ZG

CPU frequency [kHz]: 72016

When running in normal mode J-Link SWO Viewer automatically performs the necessary initialization to enable SWO output on the target, in GDB Server mode the initialization has to be done by the debugger.

3.7.2 List of available command line options

J-Link SWO Viewer can also be controlled from the command line if used in a automated test environment etc.

When passing all necessary information to the utility via command line, the configuration dialog at startup is suppressed. Minimum information needed by J-Link SWO Viewer is the device name (to enable CPU frequency auto detection) or the CPU clock speed.

The table below lists the commands accepted by the J-Link SWO Viewer.

Command	Description
cpufreq	Select the CPU frequency.
device	Select the target device.
itmmask	Selects a set of itm stimulus ports which should be used to listen to.
itmport	Selects a itm stimulus port which should be used to listen to.
outputfile	Print the output of SWO Viewer to the selected file.
settingsfile	Specify a J-Link settings file.
swofreq	Select the CPU frequency.

Table 3.13: Available command line options

3.7.2.1 cpufreq

Defines the speed in Hz the CPU is running at. If the CPU is for example running at 96 MHz, the command line should look as below.

Syntax

```
-cpufreq <CPUFreq>
```

Example

```
-cpufreq 96000000
```

3.7.2.2 device

Select the target device to enable the CPU frequency auto detection of the J-Link DLL. To select a ST STM32F207IG as target device, the command line should look as below.

For a list of all supported device names, please refer to <Ref>

Syntax

```
-device <DeviceID>
```

Example

```
-deivce STM32F207IG
```

3.7.2.3 itmmask

Defines a set of stimulusports from which SWO data is received and displayed by SWO Viewer.

If itmmask is given, itmport will be ignored.

Syntax

```
-itmmask <Mask>
```

Example

Listen on ports 0 and 2

```
-itmmask 0x5
```

3.7.2.4 itmport

Defines the stimulus port from which SWO data is received and displayed by the SWO Viewer. Default is stimulus port 0. The command line should look as below.

Syntax

```
-itmport <ITMPortIndex>
```

Example

```
-itmport 0
```

3.7.2.5 outputfile

Define a file to which the output of SWO Viewer is printed.

Syntax

```
-outputfile <PathToFile>
```

Example

```
-outputfile "C:\Temp\Output.log"
```

3.7.2.6 settingsfile

Select a J-Link settings file to use for the target device.

Syntax

```
-settingsfile <PathToFile>
```

Example

```
-settingsfile "C:\Temp\Settings.jlink"
```

3.7.2.7 swofreq

Define the SWO frequency that shall be used by J-Link SWO Viewer for sampling SWO data.

Usually not necessary to define since optimal SWO speed is calculated automatically based on the CPU frequency and the capabilities of the connected J-Link.

Syntax

```
-swofreq <SWOFreq>
```

Example

```
-swofreq 6000
```

3.7.3 Configure SWO output after device reset

In some situations it might happen that the target application is reset and it is desired to log the SWO output of the target after reset during the booting process. For such situations, the target application itself needs to initialize the CPU for SWO output, since the SWO Viewer is not restarted but continuously running.

Example code for enabling SWO out of the target application

```
#define ITM_ENA    (*(volatile unsigned int*)0xE0000E00) // ITM Enable
#define ITM_TPR    (*(volatile unsigned int*)0xE0000E40) // Trace Privilege Register
#define ITM_TCR    (*(volatile unsigned int*)0xE0000E80) // ITM Trace Control Reg.
#define ITM_LSR    (*(volatile unsigned int*)0xE0000FB0) // ITM Lock Status Register
#define DHCSR      (*(volatile unsigned int*)0xE000EDF0) // Debug register
#define DEMCR      (*(volatile unsigned int*)0xE000EDFC) // Debug register
#define TPIU_ACPR  (*(volatile unsigned int*)0xE0040010) // Async Clock         \
                                                         // presacler register
#define TPIU_SPPR  (*(volatile unsigned int*)0xE00400F0) // Selected Pin Protocol  \
                                                         // Register
#define DWT_CTRL   (*(volatile unsigned int*)0xE0001000) // DWT Control Register
#define FFCR       (*(volatile unsigned int*)0xE0040304) // Formatter and flush    \
                                                         // Control Register
U32 _ITMPort = 0; // The stimulus port from which SWO data is received and displayed.
U32 TargetDiv = 1;// Has to be calculated according to                           \
                  // the CPU speed and the output baud rate

static void _EnableSWO() {
  U32 StimulusRegs;
  //
  // Enable access to SWO registers
  //
  DEMCR |= (1 << 24);
  ITM_LSR = 0xC5ACCE55;
  //
  // Initially disable ITM and stimulus port
  // To make sure that nothing is transferred via SWO
  // when changing the SWO prescaler etc.
  //
  Stimulusregs = ITM_ENA;
  StimulusRegs &= ~(1 << _ITMPort);
  ITM_ENABLE = StimulusRegs;                             // Disable ITM stimulus port
  ITM_TCR    = 0;                                        // Disable ITM
  //
  // Initialize SWO (prescaler, etc.)
  //
  TPIU_SPPR  = 0x00000002;                               // Select NRZ mode
  TPIU_ACPR  = TargetDiv - 1;                            // Example: 72/48 = 1,5 MHz
  ITM_TPR    = 0x00000000;
  DWT_CTRL   = 0x400003FE;
  FFCR       = 0x00000100;
  //
  // Enable ITM and stimulus port
  //
  ITM_TCR     = 0x1000D;                                 // Enable ITM
  ITM_ENA     = StimulusRegs | (1 << _ITMPort));         // Enable ITM stimulus port
}
```

3.7.4 Target example code for terminal output

```
/*********************************************************************
*                SEGGER MICROCONTROLLER GmbH & Co KG                 *
*        Solutions for real time microcontroller applications        *
**********************************************************************
*                                                                    *
*        (c) 2012-2013  SEGGER Microcontroller GmbH & Co KG          *
*                                                                    *
*        www.segger.com     Support: support@segger.com              *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File    : SWO.c
Purpose : Simple implementation for output via SWO for Cortex-M processors.
          It can be used with any IDE. This sample implementation ensures that
          output via SWO is enabled in order to gurantee that the application
          does not hang.

--------  END-OF-HEADER  ---------------------------------------------
*/

/*********************************************************************
*
*       Prototypes (to be placed in a header file such as SWO.h)
*/
void SWO_PrintChar  (char c);
void SWO_PrintString(const char *s);

/*********************************************************************
*
*       Defines for Cortex-M debug unit
*/
#define ITM_STIM_U32 (*(volatile unsigned int*)0xE0000000) // STIM word access
#define ITM_STIM_U8  (*(volatile         char*)0xE0000000) // STIM Byte access
#define ITM_ENA      (*(volatile unsigned int*)0xE0000E00) // ITM Enable Reg.
#define ITM_TCR      (*(volatile unsigned int*)0xE0000E80) // ITM Trace Control Reg.

/*********************************************************************
*
*       SWO_PrintChar()
*
* Function description
*   Checks if SWO is set up. If it is not, return,
*    to avoid program hangs if no debugger is connected.
*   If it is set up, print a character to the ITM_STIM register
*    in order to provide data for SWO.
* Parameters
*   c:    The Chacracter to be printed.
* Notes
*   Additional checks for device specific registers can be added.
*/
void SWO_PrintChar(char c) {
  //
  // Check if ITM_TCR.ITMENA is set
  //
  if ((ITM_TCR & 1) == 0) {
    return;
  }
  //
  // Check if stimulus port is enabled
  //
  if ((ITM_ENA & 1) == 0) {
    return;
  }
  //
  // Wait until STIMx is ready,
  // then send data
  //
  while ((ITM_STIM_U8 & 1) == 0);
  ITM_STIM_U8 = c;
}
```

```
/*********************************************************************
*
*       SWO_PrintString()
*
* Function description
*   Print a string via SWO.
*
*/
void SWO_PrintString(const char *s) {
  //
  // Print out character per character
  //
  while (*s) {
    SWO_PrintChar(*s++);
  }
}
```

3.8 J-Scope

J-Scope is a free-of-charge software to analyze and visualize data on a microcontroller in real-time, while the target is running.

3.8.1 Getting Started

J-Scope is part of the J-Link Software and Documentation Package (JScope.exe).

Preparation

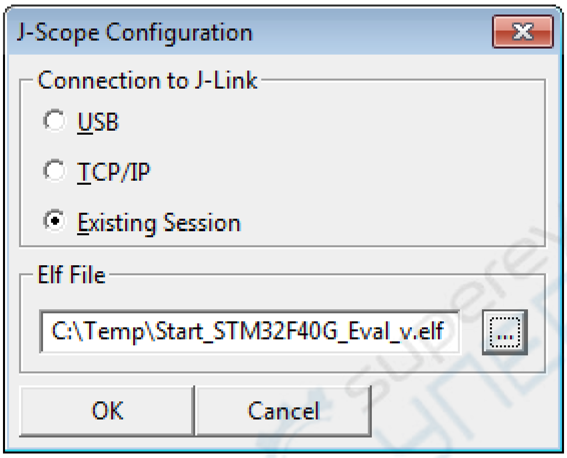
Before starting J-Scope make sure you have an elf-file of your target application and the target is running the application. The application can either be flashed with J-Flash or directly ran from an IDE.

Start J-Scope by double-clicking on the executable.

The Project Configuration Dialog will open.

3.8.1.1 Set up Connection

Attach to a running instance



J-Scope Configuration

Connection to J-Link

○ USB

○ TCP/IP

● Existing Session

Elf File

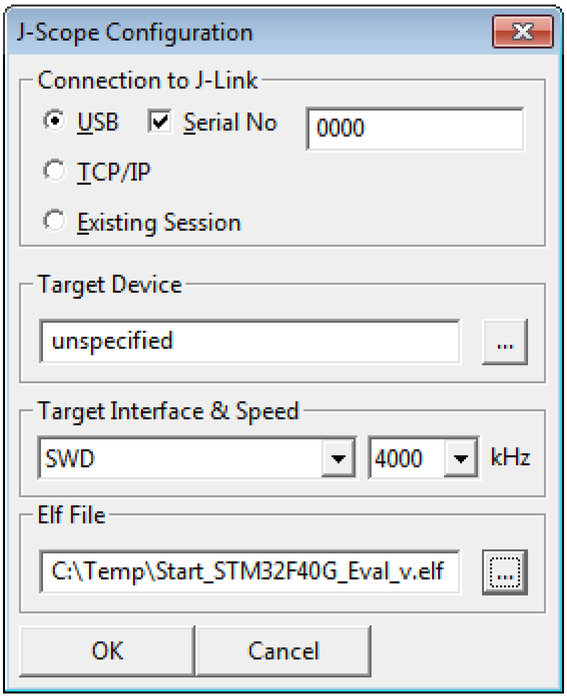
C:\Temp\Start_STM32F40G_Eval_v.elf

J-Scope can be used alongside other applications, e.g. a debugger, which are connected to J-Link and share the connection to the J-Link. If J-Scope is used in this mode select "Existing Session" as connection type.

All Target options will be disabled, as they are already determined by the running session.

Note: In attach mode J-Scope will connect to J-Link indirectly via another application connected to J-Link. J-Scope can only be used while the connection is active, e.g. a debug session is running. It is the applications responsibility to set up the target and J-Link connection and to manage target execution.

Connecting to J-Link


J-Scope Configuration

Connection to J-Link

● USB ☑ Serial No 0000

○ TCP/IP

○ Existing Session

Target Device: unspecified

Target Interface & Speed: SWD, 4000 kHz

Elf File: C:\Temp\Start_STM32F40G_Eval_v.elf

If J-Scope is used in standalone mode, opening a dedicated connection to J-Link, choose "USB" or "TCP/IP" as connection type and optionally enter the serial number or the IP of the J-Link to connect to a specific J-Link.

For most targets, the target configuration does not have to be specified. If there are any problems, select your target device and configure the interface accordingly.

3.8.1.2 Select application file and symbols

In the configuration dialog, select your elf-file and click OK. The Symbol Selection Dialog will open.

Select the symbols of your application, you want to watch and analyze. Click OK to finalize the setup.

3.8.1.3 Start sampling

Data sampling can be started via the record-button (red dot) in the toolbar. On start of sampling, the connection to J-Link or running session will be established.

3.8.2 Project Files

The project configuration, which are the J-Link connection settings, the elf-application file and the configuration of the selected symbols can be saved to a J-Scope project file (.jscope).

J-Scope projects can be used for further sessions and exchanged between different users.

3.8.3 Exporting Sampled Data

3.8.3.1 Export to CSV

The sampled data which is currently in the buffer can be exported into CSV format, to be opened and analyzed by other tools, for example spreadsheet programs like MS Excel, Apple Numbers or Libre Office Calc, by pressing File -> Export Data.

3.8.4 Symbol Configuration

The symbol context menu can be accessed by right-clicking the corresponding row in the watch below the graph.

From the context menu the Y-resolution, visibility and color of the graph, as well as the display as hexadecimal or decimal can be configured.

When sampling is stopped, symbols can be removed or added via the context menu.

3.8.5 Short Cuts

The Y-Resolution of a symbol can be changed be selecting the respective row in the watch and using the "+" and "-" keys.

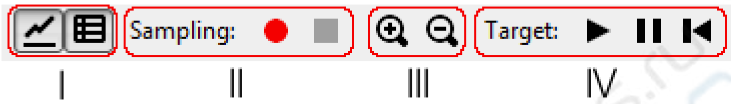
The visibility of a symbol can be changed by pressing "enter".

Scrolling through the Graph Window can be done with the mouse wheel.

Holding control while using the mouse wheel will zoom in and out the time base.

3.8.6 GUI Meters and Controls

3.8.6.1 Toolbar



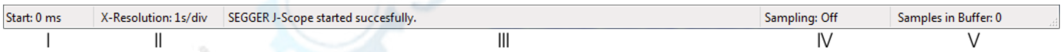
I. Show/Hide the Graph | Show/Hide Watch Window

II. Start sampling | Stop sampling | Pause / Resume sampling

III. Zoom in | Zoom out

IV. Start the target | Halt the target | Reset the target

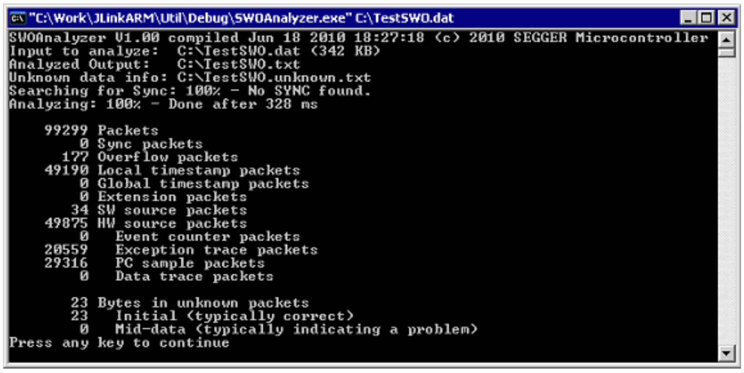
3.8.6.2 Statusbar


Start: 0 ms | X-Resolution: 1s/div | SEGGER J-Scope started succesfully. | Sampling: Off | Samples in Buffer: 0

I. Start: Timestamp of the 1st value currently shown in the graph.
II. X-Resolution: Displayed time per division.
III. Status: Shows the lastest status message
IV. Sampling: Indicates whether HSS is currently running or not
V. Samples in Buffer: Shows the current amount of datapoints stored in buffer

3.9 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.


"C:\Work\JLinkARM\Util\Debug\SWOAnalyzer.exe" C:\TestSWO.dat

```
SWOAnalyzer V1.00 compiled Jun 18 2010 18:27:18 (c) 2010 SEGGER Microcontroller
Input to analyze:  C:\TestSWO.dat (342 KB)
Analyzed Output:   C:\TestSWO.txt
Unknown data info: C:\TestSWO.unknown.txt
Searching for Sync: 100% - No SYNC found.
Analyzing: 100% - Done after 328 ms

    99299 Packets
        0 Sync packets
      177 Overflow packets
    49190 Local timestamp packets
        0 Global timestamp packets
        0 Extension packets
       34 SW source packets
    49875 HW source packets
        0   Event counter packets
    20559   Exception trace packets
    29316   PC sample packets
        0   Data trace packets

       23 Bytes in unknown packets
       23   Initial (typically correct)
        0   Mid-data (typically indicating a problem)
Press any key to continue
```

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.

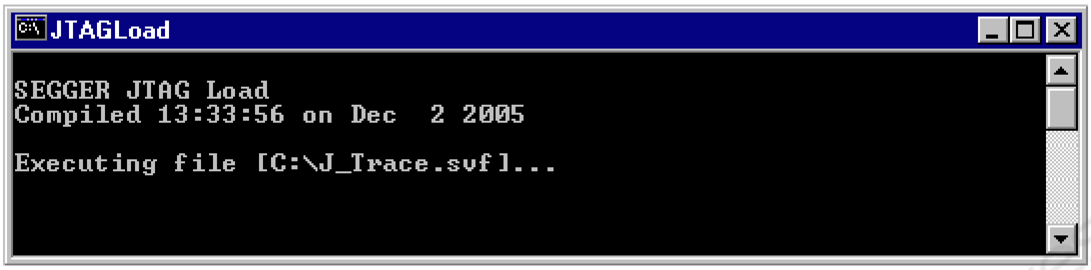
It should contain the following lines:

```
[SWO]
```

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

3.10 JTAGLoad (Command line tool)

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



```
SEGGER JTAG Load
Compiled 13:33:56 on Dec  2 2005

Executing file [C:\J_Trace.svf]...
```

SVF is a standard format for boundary scan vectors to be used with different tools and targets. SVF files contain human-readable ASCII SVF statements consisting of an SVF command, the data to be sent, the expected response, a mask for the response or additional information.

JTAGLoad supports following SVF commands:

- ENDDR
- ENDIR
- FREQUENCY
- HDR
- HIR
- PIOMAP
- PIO
- RUNTEST
- SDR
- SIR
- STATE
- TDR
- TIR

A simple SVF file to read the JTAG ID of the target can look like following:

```
! Set JTAG frequency
FREQUENCY 12000000HZ;
! Configure scan chain
! For a single device in chain, header and trailer data on DR and IR are 0
! Set TAP to IDLE state
STATE IDLE;
! Configure end state of DR and IR after scan operations
ENDDR IDLE;
ENDIR IDLE;
! Start of test
! 32 bit scan on DR, In: 32 0 bits, Expected out: Device ID (0x0BA00477)
SDR 32 TDI (0) TDO (0BA00477) MASK (0FFFFFFF);
! Set TAP to IDLE state
STATE IDLE;
! End of test
```

SVD files allow even more complex tasks, basically everything which is possible via JTAG and the devices in the scan chain, like configuring an FPGA or loading data into memory.

3.11 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.

3.11.1 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*.

3.12 Processor specific tools

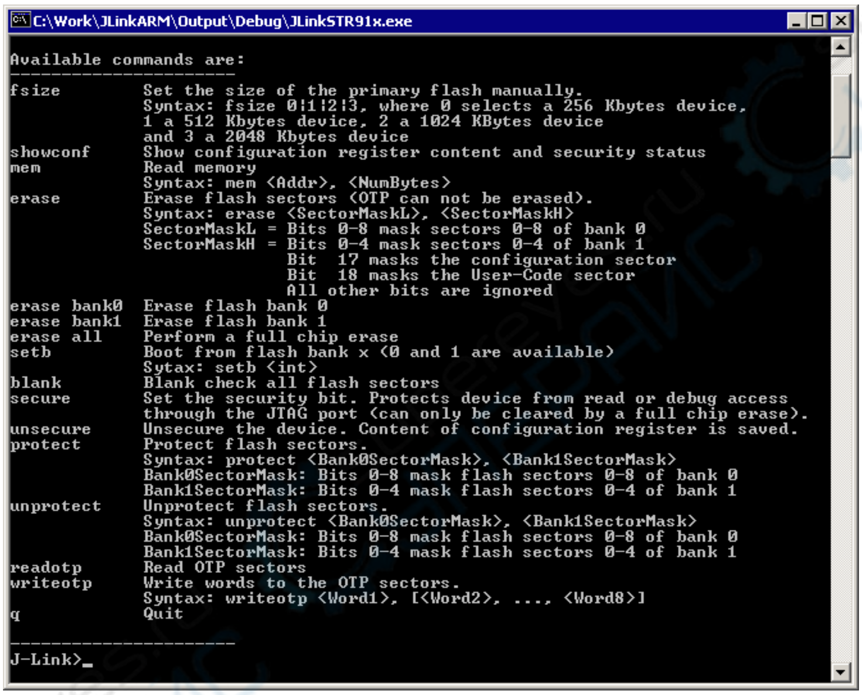
The J-Link software and documentation package includes some tools which support processor specific functionalities, like unlocking a device.

3.12.1 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 CPU core to stall.



```
Available commands are:
-----------------------
fsize        Set the size of the primary flash manually.
             Syntax: fsize 0|1|2|3, where 0 selects a 256 Kbytes device,
             1 a 512 Kbytes device, 2 a 1024 KBytes device
             and 3 a 2048 Kbytes device
showconf     Show configuration register content and security status
mem          Read memory
             Syntax: mem <Addr>, <NumBytes>
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  17 masks the configuration sector
                           Bit  18 masks the User-Code sector
                           All other bits are ignored
erase bank0  Erase flash bank 0
erase bank1  Erase flash bank 1
erase all    Perform a full chip erase
setb         Boot from flash bank x (0 and 1 are available)
             Sytax: 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     Unsecure the device. Content of configuration register is saved.
protect      Protect flash sectors.
             Syntax: protect <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
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>]
q            Quit

-----------------------
J-Link>_
```

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.

3.12.1.1 Command line options

J-Link STR91x Commander can be started with different command line options. In the following, the command line options which are available for J-Link STR91x Commander are explained.

-CommandFile

Selects a command file and starts J-Link STR91x Commander in batch mode. The batch mode of J-Link Commander is similar to the execution of a batch file. The command file is parsed line by line and one command is executed at a time.

Syntax

```
-CommanderScript <CommandFilePath>
```

Example

See *Using command files* on page 64

-DRPre, -DRPost, -IRPre and -IRPost (Scan-Chain Configuration)

STR91x allows to configure a specific scan-chain via command-line. To use this feature four command line options has to be specified in order to allow a proper connection to the proper device. In case of passing an incomplete configuration, the utility tries to auto-detect.

Syntax

```
-DRPre <DRPre>
-DRPost <DRPost>
-IRPre <IRPre>
-IRPost <IRPost>
```

Example

```
JLink.exe -DRPre 1 -DRPost 4 -IRPre  16  -IRPost 20
```

-IP

Selects IP as host interface to connect to J-Link. Default host interface is USB.

Syntax

```
-IP <IPAddr>
```

Example

```
JLinkSTR91x.exe -IP 192.168.1.17
```

Additional information

To select from a list of all available emulators on Ethernet, please use `*` as `<IPAddr>`.

-SelectEmuBySN

Connect to a J-Link with a specific serial number via USB. Useful if multiple J-Links are connected to the same PC and multiple instances of J-Link STR91x Commander shall run and each connects to another J-Link.

Syntax

```
-SelectEmuBySN <SerialNo>
```

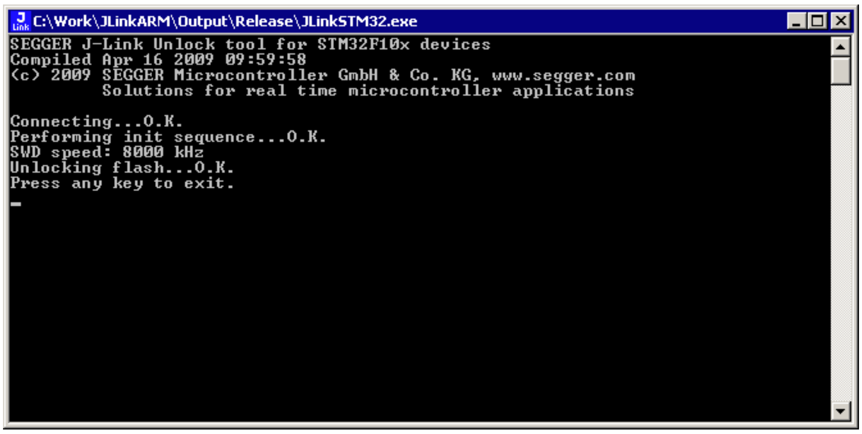
Example

```
JLinkSTR91x.exe -SelectEmuBySN 580011111
```

3.12.2 J-Link STM32 Unlock (Command line tool)

J-Link STM32 Unlock (`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.



```
SEGGER J-Link Unlock tool for STM32F10x devices
Compiled Apr 16 2009 09:59:58
(c) 2009 SEGGER Microcontroller GmbH & Co. KG, www.segger.com
         Solutions for real time microcontroller applications

Connecting...O.K.
Performing init sequence...O.K.
SWD speed: 8000 kHz
Unlocking flash...O.K.
Press any key to exit.
```

3.13 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 halting/stepping the CPU core, reading/writing CPU and ICE registers and reading/writing memory. Therefore it can be used in any kind of application accessing an CPU 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.

The J-Link SDK requires an additional license and is available upon request from *www.segger.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 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*.

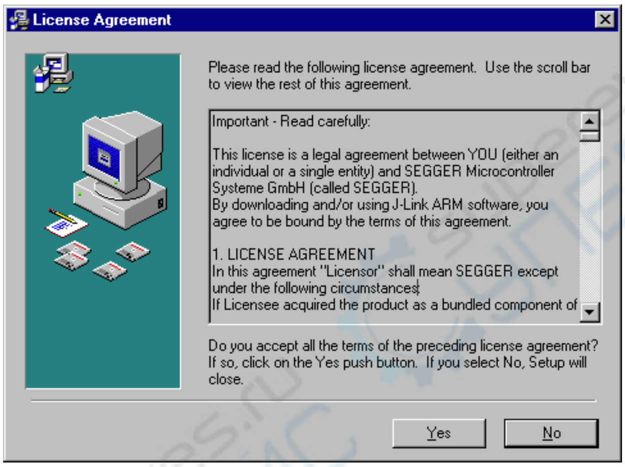
Refer to chapter *J-Link software and documentation package* on page 61 for an overview about the J-Link software and documentation pack.

4.1.1 Setup procedure

To install the J-Link 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.segger.com/download_jlink.html

1. 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<VersionNumber>.exe`. The **license Agreement** dialog box will be opened. Accept the terms with the **Yes** button.



License Agreement

Please read the following license agreement. Use the scroll bar to view the rest of this agreement.

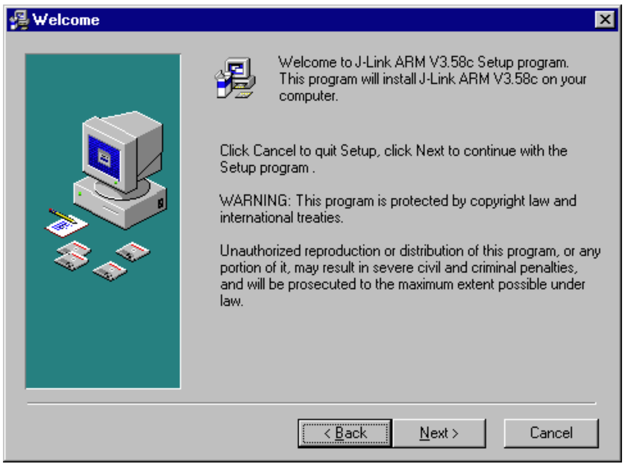
Important - Read carefully:

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 circumstances:
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.

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



Welcome

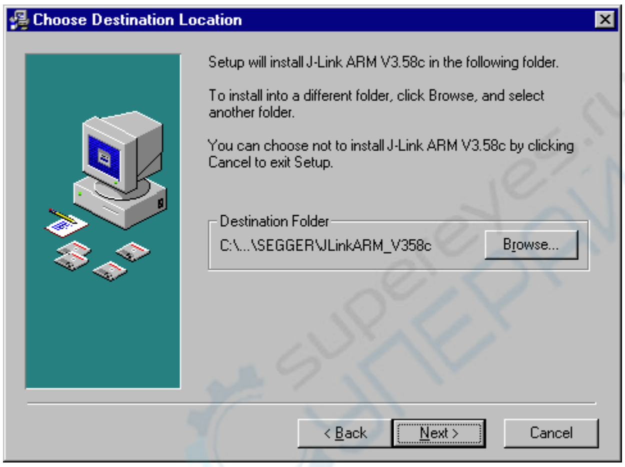
Welcome to J-Link ARM V3.58c Setup program. This program will install J-Link ARM V3.58c on your computer.

Click Cancel to quit Setup, click Next to continue with the Setup program .

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

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



Choose Destination Location

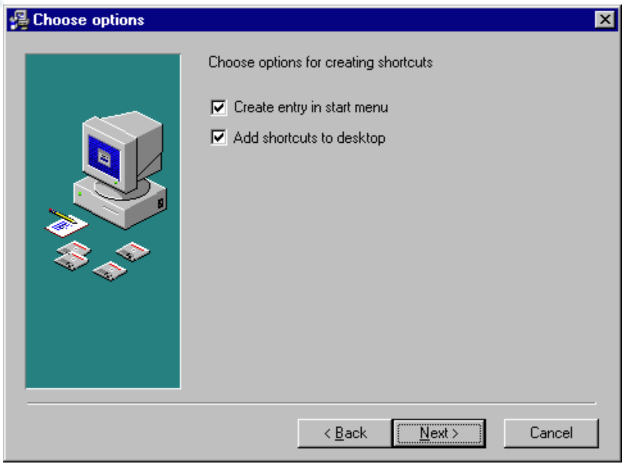
Setup will install J-Link ARM V3.58c in the following folder.

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

4. The **Choose options** dialog is opened.
The **Install J-Link Serial Port Driver** installs the driver for J-Links with CDC functionality. It is not preselected since J-Links without CDC functionality do not need this driver.
The **Create entry in start menu** creates a entry in start menu. It is preselected.
The **Add shortcuts to desktop** option can be selected in order to create a shortcut on the desktop.
Accept or deselect the options and confirm the selection with the **Next >** button.

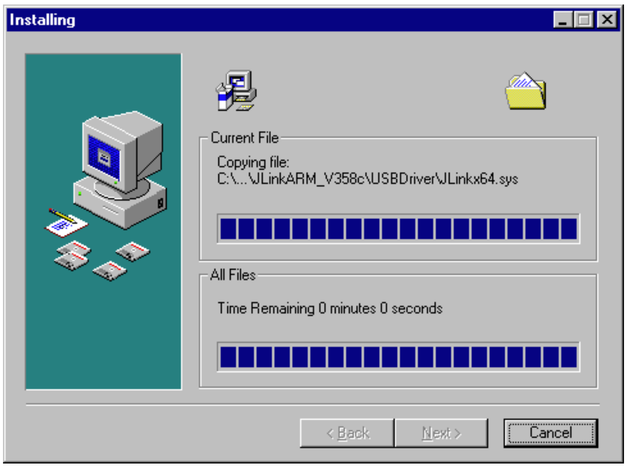


Choose options

Choose options for creating shortcuts

☑ Create entry in start menu
☑ Add shortcuts to desktop

5. The installation process will be started.



Installing

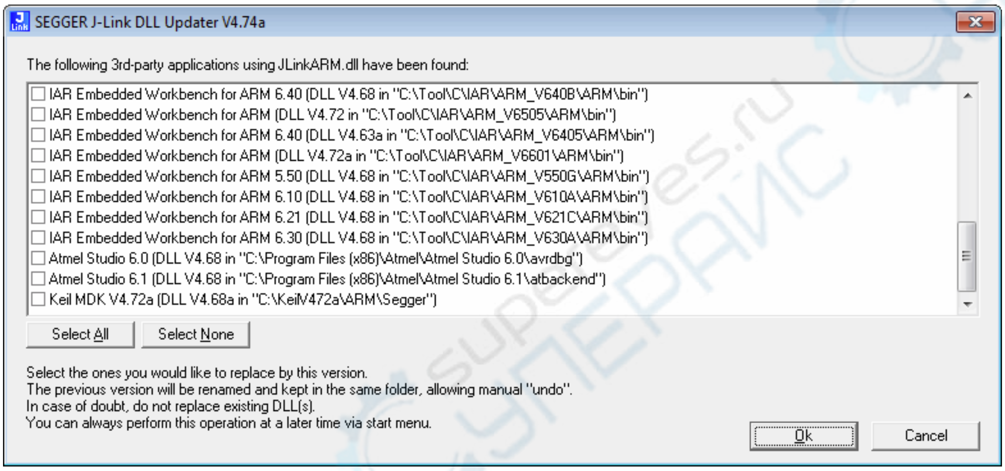
Current File

Copying file:
C:\...\JLinkARM_V358c\USBDriver\JLinkx64.sys

All Files

Time Remaining 0 minutes 0 seconds

6. The J-Link DLL Updater pops up, which allows you to update the DLL of an installed IDE to the DLL verion which is included in the installer. For furhter information about the J-Link DLL updater, please refer to *J-Link DLL updater* on page 135.



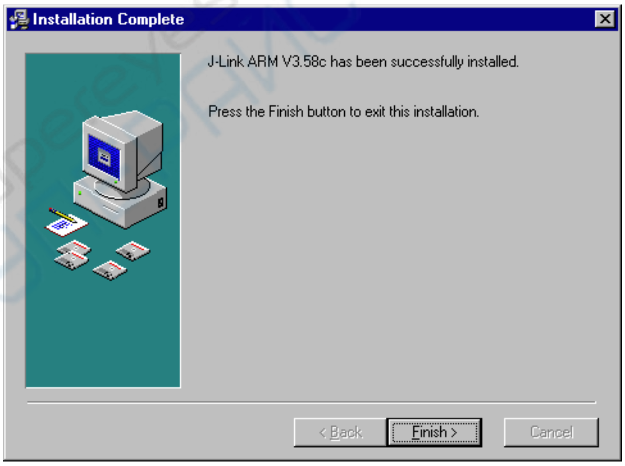
SEGGER J-Link DLL Updater V4.74a

The following 3rd-party applications using JLinkARM.dll have been found:

□ IAR Embedded Workbench for ARM 6.40 (DLL V4.68 in "C:\Tool\C\IAR\ARM_V640B\ARM\bin")
□ IAR Embedded Workbench for ARM (DLL V4.72 in "C:\Tool\C\IAR\ARM_V6505\ARM\bin")
□ IAR Embedded Workbench for ARM 6.40 (DLL V4.63a in "C:\Tool\C\IAR\ARM_V6405\ARM\bin")
□ IAR Embedded Workbench for ARM (DLL V4.72a in "C:\Tool\C\IAR\ARM_V6601\ARM\bin")
□ IAR Embedded Workbench for ARM 5.50 (DLL V4.68 in "C:\Tool\C\IAR\ARM_V550G\ARM\bin")
□ IAR Embedded Workbench for ARM 6.10 (DLL V4.68 in "C:\Tool\C\IAR\ARM_V610A\ARM\bin")
□ IAR Embedded Workbench for ARM 6.21 (DLL V4.68 in "C:\Tool\C\IAR\ARM_V621C\ARM\bin")
□ IAR Embedded Workbench for ARM 6.30 (DLL V4.68 in "C:\Tool\C\IAR\ARM_V630A\ARM\bin")
□ Atmel Studio 6.0 (DLL V4.68 in "C:\Program Files (x86)\Atmel\Atmel Studio 6.0\avrdbg")
□ Atmel Studio 6.1 (DLL V4.68 in "C:\Program Files (x86)\Atmel\Atmel Studio 6.1\atbackend")
□ Keil MDK V4.72a (DLL V4.68a in "C:\Keil\V472a\ARM\Segger")

Select the ones you would like to replace by this version.
The previous version will be renamed and kept in the same folder, allowing manual "undo".
In case of doubt, do not replace existing DLL(s).
You can always perform this operation at a later time via start menu.

7. 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.

The J-Link software and documentation pack is successfully installed on your PC.

8. 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.

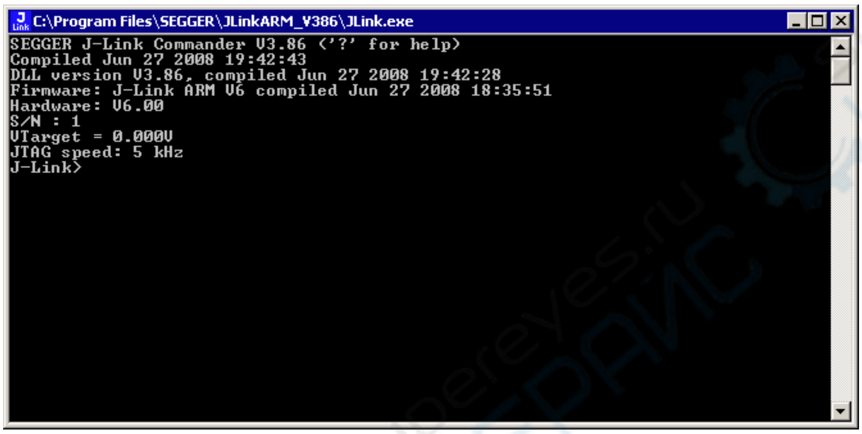
4.2 Setting up the USB interface

After installing the J-Link 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

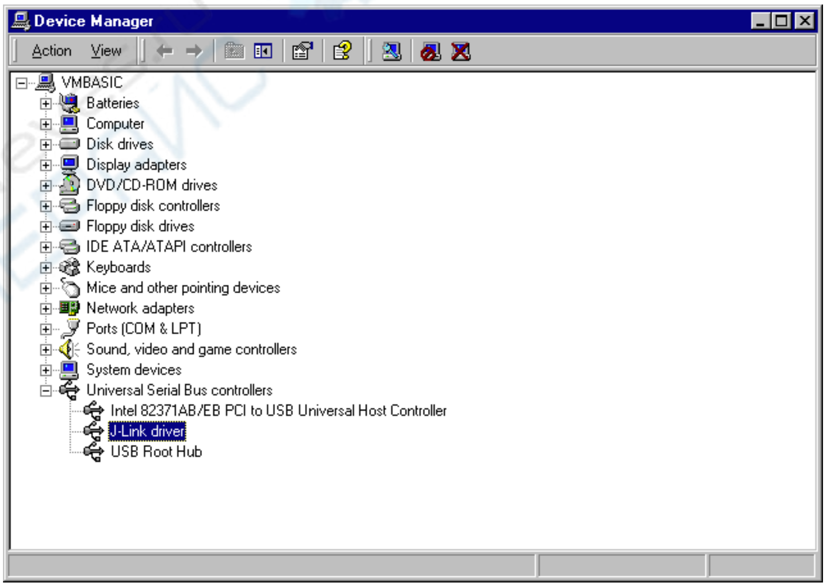
To verify the correct installation of the driver, disconnect and reconnect J-Link / J-Trace to the USB port. During the enumeration process which takes about 2 seconds, the LED on J-Link / J-Trace is flashing. After successful enumeration, the LED stays on permanently.

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.

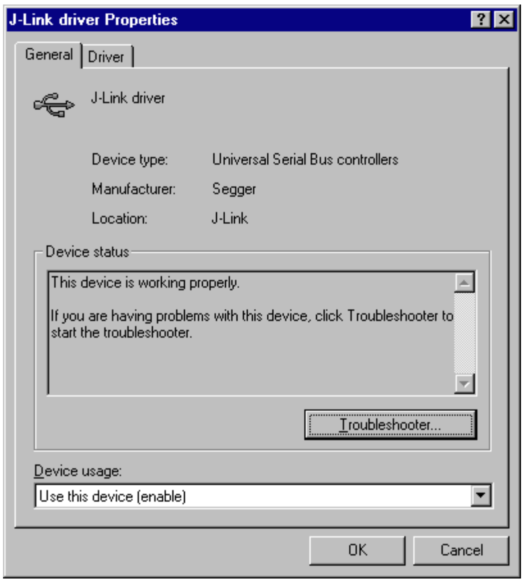


```
SEGGER J-Link Commander V3.86 ('?' for help)
Compiled Jun 27 2008 19:42:43
DLL version V3.86, compiled Jun 27 2008 19:42:28
Firmware: J-Link ARM V6 compiled Jun 27 2008 18:35:51
Hardware: V6.00
S/N : 1
VTarget = 0.000V
JTAG speed: 5 kHz
J-Link>
```

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**.



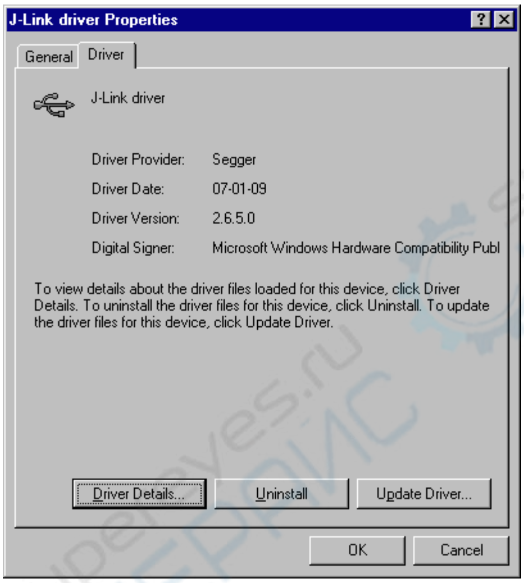
Device type: Universal Serial Bus controllers
Manufacturer: Segger
Location: J-Link

This device is working properly.

If you are having problems with this device, click Troubleshooter to start the troubleshooter.

Device usage: Use this device (enable)

If you experience problems, refer to the chapter *Support and FAQs* on page 333 for help. You can select the **Driver** tab for detailed information about driver provider, version, date and digital signer.



Driver Provider: Segger
Driver Date: 07-01-09
Driver Version: 2.6.5.0
Digital Signer: Microsoft Windows Hardware Compatibility Publ

To view details about the driver files loaded for this device, click Driver Details. To uninstall the driver files for this device, click Uninstall. To update the driver files for this device, click Update Driver.

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.

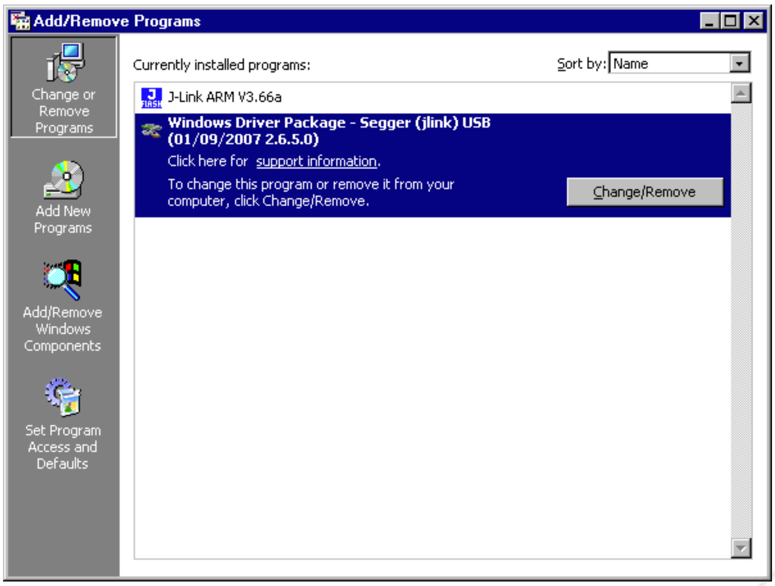
This might be the case when:

- 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.
2. Open the **Add/Remove Programs** dialog (`Start > Settings > Control Panel > Add/Remove Programs`) and select **Windows Driver Package - Segger**

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



Add/Remove Programs
Change or Remove Programs
Add New Programs
Add/Remove Windows Components
Set Program Access and Defaults
Change/Remove

Currently installed programs: Sort by: Name

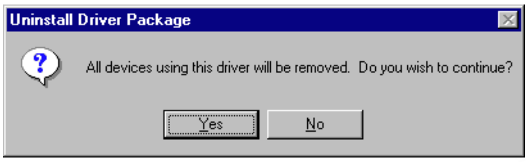
J-Link ARM V3.66a

Windows Driver Package - Segger (jlink) USB (01/09/2007 2.6.5.0)

Click here for support information.

To change this program or remove it from your computer, click Change/Remove.

3. Confirm the uninstallation process.



All devices using this driver will be removed. Do you wish to continue?

4.3 Setting up the IP interface

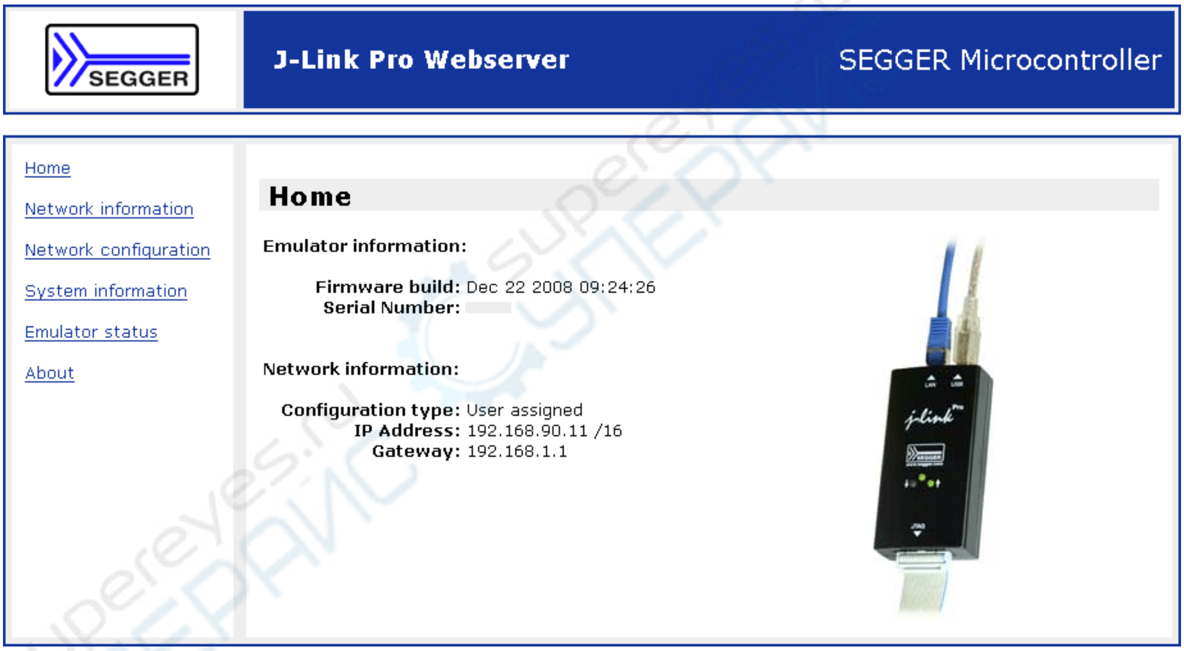
Some emulators of the J-Link family have (or future members will have) an additional Ethernet interface, to communicate with the host system. These emulators will also come with a built-in web server which allows configuration of the emulator via web interface. In addition to that, you can set a default 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 131.

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.



Home

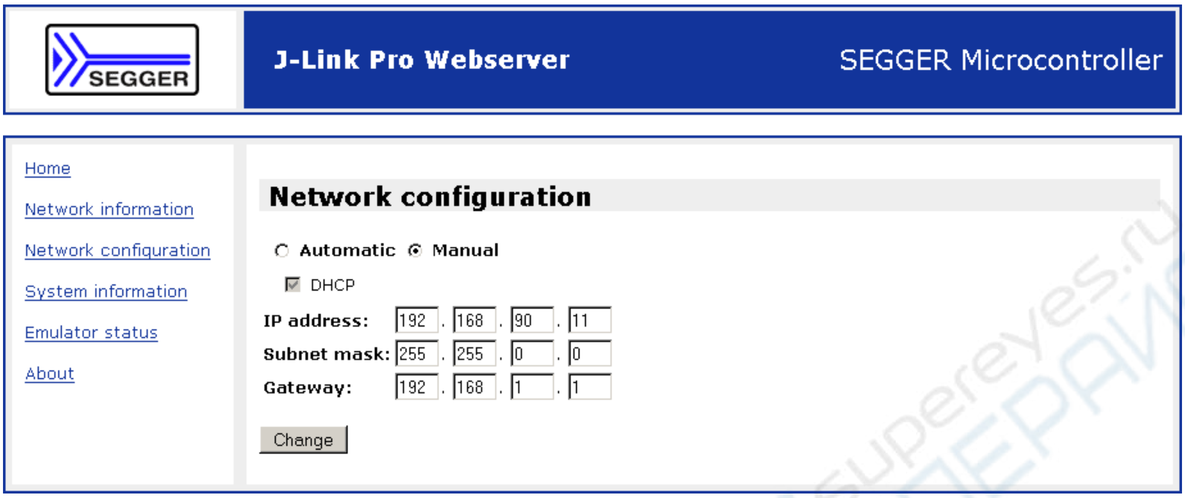
Emulator information:

Firmware build: Dec 22 2008 09:24:26
Serial Number: [redacted]

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.



Network configuration

○ Automatic ● Manual

☑ DHCP

IP address: 192 . 168 . 90 . 11

Subnet mask: 255 . 255 . 0 . 0

Gateway: 192 . 168 . 1 . 1

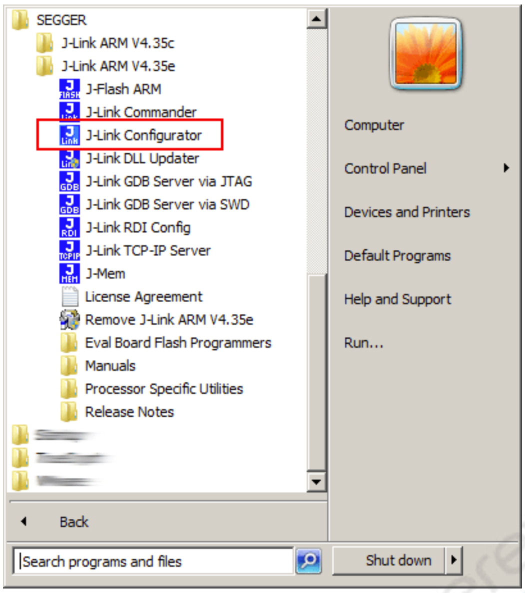
4.4 FAQs

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.

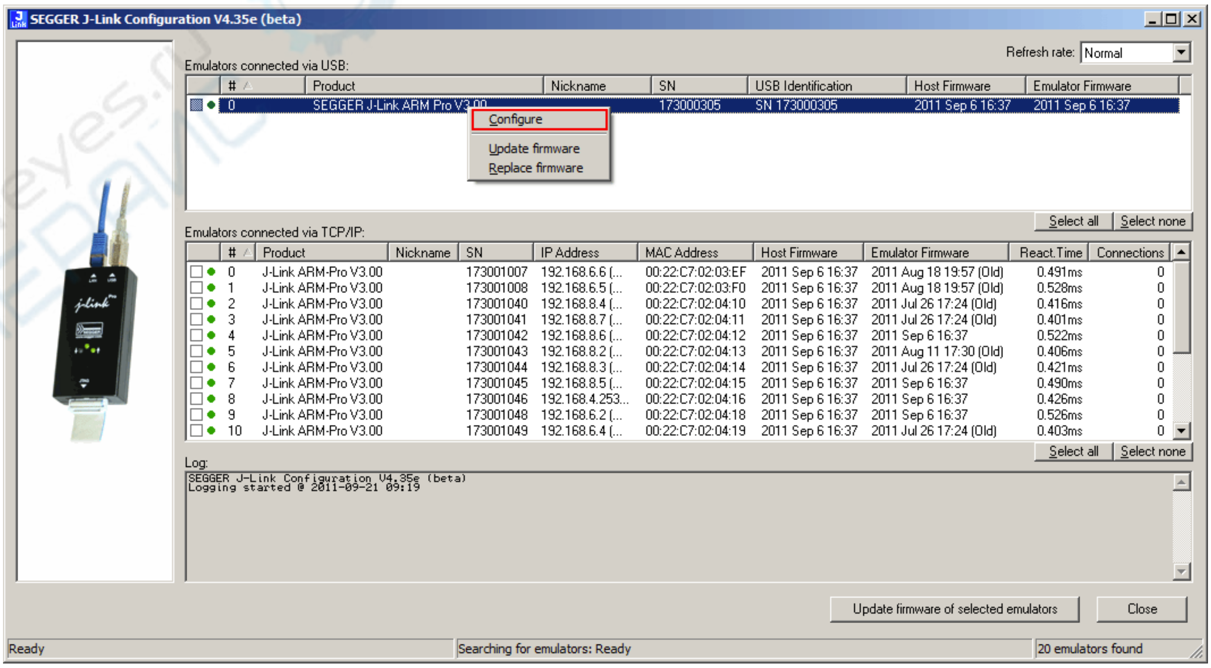
4.5 J-Link Configurator

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)
Configure
Update firmware
Replace firmware
Select all
Select none
Update firmware of selected emulators
Close

Refresh rate: Normal

Emulators connected via USB:

	#	Product	Nickname	SN	USB Identification	Host Firmware	Emulator Firmware
▨ ●	0	SEGGER J-Link ARM Pro V3.00		173000305	SN 173000305	2011 Sep 6 16:37	2011 Sep 6 16:37

Emulators connected via TCP/IP:

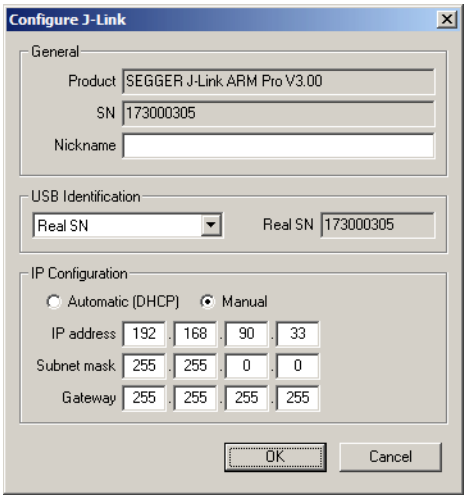
	#	Product	Nickname	SN	IP Address	MAC Address	Host Firmware	Emulator Firmware	React.Time	Connections
□ ●	0	J-Link ARM-Pro V3.00		173001007	192.168.6.6 (...	00:22:C7:02:03:EF	2011 Sep 6 16:37	2011 Aug 18 19:57 (Old)	0.491ms	0
□ ●	1	J-Link ARM-Pro V3.00		173001008	192.168.6.5 (...	00:22:C7:02:03:F0	2011 Sep 6 16:37	2011 Aug 18 19:57 (Old)	0.528ms	0
□ ●	2	J-Link ARM-Pro V3.00		173001040	192.168.8.4 (...	00:22:C7:02:04:10	2011 Sep 6 16:37	2011 Jul 26 17:24 (Old)	0.416ms	0
□ ●	3	J-Link ARM-Pro V3.00		173001041	192.168.8.7 (...	00:22:C7:02:04:11	2011 Sep 6 16:37	2011 Jul 26 17:24 (Old)	0.401ms	0
□ ●	4	J-Link ARM-Pro V3.00		173001042	192.168.8.6 (...	00:22:C7:02:04:12	2011 Sep 6 16:37	2011 Sep 6 16:37	0.522ms	0
□ ●	5	J-Link ARM-Pro V3.00		173001043	192.168.8.2 (...	00:22:C7:02:04:13	2011 Sep 6 16:37	2011 Aug 11 17:30 (Old)	0.406ms	0
□ ●	6	J-Link ARM-Pro V3.00		173001044	192.168.8.3 (...	00:22:C7:02:04:14	2011 Sep 6 16:37	2011 Jul 26 17:24 (Old)	0.421ms	0
□ ●	7	J-Link ARM-Pro V3.00		173001045	192.168.8.5 (...	00:22:C7:02:04:15	2011 Sep 6 16:37	2011 Sep 6 16:37	0.490ms	0
□ ●	8	J-Link ARM-Pro V3.00		173001046	192.168.4.253...	00:22:C7:02:04:16	2011 Sep 6 16:37	2011 Sep 6 16:37	0.426ms	0
□ ●	9	J-Link ARM-Pro V3.00		173001048	192.168.6.2 (...	00:22:C7:02:04:18	2011 Sep 6 16:37	2011 Sep 6 16:37	0.526ms	0
□ ●	10	J-Link ARM-Pro V3.00		173001049	192.168.6.4 (...	00:22:C7:02:04:19	2011 Sep 6 16:37	2011 Jul 26 17:24 (Old)	0.403ms	0

Log:

```
SEGGER J-Link Configuration V4.35e (beta)
Logging started @ 2011-09-21 09:19
```

Ready | Searching for emulators: Ready | 20 emulators found

In order to configure a old J-Link, which uses the old USB 0 - 3 USB identification method, 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.



Configure J-Link

General

Product: SEGGER J-Link ARM Pro V3.00

SN: 173000305

Nickname:

USB Identification

Real SN ▾ Real SN: 173000305

IP Configuration

○ Automatic (DHCP) ● Manual

IP address: 192 . 168 . 90 . 33

Subnet mask: 255 . 255 . 0 . 0

Gateway: 255 . 255 . 255 . 255

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.

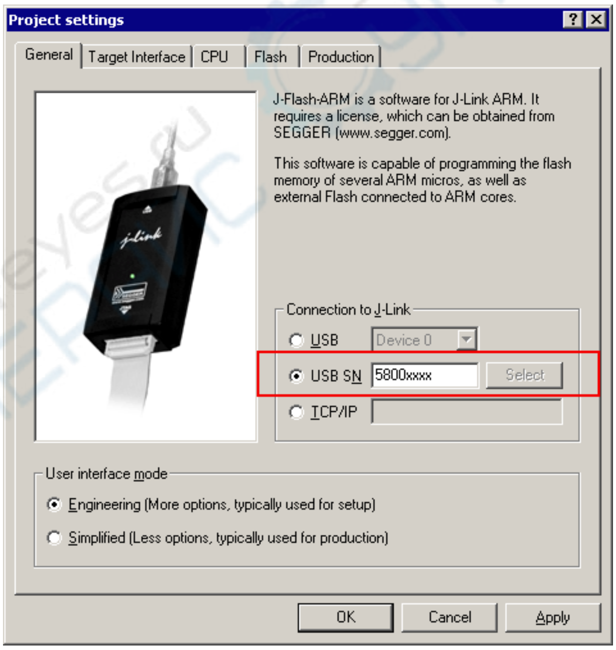
Identification	PID	Serial number
Serial number (default)	0x0101	Serial number is real serial number of the J-Link or user assigned.
USB address 0 (Deprecated)	0x0101	123456
USB address 1 (Deprecated)	0x0102	123456
USB address 2 (Deprecated)	0x0103	123456
USB address 3 (Deprecated)	0x0104	123456

Table 4.1: 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:



J-Flash-ARM is a software for J-Link ARM. It requires a license, which can be obtained from SEGGER (www.segger.com).

This software is capable of programming the flash memory of several ARM micros, as well as external Flash connected to ARM cores.

Connection to J-Link

○ USB Device 0

● USB SN 5800xxxx Select

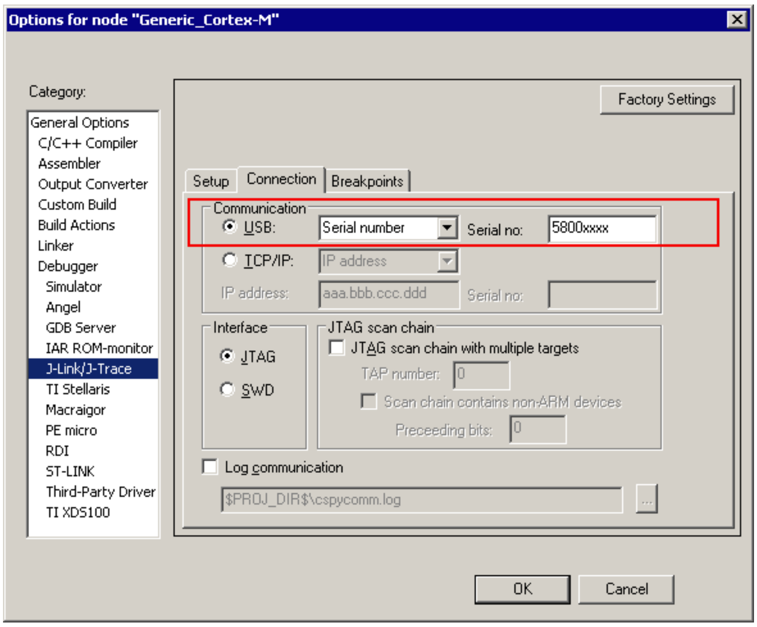
○ TCP/IP

User interface mode

● Engineering (More options, typically used for setup)

○ Simplified (Less options, typically used for production)

The following screenshot shows the connection dialog of IAR EWARM:



Options for node "Generic_Cortex-M"

Category:

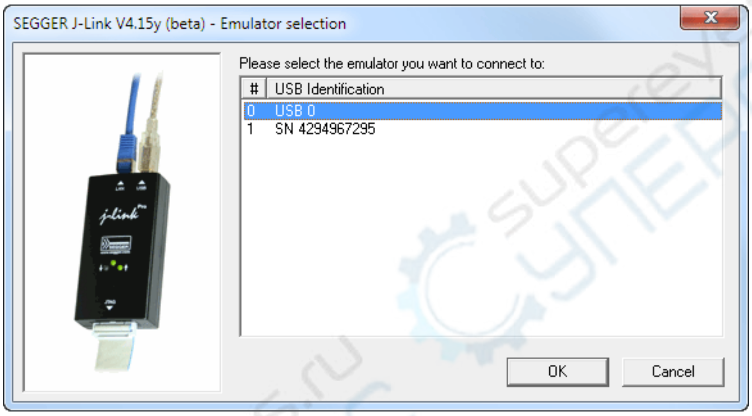
Communication
● USB: Serial number Serial no: 5800xxxx
○ TCP/IP: IP address
IP address: aaa.bbb.ccc.ddd Serial no:

Interface
● JTAG
○ SWD

JTAG scan chain
☐ JTAG scan chain with multiple targets
TAP number: 0
☐ Scan chain contains non-ARM devices
Preceeding bits: 0

☐ Log communication
$PROJ_DIR$\cspycomm.log

For debuggers / software which does not provide such a functionality, the J-Link DLL automatically detects that mutliple 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
0	USB 0
1	SN 4294967295

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

4.7 Using the J-Link DLL

4.7.1 What is the JLink 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 a CPU core.

4.7.2 Updating the DLL in third-party programs

The JLink DLL can be used by any debugger that is designed to work with it. Some debuggers are usually shipped with the J-Link 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.

4.7.2.1 Updating the J-Link DLL in the IAR Embedded Workbench for ARM (EWARM)

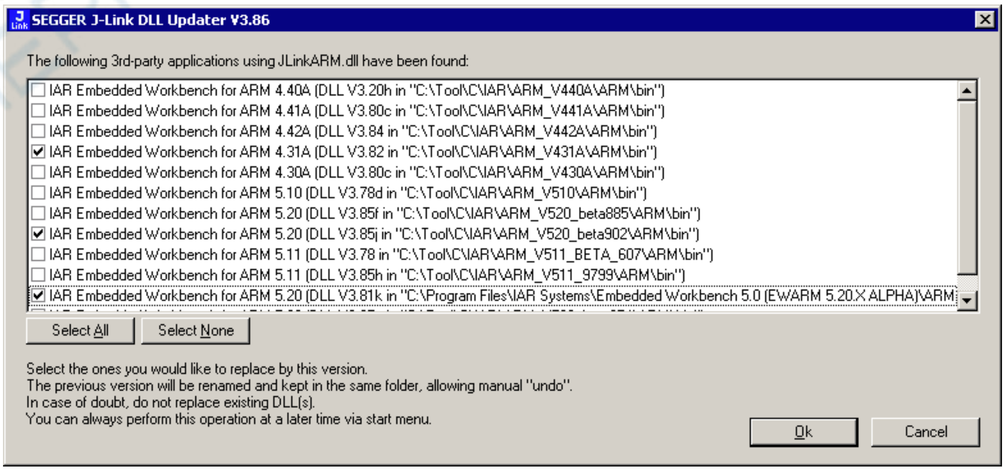
It's recommended to use the J-Link DLL updater to update the J-Link DLL in the IAR Embedded Workbench. The IAR Embedded Workbench IDE is a high-performance integrated development environment with an editor, compiler, linker, debugger. The compiler generates very efficient code and is widely used. It 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 136.

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.



SEGGER J-Link DLL Updater V3.86

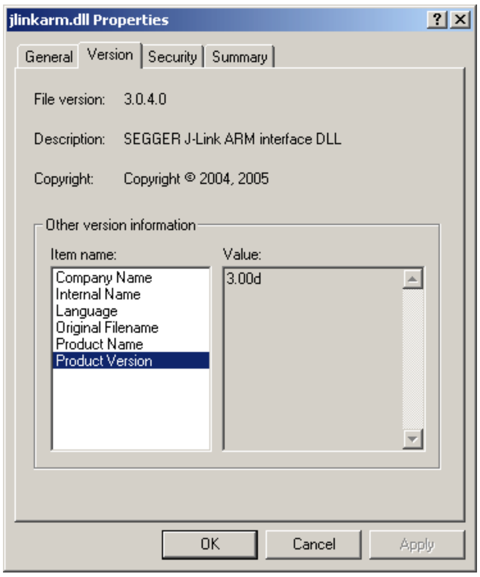
The following 3rd-party applications using JLinkARM.dll have been found:

☐ IAR Embedded Workbench for ARM 4.40A (DLL V3.20h in "C:\Tool\C\IAR\ARM_V440A\ARM\bin")
☐ IAR Embedded Workbench for ARM 4.41A (DLL V3.80c in "C:\Tool\C\IAR\ARM_V441A\ARM\bin")
☐ IAR Embedded Workbench for ARM 4.42A (DLL V3.84 in "C:\Tool\C\IAR\ARM_V442A\ARM\bin")
☑ IAR Embedded Workbench for ARM 4.31A (DLL V3.82 in "C:\Tool\C\IAR\ARM_V431A\ARM\bin")
☐ IAR Embedded Workbench for ARM 4.30A (DLL V3.80c in "C:\Tool\C\IAR\ARM_V430A\ARM\bin")
☐ IAR Embedded Workbench for ARM 5.10 (DLL V3.78d in "C:\Tool\C\IAR\ARM_V510\ARM\bin")
☐ IAR Embedded Workbench for ARM 5.20 (DLL V3.85f in "C:\Tool\C\IAR\ARM_V520_beta885\ARM\bin")
☑ IAR Embedded Workbench for ARM 5.20 (DLL V3.85j in "C:\Tool\C\IAR\ARM_V520_beta902\ARM\bin")
☐ IAR Embedded Workbench for ARM 5.11 (DLL V3.78 in "C:\Tool\C\IAR\ARM_V511_BETA_607\ARM\bin")
☐ IAR Embedded Workbench for ARM 5.11 (DLL V3.85h in "C:\Tool\C\IAR\ARM_V511_9799\ARM\bin")
☑ IAR Embedded Workbench for ARM 5.20 (DLL V3.81k in "C:\Program Files\IAR Systems\Embedded Workbench 5.0 (EWARM 5.20.X ALPHA)\ARM

Select the ones you would like to replace by this version.
The previous version will be renamed and kept in the same folder, allowing manual "undo".
In case of doubt, do not replace existing DLL(s).
You can always perform this operation at a later time via start menu.

4.7.3 Determining the version of JLink 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.



File version: 3.0.4.0

Description: SEGGER J-Link ARM interface DLL

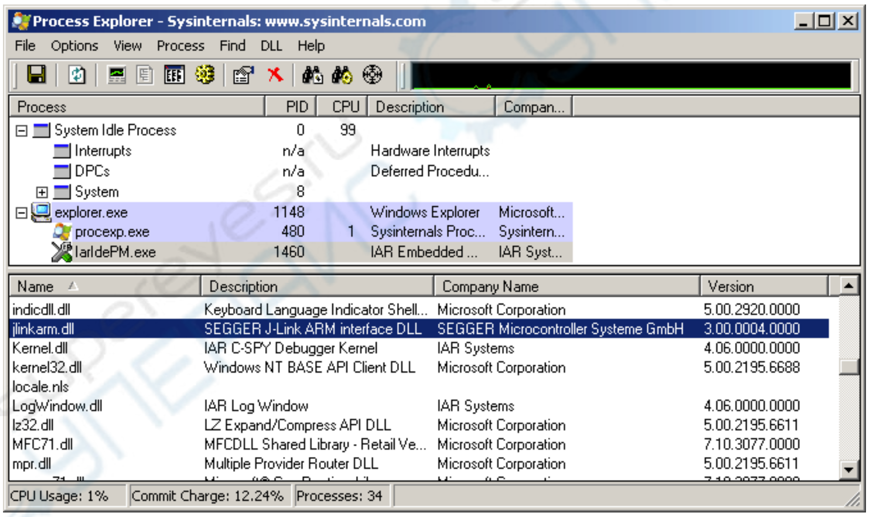
Copyright: Copyright © 2004, 2005

Other version information

Item name:	Value:
Company Name Internal Name Language Original Filename Product Name Product Version	3.00d

4.7.4 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	PID	CPU	Description	Compan...
System Idle Process	0	99		
Interrupts	n/a		Hardware Interrupts	
DPCs	n/a		Deferred Procedu...	
System	8			
explorer.exe	1148		Windows Explorer	Microsoft...
procexp.exe	480	1	Sysinternals Proc...	Sysintern...
IarIdePM.exe	1460		IAR 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 DLL	SEGGER Microcontroller 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 DLL	Microsoft Corporation	5.00.2195.6688
locale.nls			
LogWindow.dll	IAR Log Window	IAR Systems	4.06.0000.0000
lz32.dll	LZ Expand/Compress API DLL	Microsoft Corporation	5.00.2195.6611
MFC71.dll	MFCDLL Shared Library - Retail Ve...	Microsoft Corporation	7.10.3077.0000
mpr.dll	Multiple Provider Router DLL	Microsoft Corporation	5.00.2195.6611

CPU Usage: 1% Commit Charge: 12.24% Processes: 34

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

4.8 Getting started with J-Link and ARM DS-5

J-Link supports ARM DS-5 Development via the RDDI protocol.

For commerically using J-Link via RDDI in ARM DS-5, an RDI/RDDI license is required. J-Link models which come with an RDI license, can also be used via RDDI. RDDI can be evaluated free of charge.

In order to use J-Link in ARM DS-5 Development Studio, the RDDI DLL in DS-5 needs to be replaced by the SEGGER version of this DLL. The SEGGER version of the RDDI still allows using ARM's DSTREAM in DS-5. After installing the J-Link software and documentation package, the J-Link DLL Updater is started which allows easily updating the RDDI DLL in DS-5. An backup of the original DLL is made automatically.

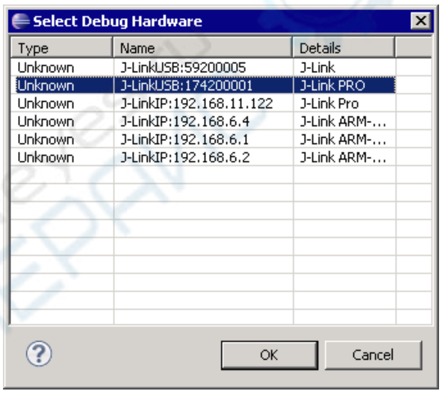
4.8.1 Replacing the RDDI DLL manually

If J-Link DLL Updater is unable to find a DS-5 installation and does not list it for updating, the RDDI DLL can always be replaced manually. For more information about how to manually update the RDDI DLL, please refer to `$JLINK_INST_DIR$\RDDI\ManualInstallation.txt`.

4.8.2 Using J-Link in DS-5 Development Studio

Please follow the following steps, in order to use J-Link in DS-5, after replacing the RDDI DLL accordingly:

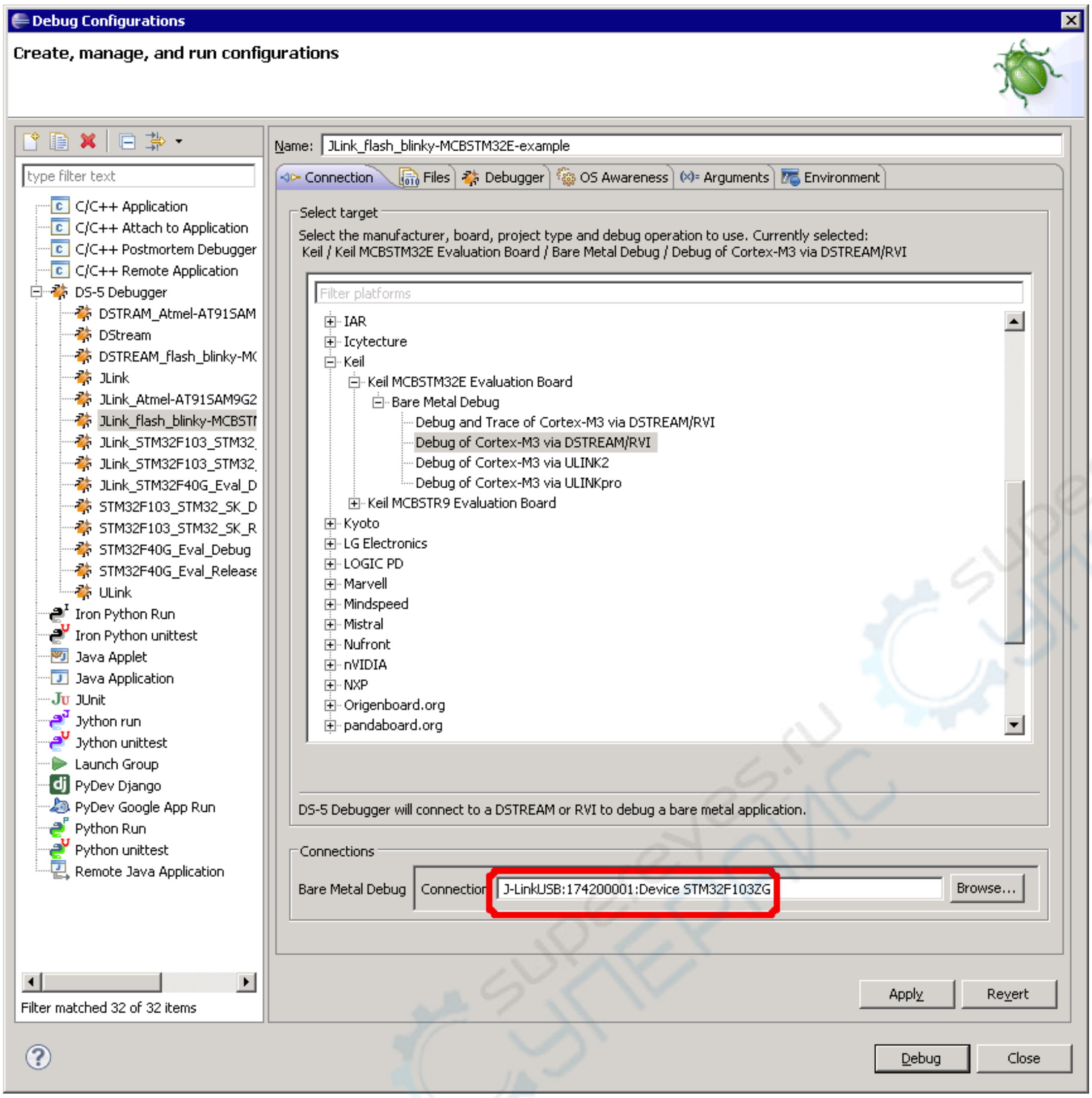
- Connect J-Link and target.
- Open ARM DS-5.
- Open DS-5 Project for target.
- Open **Debug Configurations...**
- Select DS-5 Debugger on the left side.
- Press **New** button.
- In the Connection tab, select The target from the device database, **Bare Metal Debug, Debug via DSTREAM/RVI.**
- Click on the **Browse...** button right to the Text box at the bottom **Connection.**
- In the Dialog select the J-Link which is connected to the target (e.g. **JLinkUSB:174200001**).



Type	Name	Details
Unknown	J-LinkUSB:59200005	J-Link
Unknown	J-LinkUSB:174200001	J-Link PRO
Unknown	J-LinkIP:192.168.11.122	J-Link Pro
Unknown	J-LinkIP:192.168.6.4	J-Link ARM-...
Unknown	J-LinkIP:192.168.6.1	J-Link ARM-...
Unknown	J-LinkIP:192.168.6.2	J-Link ARM-...

- Click **OK**.
- Add the device name to the connection string (e.g. JLinkUSB:174200001**:Device**

STM32F103ZG)



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- DS-5 Debugger
  - DSTRAM_Atmel-AT91SAM
  - DStream
  - DSTREAM_flash_blinky-MC
  - JLink
  - JLink_Atmel-AT91SAM9G2
  - JLink_flash_blinky-MCBSTI
  - JLink_STM32F103_STM32_
  - JLink_STM32F103_STM32_
  - JLink_STM32F40G_Eval_D
  - STM32F103_STM32_SK_D
  - STM32F103_STM32_SK_R
  - STM32F40G_Eval_Debug
  - STM32F40G_Eval_Release
  - ULink
- Iron Python Run
- Iron Python unittest
- Java Applet
- Java Application
- JUnit
- Jython run
- Jython unittest
- Launch Group
- PyDev Django
- PyDev Google App Run
- Python Run
- Python unittest
- Remote Java Application

Filter matched 32 of 32 items

Name: JLink_flash_blinky-MCBSTM32E-example

Connection | Files | Debugger | OS Awareness | Arguments | Environment

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Keil / Keil MCBSTM32E Evaluation Board / Bare Metal Debug / Debug of Cortex-M3 via DSTREAM/RVI

Filter platforms

- IAR
- Icytecture
- Keil
  - Keil MCBSTM32E Evaluation Board
    - Bare Metal Debug
      - Debug and Trace of Cortex-M3 via DSTREAM/RVI
      - Debug of Cortex-M3 via DSTREAM/RVI
      - Debug of Cortex-M3 via ULINK2
      - Debug of Cortex-M3 via ULINKpro
  - Keil MCBSTR9 Evaluation Board
- Kyoto
- LG Electronics
- LOGIC PD
- Marvell
- Mindspeed
- Mistral
- Nufront
- nVIDIA
- NXP
- Origenboard.org
- pandaboard.org

DS-5 Debugger will connect to a DSTREAM or RVI to debug a bare metal application.

Connections

Bare Metal Debug Connection J-LinkUSB:174200001:Device STM32F103ZG Browse...

- Click **Apply**.
- In the **Files** tab, select the application to download.
- In the **Debugger** tab, select 'Debug from symbol' and enter main or select **Debug from entry point**.
- Click **Apply**.
- Start a new debug session with the newly created debug configuration.
- Now the debug session should start and downloaded the application to the target.

Chapter 5

Working with J-Link and J-Trace

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

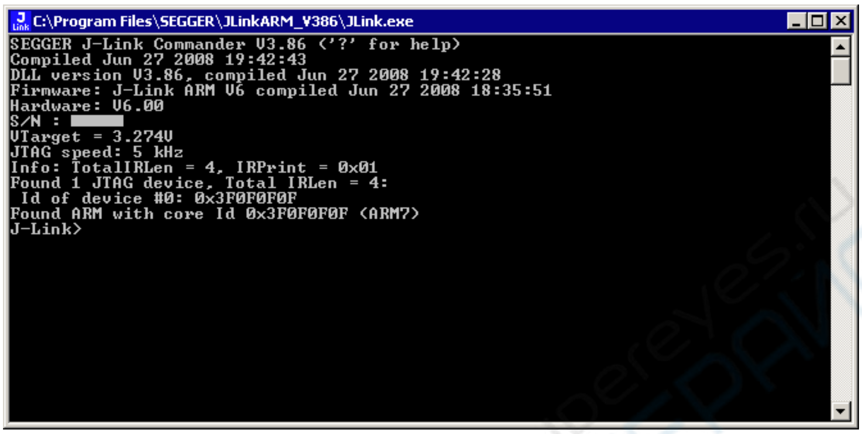
5.1 Connecting the target system

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.



```
SEGGER J-Link Commander V3.86 ('?' for help)
Compiled Jun 27 2008 19:42:43
DLL version V3.86, compiled Jun 27 2008 19:42:28
Firmware: J-Link ARM V6 compiled Jun 27 2008 18:35:51
Hardware: V6.00
S/N : [redacted]
VTarget = 3.274V
JTAG speed: 5 kHz
Info: TotalIRLen = 4, IRPrint = 0x01
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x3F0F0F0F
Found ARM with core Id 0x3F0F0F0F (ARM7)
J-Link>
```

5.1.3 Problems

If you experience problems with any of the steps described above, read the chapter *Support and FAQs* on page 333 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 333 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 executing 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 executing 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).

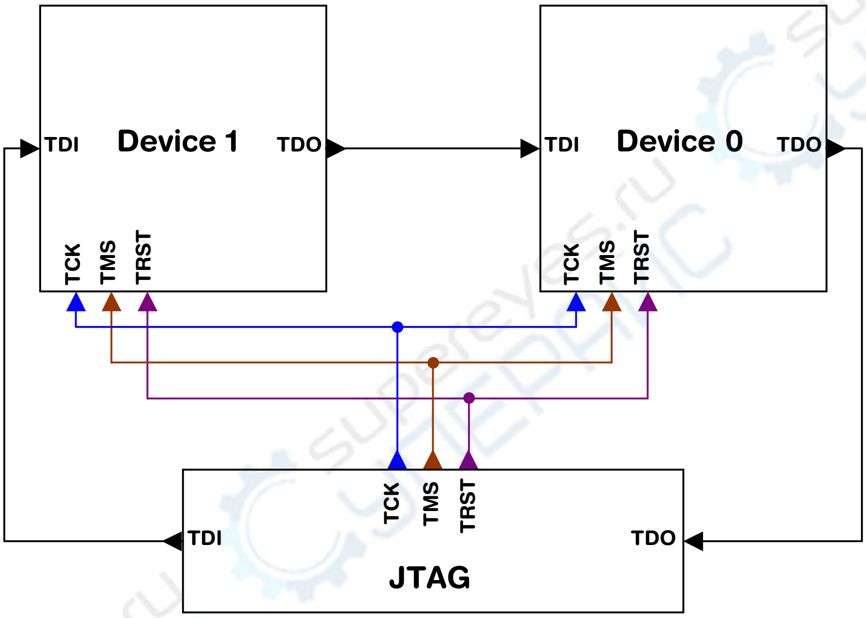
Table 5.4: J-Link bi-color output indicator

5.3 JTAG interface

By default, only one 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 CPU 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 CPU cores; the other devices can be of any other type but need to comply with the JTAG standard.

5.3.1.1 Configuration

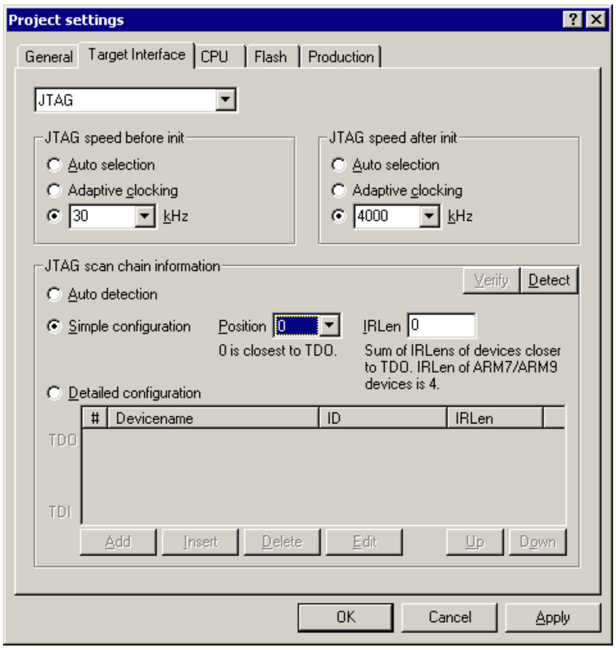
The configuration of the scan chain depends on the application used. Read *JTAG interface* on page 144 for further instructions and configuration examples.

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
- ○ Auto selection
- ○ Adaptive clocking
- ● 30 kHz

JTAG speed after init
- ○ Auto selection
- ○ Adaptive clocking
- ● 4000 kHz

JTAG scan chain information

Verify | Detect

- ○ Auto detection
- ● Simple configuration — Position 0 — IRLen 0

0 is closest to TDO.

Sum of IRLens of devices closer to TDO. IRLen of ARM7/ARM9 devices is 4.

- ○ Detailed configuration

#	Devicename	ID	IRLen

TDO

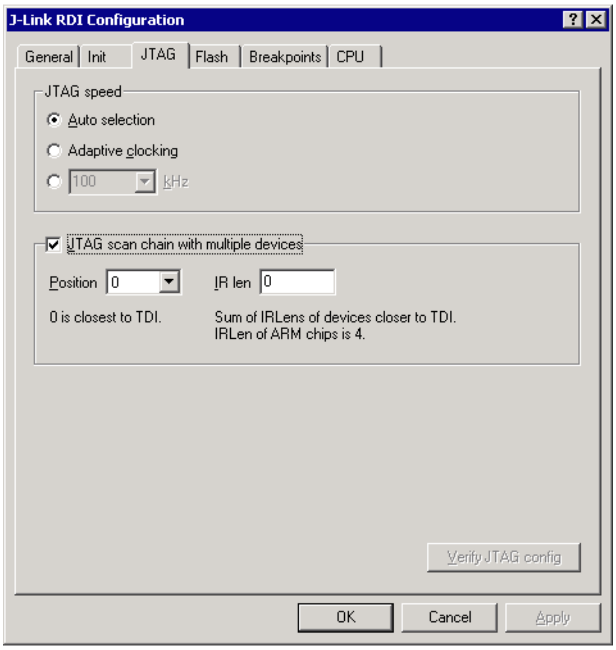
TDI

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.



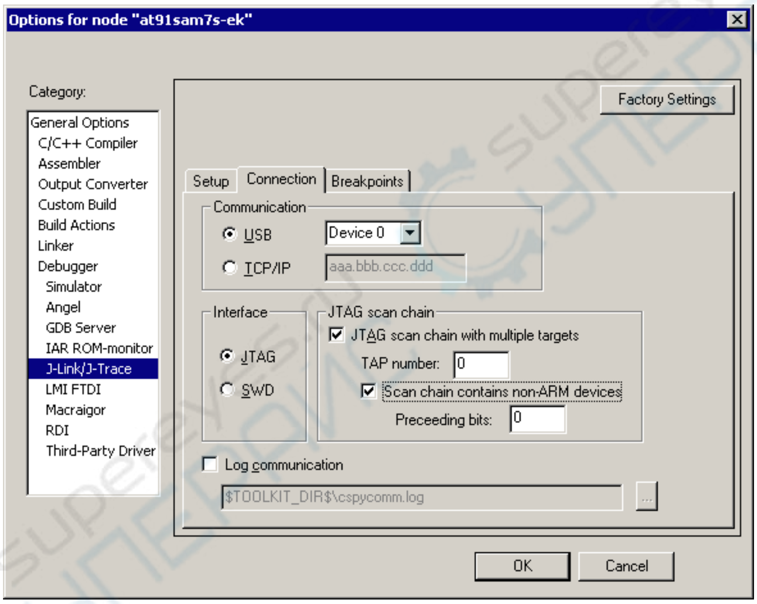
J-Link RDI Configuration

JTAG speed
● Auto selection
○ Adaptive clocking
○ 100 kHz

☑ JTAG scan chain with multiple devices
Position: 0
IR len: 0
0 is closest to TDI.
Sum of IRLens of devices closer to TDI. IRLen of ARM chips is 4.

IAR J-Link configuration dialog box

This dialog box can be found under `Project|Options`.



Options for node "at91sam7s-ek"

Category:

Communication
● USB: Device 0
○ TCP/IP: aaa.bbb.ccc.ddd

Interface
● JTAG
○ SWD

JTAG scan chain
☑ JTAG scan chain with multiple targets
TAP number: 0
☑ Scan chain contains non-ARM devices
Preceeding bits: 0

□ Log communication
$TOOLKIT_DIR$\cspycomm.log

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:

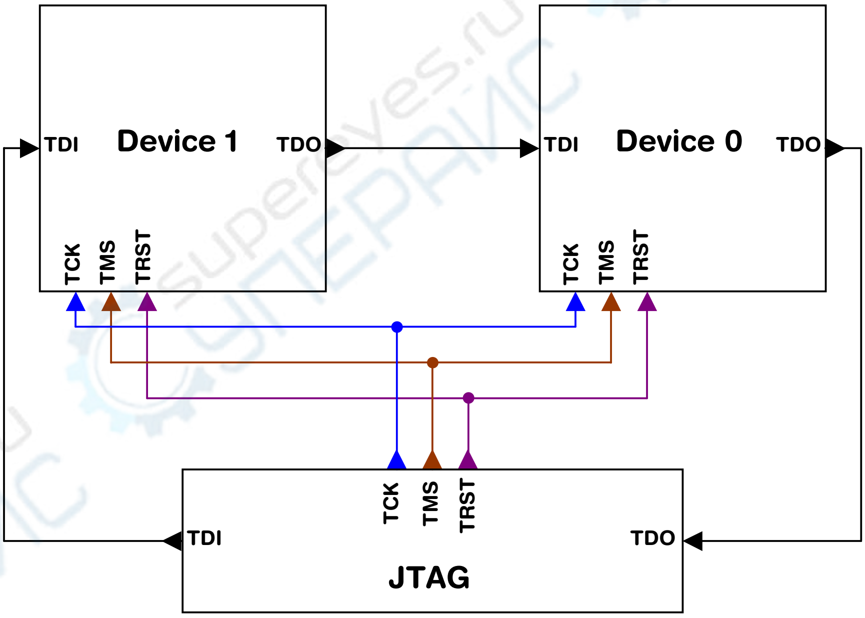
- The position of the target device in the scan chain
- The total number of bits in the instruction registers of the devices before the target device (IR len).

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

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 CPU 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 non-adaptive 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 24.

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. To get the supported SWO speeds for your emulator, use J-Link Commander:

```
J-Link> si 1   //Select target interface SWD
J-Link> SWOSpeed
```

Currently, following speeds are supported:

Emulator	Speed formula	Resulting max. speed
J-Link V9	60MHz/n, n >= 8	7.5 MHz
J-Link Pro/ULTRA V4	3.2GHz/n, n >= 64	50 MHz

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 149 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 V9: Supported SWO input speeds are: 60MHz / n, n>= 8:
7.5MHz, 6.66MHz, 6MHz, ...

Permitted combinations are:

SWO output	SWO input	Deviation percent
6MHz, n = 12	6MHz, n = 10	0
4MHz, n = 18	4MHz, n = 15	0
...	...	<= 3
2MHz, n = 36	2MHz, n = 30	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

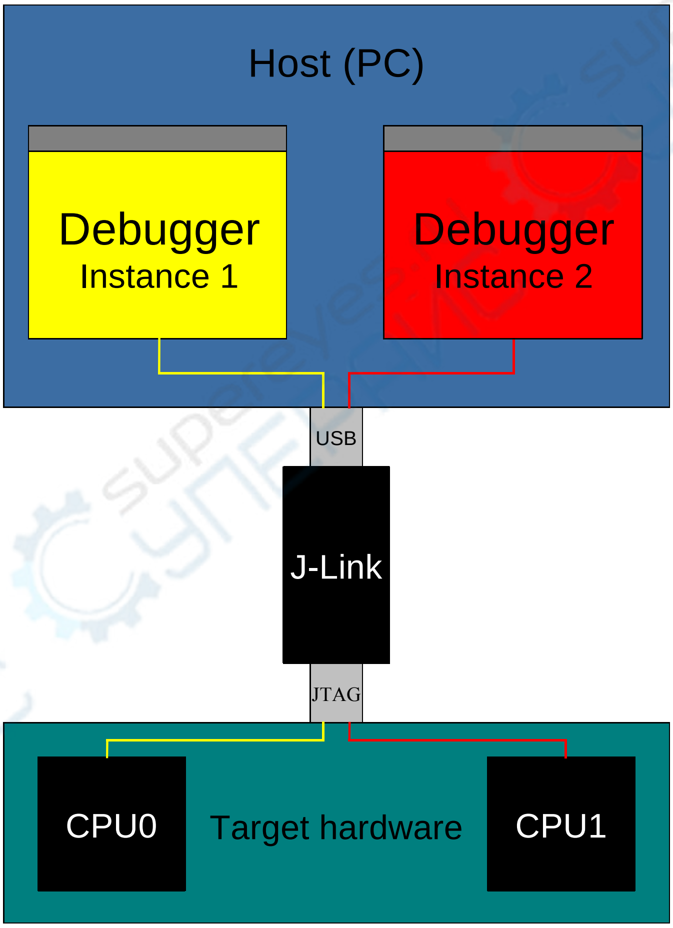
5.5 Multi-core debugging

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.

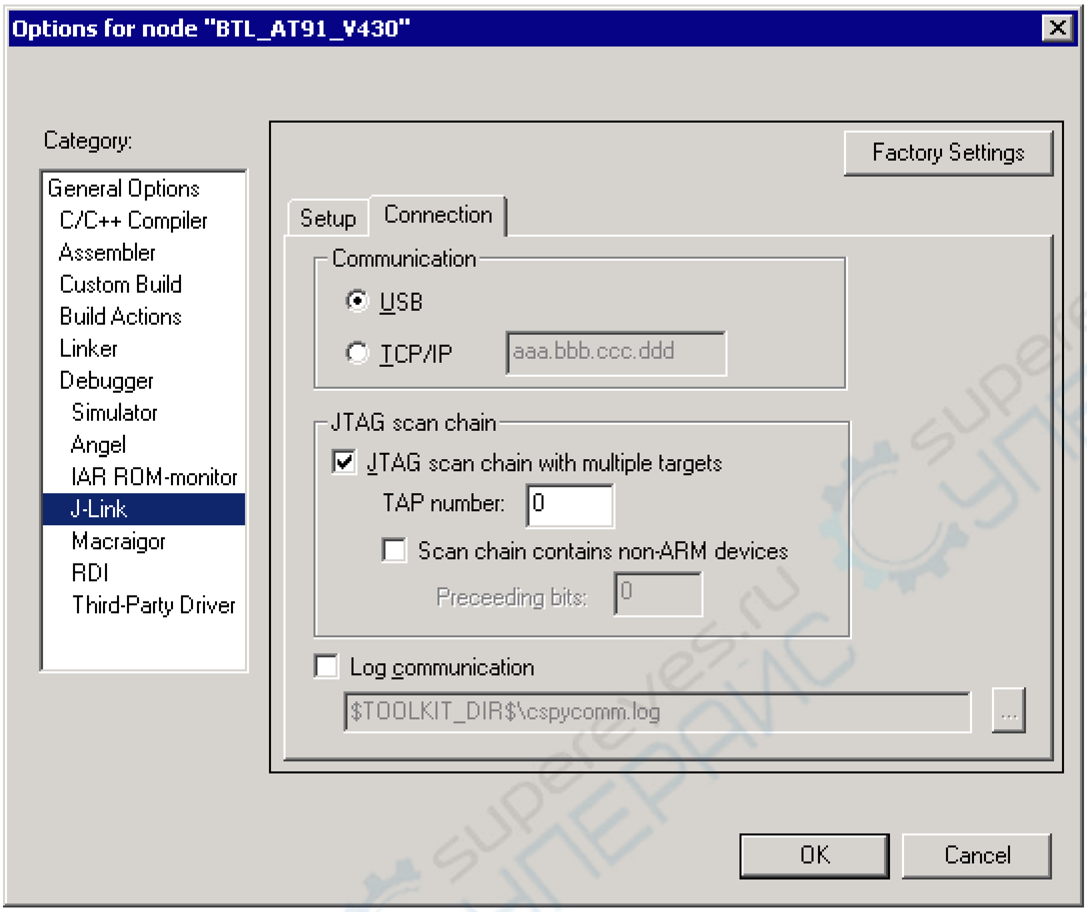
The following figure shows a host, debugging two CPU cores with two instances of the same debugger.



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 CPU core on your target.



Options for node "BTL_AT91_V430"

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

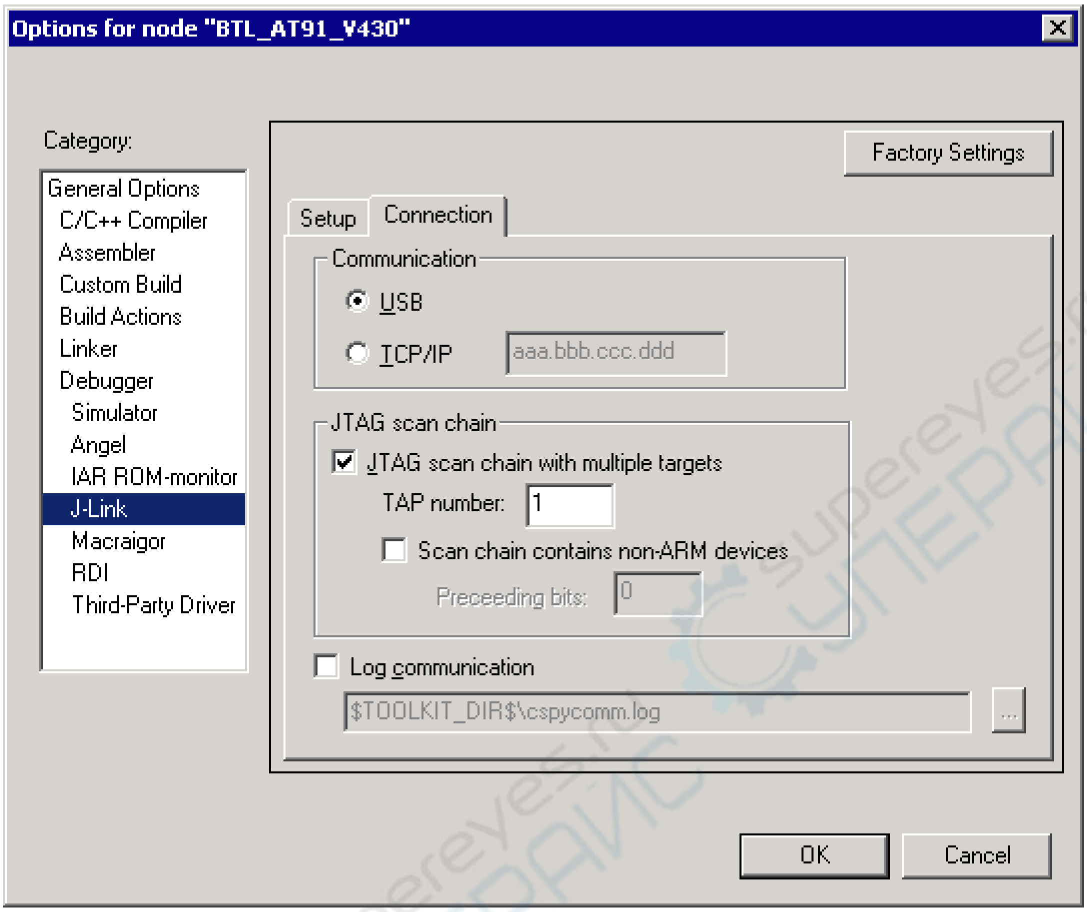
Communication
- ● USB
- ○ TCP/IP aaa.bbb.ccc.ddd

JTAG scan chain
- ☑ JTAG scan chain with multiple targets
- TAP number: 0
- ☐ Scan chain contains non-ARM devices
- Preceeding bits: 0

☐ Log communication
$TOOLKIT_DIR$\cspycomm.log

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.



Communication
● USB
○ TCP/IP aaa.bbb.ccc.ddd

JTAG scan chain
☑ JTAG scan chain with multiple targets
TAP number: 1
□ Scan chain contains non-ARM devices
Preceeding bits: 0

□ Log communication
$TOOLKIT_DIR$\cspycomm.log

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
ARM7TDMI-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

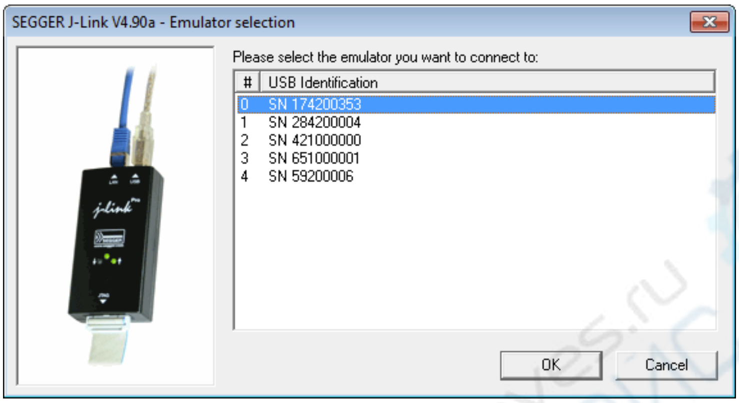
All cores share the same RESET line. You should be aware that resetting one core through the RESET line means resetting all cores which have their RESET pins connected to the RESET line on the target.

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. In order to connect to the correct J-Link, the user has to make sure that the correct J-Link is selected (by SN or IP). In cases where no specific J-Link is selected, following pop-up will shop and allow the user to select the proper J-Link:

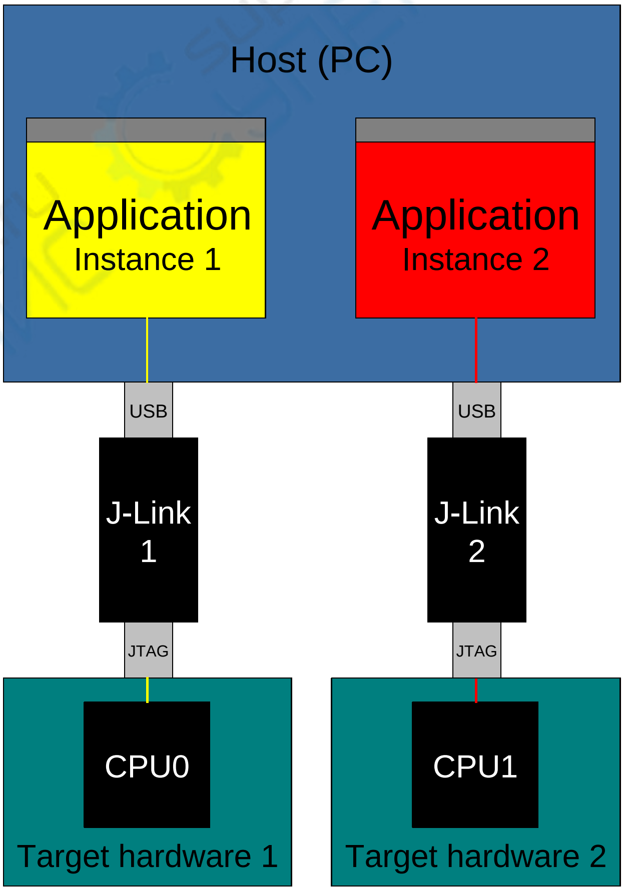


SEGGER J-Link V4.90a - Emulator selection

Please select the emulator you want to connect to:

#	USB Identification
0	SN 174200353
1	SN 284200004
2	SN 421000000
3	SN 651000001
4	SN 59200006

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



Older J-Links may report USB0-3 instead of unique serial number when enumerating via USB. For these J-Links, we recommend to re-configure them to use the new enumeration method (report real serial number) since the old USB0-3 behavior is absolete.

Re-configuration can be done by using the J-Link Configurator, which is part of the J-Link software and documentation package. For further information about the J-Link configurator and how to use it, please refer to *J-Link Configurator* on page 131.

Re-configuration to the old USB 0-3 enumeration method

In some special cases, it may be necessary to switch back to the obsolete USB 0-3 enumeration method. For example, old IAR EWARM versions supports connecting to a J-Link via the USB0-3 method only. As soon as more then one J-Link is connected to the pc, there is no oppertunity to pre-select the J-Link which should be used for a debug session.

Below, a small instruction of how to re-configure J-Link to enumerate with the old obsolete enumeration method in order to prevent compatibility problems, is given.

Config area byte	Meaning
0	USB-Address. Can be set to 0-3, 0xFF is default which means USB-Address 0.
1	Enumeration method 0x00 / 0xFF: USB-Address is used for enumeration 0x01: Real-SN is used for enumeration

Table 5.10: Config area layout: USB-Enumeration settings

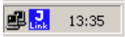
Example for setting enumeration method to USB 2:

1. Start J-Link Commander (JLink.exe) which is part of the J-Link software
2. Enter `wconf 0 02` // Set USB-Address 2
3. Enter `wconf 1 00` // Set enumeration method to USB-Address
4. Power-cycle J-Link in order to apply new configuration.

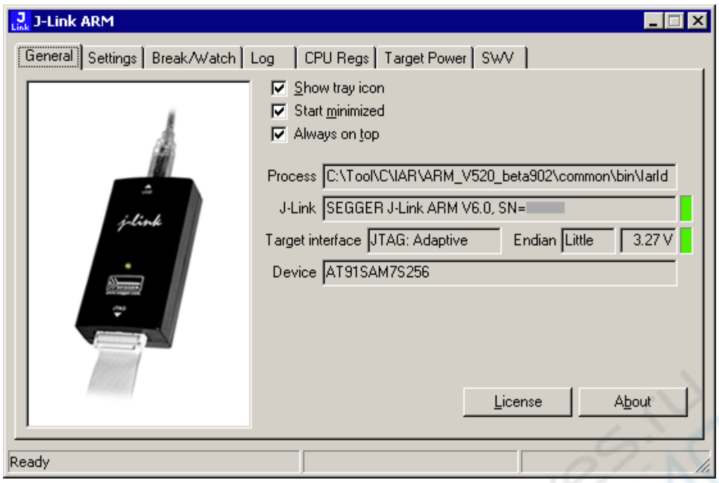
Re-configuration to REAL-SN enumeration can be done by using the J-Link Configurator, which is part of the J-Link software and documentation package. For further information about the J-Link configurator and how to use it, please refer to *J-Link Configurator* on page 131.

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 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.



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



☑ Show tray icon
☑ Start minimized
☑ Always on top

Process: C:\Tool\C\IAR\ARM_V520_beta902\common\bin\arld

J-Link: SEGGER J-Link ARM V6.0, SN=[redacted]

Target interface: JTAG: Adaptive   Endian: Little   3.27 V

Device: AT91SAM7S256

Ready

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:

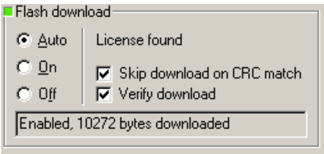
- **Process**: Shows the path of the file which loaded the DLL.
- **J-Link**: Shows OEM of the connected J-Link, the hardware version and the Serial number. If no J-Link is connected it shows "not connected" and the color indicator is red.
- **Target interface**: Shows the selected target interface (JTAG/SWD) 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 FlashDL` feature and related settings can be configured. When a license for `J-Link FlashDL` is found, the color indicator is green and "License found" appears right to the `J-Link FlashDL` usage settings.



🟩 Flash download

● Auto
○ On
○ Off

License found

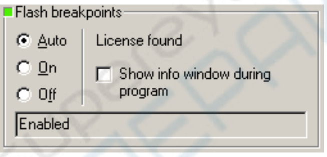
☑ Skip download on CRC match
☑ Verify download

Enabled, 10272 bytes downloaded

- **Auto**: This is the default setting of `J-Link FlashDL` usage. If a license is found `J-Link FlashDL` is enabled. Otherwise `J-Link FlashDL` will be disabled internally.
- **On**: Enables the `J-Link FlashDL` feature. If no license has been found an error message appears.
- **Off**: Disables the `J-Link 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 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 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.



🟩 Flash breakpoints

● Auto
○ On
○ Off

License found

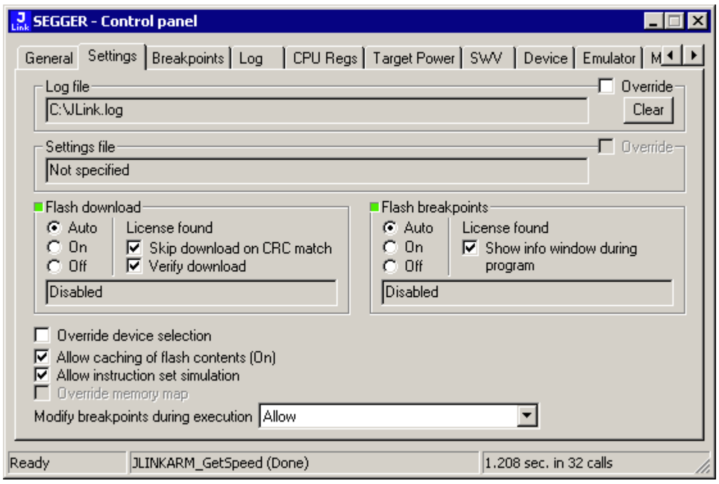
☐ Show info window during program

Enabled

- **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.



Log file: C:\JLink.log □ Override

Settings file: Not specified □ Override

Flash download: ● Auto ○ On ○ Off
License found
☑ Skip download on CRC match
☑ Verify download
Disabled

Flash breakpoints: ● Auto ○ On ○ Off
License found
☑ Show info window during program
Disabled

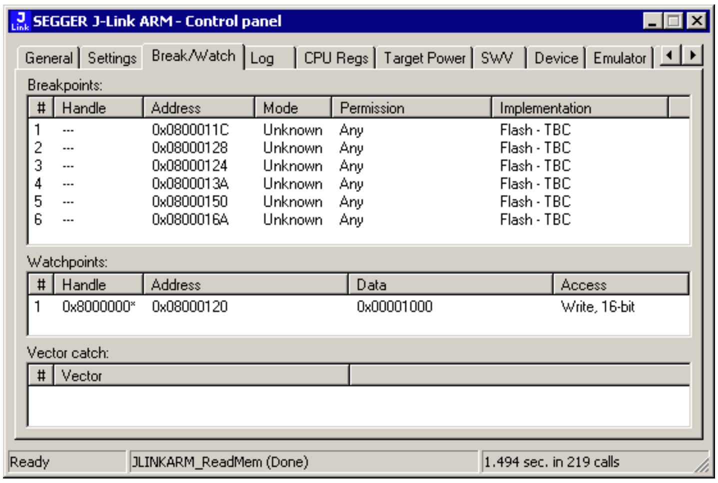
□ Override device selection
☑ Allow caching of flash contents (On)
☑ Allow instruction set simulation
□ Override memory map
Modify breakpoints during execution: Allow

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, 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.



Breakpoints:

#	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

Watchpoints:

#	Handle	Address	Data	Access
1	0x8000000*	0x08000120	0x00001000	Write, 16-bit

Vector catch:

#	Vector	

Ready | 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

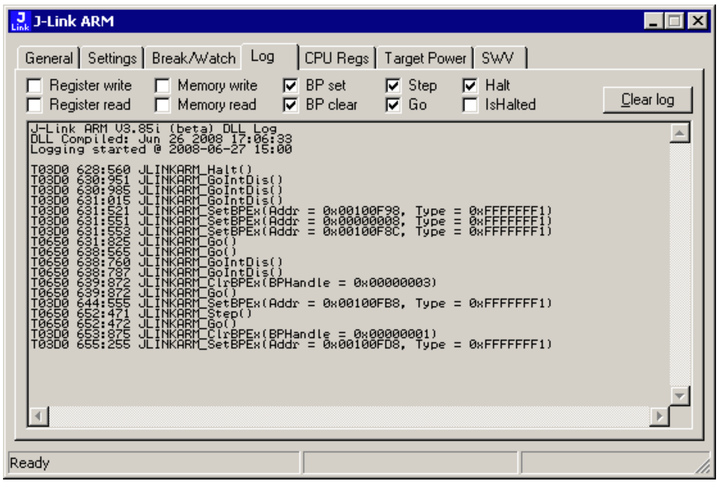
In this section, all data breakpoints which are listed in the DLL internal breakpoint list are shown.

- **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.



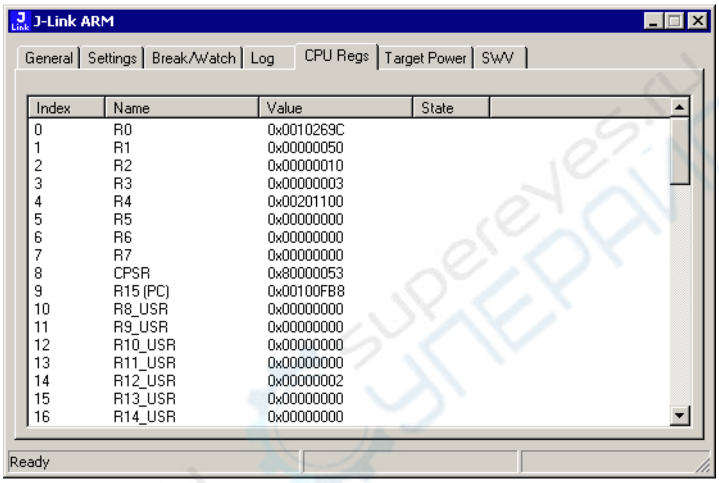
□ Register write □ Memory write ☑ BP set ☑ Step ☑ Halt
□ Register read □ Memory read ☑ BP clear ☑ Go □ IsHalted

```
J-Link ARM V3.85i (beta) DLL Log
DLL Compiled: Jun 26 2008 17:06:33
Logging started @ 2008-06-27 15:00

T03D0 628:560 JLINKARM_Halt()
T03D0 630:951 JLINKARM_GoIntDis()
T03D0 630:985 JLINKARM_GoIntDis()
T03D0 631:015 JLINKARM_GoIntDis()
T03D0 631:521 JLINKARM_SetBPEx(Addr = 0x00100F98, Type = 0xFFFFFFF1)
T03D0 631:551 JLINKARM_SetBPEx(Addr = 0x00000008, Type = 0xFFFFFFF1)
T03D0 631:553 JLINKARM_SetBPEx(Addr = 0x00100F8C, Type = 0xFFFFFFF1)
T0650 631:825 JLINKARM_Go()
T0650 638:565 JLINKARM_Go()
T0650 638:760 JLINKARM_GoIntDis()
T0650 638:787 JLINKARM_GoIntDis()
T0650 639:872 JLINKARM_ClrBPEx(BPHandle = 0x00000003)
T0650 639:872 JLINKARM_Go()
T03D0 644:555 JLINKARM_SetBPEx(Addr = 0x00100FB8, Type = 0xFFFFFFF1)
T0650 652:471 JLINKARM_Step()
T0650 652:472 JLINKARM_Go()
T03D0 653:875 JLINKARM_ClrBPEx(BPHandle = 0x00000001)
T03D0 655:255 JLINKARM_SetBPEx(Addr = 0x00100FD8, Type = 0xFFFFFFF1)
```

Ready

5.7.1.5 CPU Regs

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

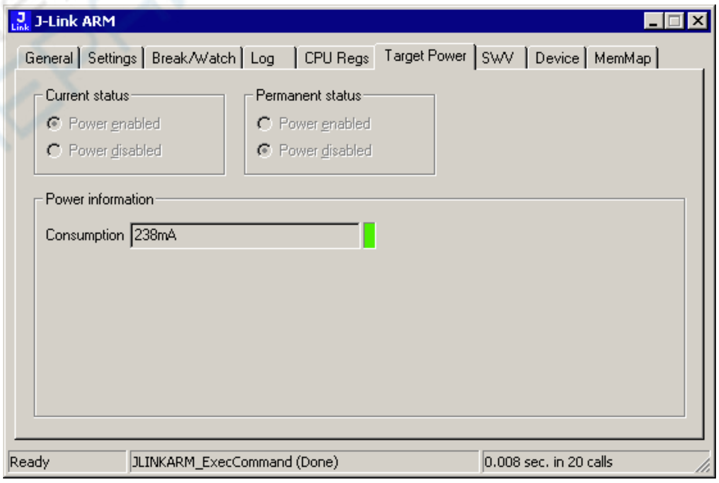


Index	Name	Value	State
0	R0	0x0010269C	
1	R1	0x00000050	
2	R2	0x00000010	
3	R3	0x00000003	
4	R4	0x00201100	
5	R5	0x00000000	
6	R6	0x00000000	
7	R7	0x00000000	
8	CPSR	0x80000053	
9	R15 (PC)	0x00100FB8	
10	R8_USR	0x00000000	
11	R9_USR	0x00000000	
12	R10_USR	0x00000000	
13	R11_USR	0x00000000	
14	R12_USR	0x00000002	
15	R13_USR	0x00000000	
16	R14_USR	0x00000000	

Ready

5.7.1.6 Target Power

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



Current status
● Power enabled
○ Power disabled

Permanent status
○ Power enabled
● Power disabled

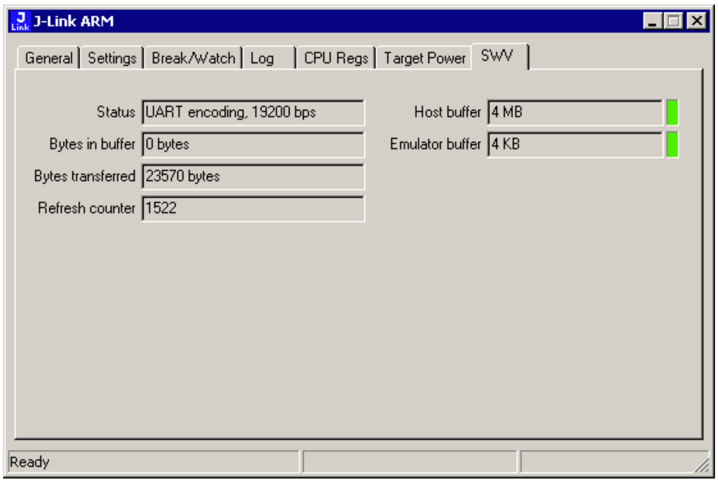
Power information

Consumption 238mA 🟩

Ready | JLINKARM_ExecCommand (Done) | 0.008 sec. in 20 calls

5.7.1.7 SWV

In this section SWV information are shown.



Status: UART encoding, 19200 bps

Bytes in buffer: 0 bytes

Bytes transferred: 23570 bytes

Refresh counter: 1522

Host buffer: 4 MB 🟩

Emulator buffer: 4 KB 🟩

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 CPU 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 0xfffffd00.

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.

Note: It is recommended that the correct device is selected in the debugger so the debugger can pass the device name to the J-Link DLL which makes it possible for J-Link to detect what is the best reset strategy for the device. Moreover, we recommend that the debugger uses reset type 0 to allow J-Link to dynamically select what reset is the best for the connected device.

5.8.2.1 Type 0: Normal

This is the default strategy. It does whatever is the best way to reset the target device.

If the correct device is selected in the debugger this reset strategy may also perform some special handling which might be necessary for the connected device. This for example is the case for devices which have a ROM bootloader that needs to run after reset and before the user application is started (especially if the debug interface is disabled after reset and needs to be enabled by the ROM bootloader).

For most devices, this reset strategy does the same as reset strategy 8 does:

1. Make sure that the device halts immediately after reset (before it can execute any instruction of the user application) by setting the `VC_CORERESET` in the `DEMCR`.
2. Reset the core and peripherals by setting the `SYSRESETREQ` bit in the `AIRCR`.
3. Wait for the `S_RESET_ST` bit in the `DHCSR` to first become high (reset active) and then low (reset no longer active) afterwards.
4. Clear `VC_CORERESET`.

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.

Note: In most cases it is not recommended to reset the core only since most target applications rely of the reset state of some peripherals (PLL, External memory interface etc.) and may be confused if they boot up but the peripherals are already configured.

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

Basically the same as reset type 8. Performs a reset of core & peripherals and halts the CPU immediately after reset. The ROM bootloader is NOT executed.

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 8: Reset core and peripherals

Performs a reset by setting the `SYSRESETREQ` bit in the `AIRCR`. `VC_CORERESET` in the `DEMCR` is also set to make sure that the CPU is halted immediately after reset and before executing any instruction.

Reset procedure:

1. Make sure that the device halts immediately after reset (before it can execute any instruction of the user application) by setting the `VC_CORERESET` in the `DEMCR`.
2. Reset the core and peripherals by setting the `SYSRESETREQ` bit in the `AIRCR`.
3. Wait for the `S_RESET_ST` bit in the `DHCSR` to first become high (reset active) and then low (reset no longer active) afterwards.
4. Clear `VC_CORERESET`.

This type of reset may fail if:

- J-Link has no connection to the debug interface of the CPU because it is in a low power mode.
- The debug interface is disabled after reset and needs to be enabled by a device internal bootloader. This would cause J-Link to lose communication after reset since the CPU is halted before it can execute the internal bootlader.

5.8.2.10 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.

5.8.2.11 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 µs 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 `JLink.exe`.

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 173.

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 173.

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 173.

Prototype

```
__api__ int JTAG_WriteIR(unsigned int Cmd);
```

5.10.2.8 JTAG_StoreIR()

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 173.

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 173.

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 173.

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);
```

5.10.2.12 JTAG_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);
```

5.10.2.13 JTAG_GetU32()

Description

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

Prototype

```
__api__ int JTAG_GetU32(int BitPos);
```

5.10.2.14 JTAG_WriteClocks()

Description

Writes a given number of clocks.

Prototype

```
__api__ int JTAG_WriteClocks(int NumClocks);
```

5.10.2.15 JTAG_StoreClocks()

Description

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

Prototype

```
__api__ int JTAG_StoreClocks(int NumClocks);
```

5.10.2.16 JTAG_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 173.

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);
```

5.10.2.17 SYS_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 176.

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
CPU	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 *Global DLL constants* on page 176. **Example** `CPU = ARM926EJS;`	W
JTAG_IRPre	Used for JTAG chain configuration. Sets the number 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 number 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 number 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 number of devices which are closer to TDI than the one we want to "communicate with. **Example** `JTAG_DRPost = 0;`	R

Table 5.11: Global DLL variables

Variable	Description	R/W
JTAG_IRLen	IR-Len (in bits) of the device we want to communicate with. **Example** `JTAG_IRLen = 4;`	R
JTAG_TotalIRLen	Computed automatically, based on the values of JTAG_IRPre, JTAG_DRPre, JTAG_IRPost and JTAG_DRPost. **Example** `v = JTAG_TotalIRLen;`	R
JTAG_AllowTAPReset	En-/Disables auto-JTAG-detection of J-Link. Has to be disabled for devices which need some special 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	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;`
		W

JTAG_TRSTPin	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;`
		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	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;`
	`}`
		W

Table 5.11: Global DLL variables

Variable	Description	R/W
EMU_ETB_UseETB	Use ETB instead of RAWTRACE capability of the emulator. Setting this variable in another function as `InitEmu()` does not have any effect. **Example** `EMU_ETB_UseETB = 0;`	R
EMU_ETM_IsPresent	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;`	R/W
EMU_ETM_UseETM	Use ETM as trace source. Setting this variable in another function as `InitEmu()` does not have any effect. **Example** `EMU_ETM_UseETM = 1;`	W
EMU_JTAG_DisableHWTransmissions	Disable use of hardware units for JTAG transmissions 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 variable 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-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_APB_AP);`
	`//`
	`// Use second AP as AHB-AP`
	`// for target communication`
	`//`
	`CORESIGHT_IndexAHBAPToUse = 1;`
		W

Table 5.11: Global DLL variables

Variable	Description	R/W

CORESIGHT_ IndexAPBAPToUse	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;`
		W

MAIN_ResetType	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 overrides the J-Link reset strategies.
	Example
	`if (MAIN_ResetType == 2) {`
	`  [...]`
	`} else {`
	`  [...]`
	`}`
		R

MAIN_IsFirstIdentify	Used to check if this is the first time we are running into InitTarget(). Useful if some init steps only need to be executed once per debug session.**Example**
	`if (MAIN_IsFirstIdentify == 1) {`
	`  [...]`
	`} else {`
	`  [...]`
	`}`
		R

Table 5.11: Global DLL variables

5.10.4 Global DLL constants

Currently there are only global DLL constants to set the global DLL variable 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
- ARM1176IF
- ARM1176JFS
- CORTEX_M0
- 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 expresisons 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.");
  JTAG_Reset();                  // Perform TAP reset and J-Link JTAG auto-detection
  if (JTAG_TotalIRLen != 9) {    // Basic check if JTAG chain information matches
    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
  JTAG_IRPre              = 0; // Cortex-A8 is closest to TDO, no no pre IR bits
  JTAG_IRPost             = 5; // custom device after Cortex-A8 has 5 bits IR len
  JTAG_IRLen              = 4; // We selected the Cortex-A8, it has 4 bits IRLen
  CPU                     = CORTEX_A8; // We are connected to a Cortex-A8
  JTAG_AllowTAPReset      = 1;         // We are allowed to enter JTAG TAP reset
  //
  // 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
  //
  CORESIGHT_CoreBaseAddr = 0x80030000;
}
```

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` in the folder wich contains the JLink.exe and the J-Link DLL (by default the installation folder e.g. "C:\Program Files\SEGGER\JLinkARM_V456\").

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 `-scriptfile` command line parameter, please refer to *J-Link GDB Server* on page 66.

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.
EnableFlashBPs	Enables the FlashPB 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 memory 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.

Table 5.12: Available command line options

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 194.

Example

```
device = AT91SAM7S256
```

5.11.1.2 DisableFlashBPs

This command disables the FlashBP feature.

Syntax

```
DisableFlashBPs
```

5.11.1.3 EnableFlashBPs

This command enables the `FlashBP` feature.

Syntax

```
EnableFlashBPs
```

5.11.1.4 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 `map 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 `map exclude` command should be used as follows:

```
map exclude 0x00080000-0x3FFFFFFF
map exclude 0x40008000-0x7FCFFFFF
map exclude 0x7FD02000-0x7FD02000
map exclude 0x80000000-0xDFFFFFFF
```

5.11.1.5 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 284 for an example.

Syntax

```
map indirectread <StartAddressOfArea>-<EndAddress>
```

Example

```
map indirectread 0x3fffc000-0x3fffcfff
```

5.11.1.6 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 284 for an example.

Syntax

```
map ram <StartAddressOfArea>-<EndAddressOfArea>
```

Example

```
map ram 0x40000000-0x40003fff;
```

5.11.1.7 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
```

5.11.1.8 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
```

5.11.1.9 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
```

5.11.1.10 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
```

5.11.1.11 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 163. 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
```

5.11.1.12 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
```

5.11.1.13 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.
```

5.11.1.14 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

```
SetSysPowerDownOnIdle = 10; // The target CPU is powered-down when there is no
                            // transmission between J-Link and target CPU for at least
10ms
```

5.11.1.15 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 | 1
```

Example

```
SupplyPower = 1
```

5.11.1.16 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
```

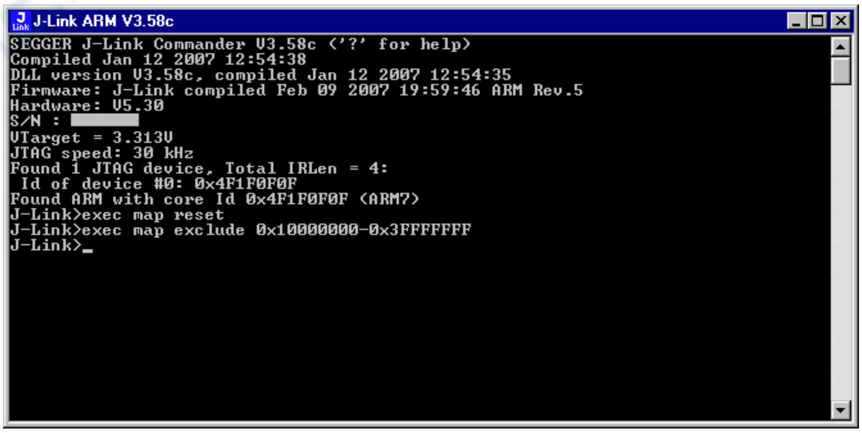
Example

```
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 `exec` supplemented by one of the command strings.



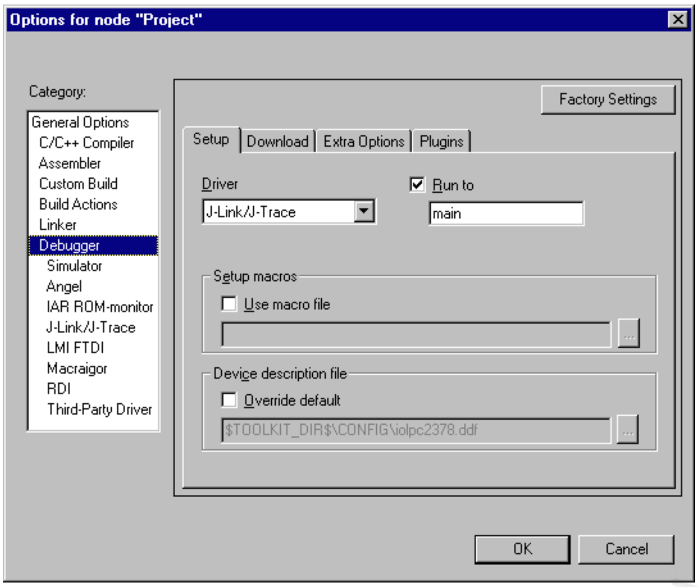
```
SEGGER J-Link Commander V3.58c ('?' for help)
Compiled Jan 12 2007 12:54:38
DLL version V3.58c, compiled Jan 12 2007 12:54:35
Firmware: J-Link compiled Feb 09 2007 19:59:46 ARM Rev.5
Hardware: V5.30
S/N : [redacted]
VTarget = 3.313V
JTAG speed: 30 kHz
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x4F1F0F0F
Found ARM with core Id 0x4F1F0F0F (ARM7)
J-Link>exec map reset
J-Link>exec map exclude 0x10000000-0x3FFFFFFF
J-Link>_
```

Example

```
exec SupplyPower = 1
exec map reset
exec map exclude 0x10000000-0x3FFFFFFF
```

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**.



Driver: J-Link/J-Trace

☑ Run to: main

Setup macros

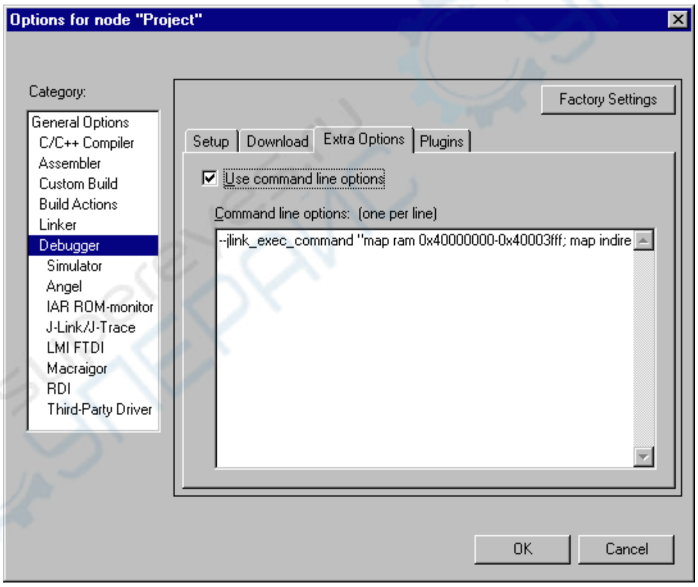
☐ Use macro file

Device description file

☐ Override default

$TOOLKIT_DIR$\CONFIG\iolpc2378.ddf

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.



☑ Use command line options

Command line options: (one per line)

```
--jlink_exec_command "map ram 0x40000000-0x40003fff; map indire
```

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 target 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.

5.14 Virtual COM Port (VCOM)

5.14.1 Configuring Virtual COM Port

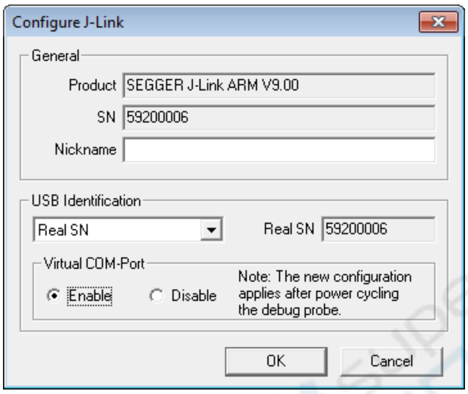
In general, the VCOM feature can be disabled and enabled for debug probes which comes with support for it via J-Link Commander and J-Link Configurator. Below, a small description howto use them to configure the feature is given.

Note: VCOM can only be used when debugging via SWD target interface. Pin 5 = J-Link-Tx (out), Pin 17 = J-Link-Rx (in).

Note: Currently, only J-Link models with hardware version 9 or newer comes with VCOM capabilites.

5.14.1.1 Via 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 enable and disable the VCOM. For more information about the J-Link Configurator, please refer to *J-Link Configurator* on page 131.



Configure J-Link

General

Product: SEGGER J-Link ARM V9.00

SN: 59200006

Nickname:

USB Identification

Real SN

Real SN: 59200006

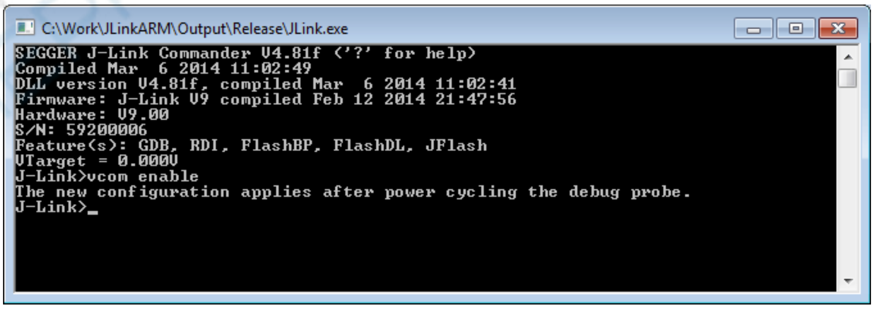
Virtual COM-Port

● Enable ○ Disable

Note: The new configuration applies after power cycling the debug probe.

5.14.1.2 Via J-Link Commander

Simply start J-Link Commander, which is part of the J-Link software and documentation package and enter the `vcom enable|disable` command as in the screenshot below. After changing the configuration a power on cycle of the debug probe is necessary in order to use the new configuration. For feature information about how to use the J-Link Commander, please refer to *J-Link Commander (Command line tool)* on page 63.



C:\Work\JLinkARM\Output\Release\JLink.exe

```
SEGGER J-Link Commander V4.81f ('?' for help)
Compiled Mar  6 2014 11:02:49
DLL version V4.81f, compiled Mar  6 2014 11:02:41
Firmware: J-Link V9 compiled Feb 12 2014 21:47:56
Hardware: V9.00
S/N: 59200006
Feature(s): GDB, RDI, FlashBP, FlashDL, JFlash
VTarget = 0.000V
J-Link>vcom enable
The new configuration applies after power cycling the debug probe.
J-Link>_
```

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.

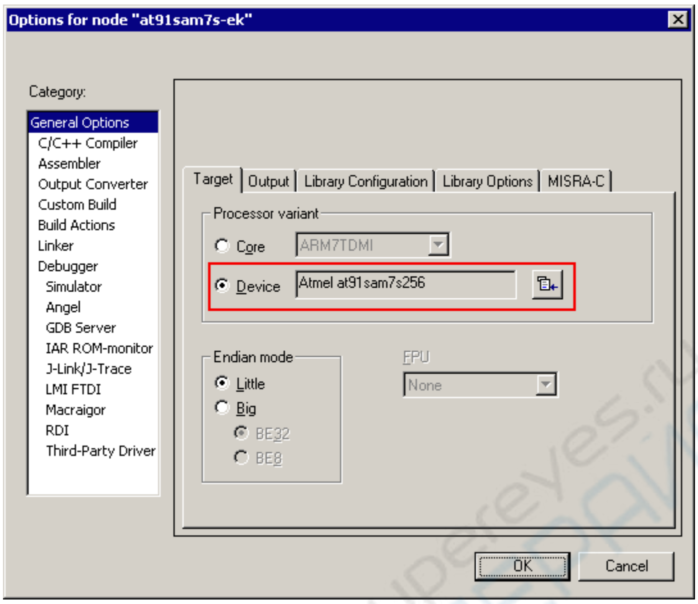
6.4 Setup for various debuggers (internal flash)

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**.



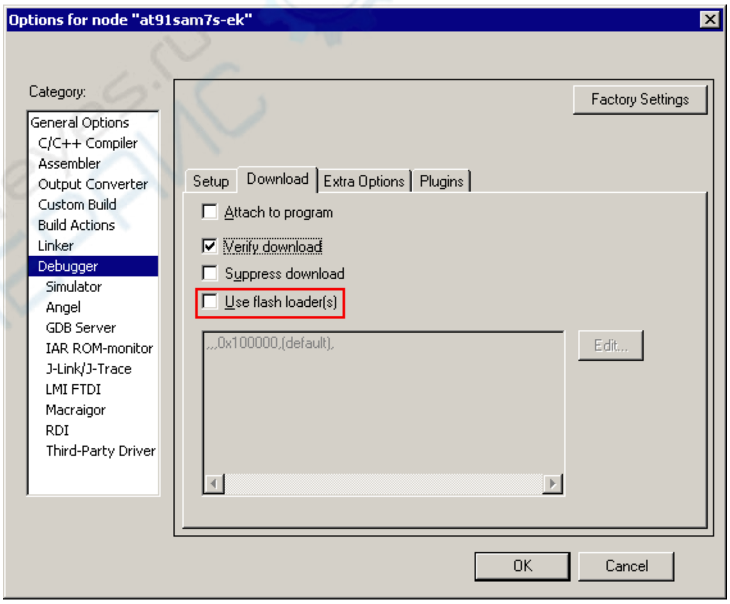
Options for node "at91sam7s-ek"

Processor variant
○ Core ARM7TDMI
● Device Atmel at91sam7s256

Endian mode
● Little
○ Big
● BE32
○ BE8

FPU None

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->Debugger->Download** has to be disabled, as shown below.



Options for node "at91sam7s-ek"

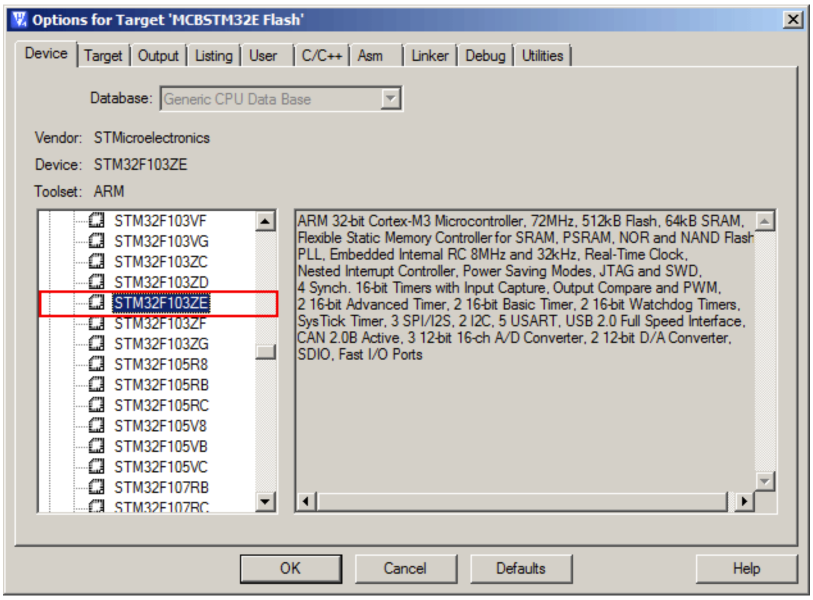
□ Attach to program
☑ Verify download
□ Suppress download
□ Use flash loader(s)

,,,0x100000,(default),

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

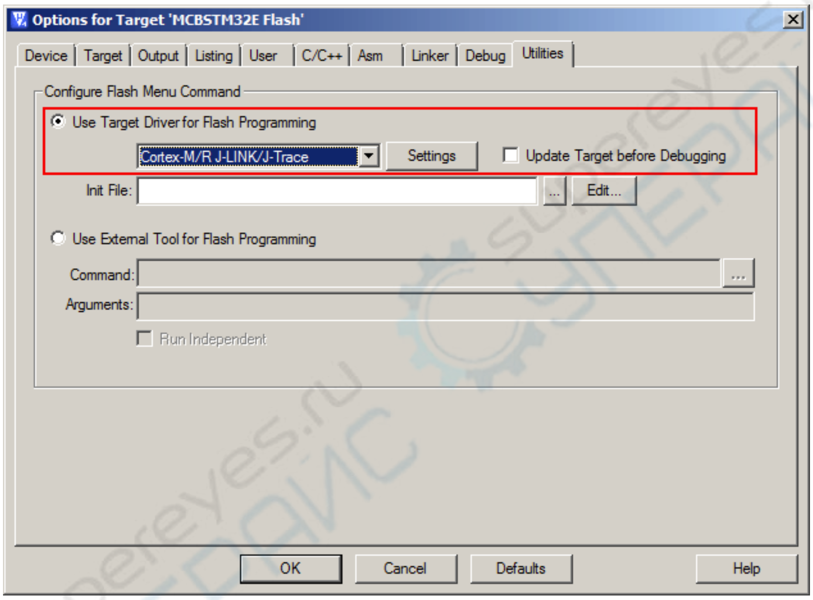
Database: Generic CPU Data Base

Vendor: STMicroelectronics
Device: STM32F103ZE
Toolset: ARM

STM32F103VF
STM32F103VG
STM32F103ZC
STM32F103ZD
STM32F103ZE
STM32F103ZF
STM32F103ZG
STM32F105R8
STM32F105RB
STM32F105RC
STM32F105V8
STM32F105VB
STM32F105VC
STM32F107RB
STM32F107RC

ARM 32-bit Cortex-M3 Microcontroller, 72MHz, 512kB Flash, 64kB SRAM,
Flexible Static Memory Controller for SRAM, PSRAM, NOR and NAND Flash
PLL, Embedded Internal RC 8MHz and 32kHz, Real-Time Clock,
Nested Interrupt Controller, Power Saving Modes, JTAG and SWD,
4 Synch. 16-bit Timers with Input Capture, Output Compare and PWM,
2 16-bit Advanced Timer, 2 16-bit Basic Timer, 2 16-bit Watchdog Timers,
SysTick Timer, 3 SPI/I2S, 2 I2C, 5 USART, USB 2.0 Full Speed Interface,
CAN 2.0B Active, 3 12-bit 16-ch A/D Converter, 2 12-bit D/A Converter,
SDIO, Fast I/O Ports

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

● Use Target Driver for Flash Programming

Cortex-M/R J-LINK/J-Trace | Settings | □ Update Target before Debugging

Init File: | ... | Edit...

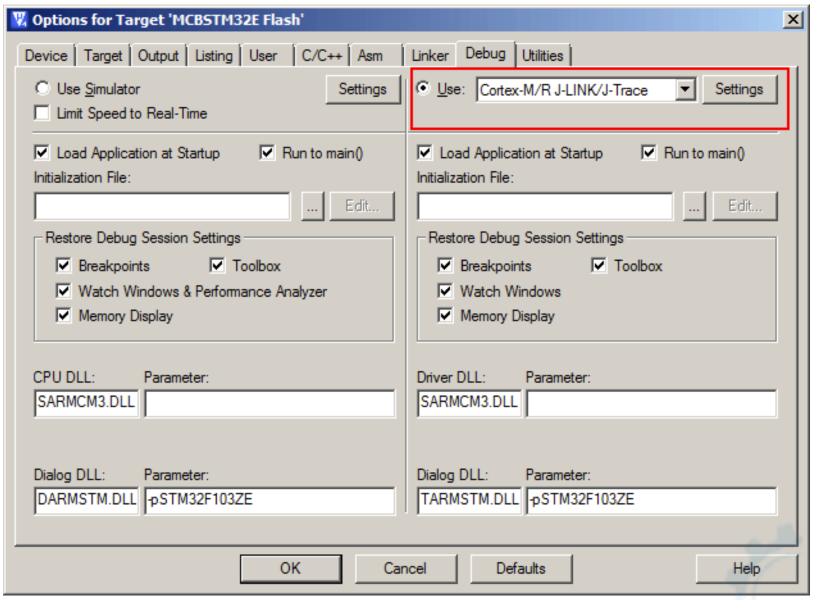
○ Use External Tool for Flash Programming

Command: | ...

Arguments:

□ Run Independent

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**.



Options for Target 'MCBSTM32E Flash'

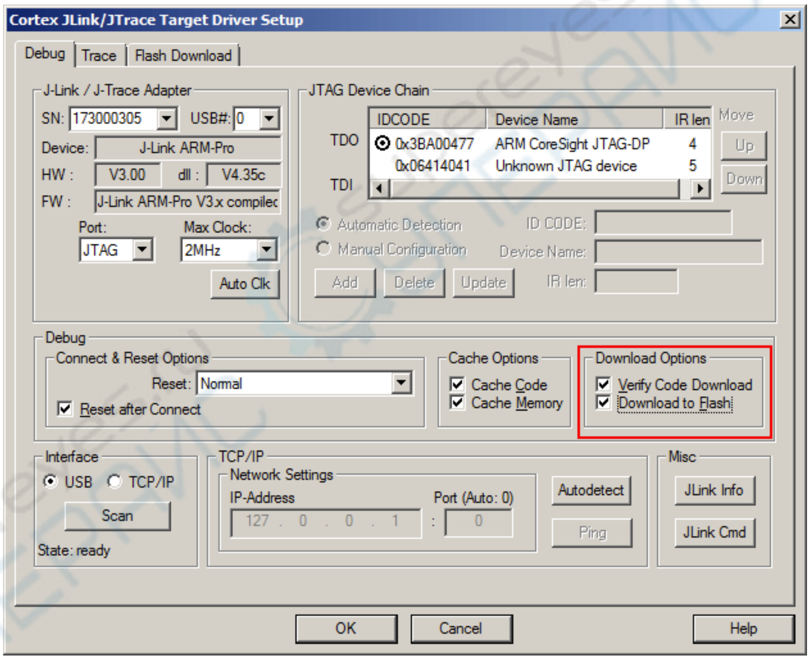
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

○ Use Simulator — Settings
□ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display
CPU DLL: SARMCM3.DLL Parameter:
Dialog DLL: DARMSTM.DLL Parameter: -pSTM32F103ZE

● Use: Cortex-M/R J-LINK/J-Trace — Settings
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows
☑ Memory Display
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TARMSTM.DLL Parameter: -pSTM32F103ZE

OK | Cancel | Defaults | Help

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

Debug | Trace | Flash Download

J-Link / J-Trace Adapter
SN: 173000305 USB#: 0
Device: J-Link ARM-Pro
HW: V3.00 dll: V4.35c
FW: J-Link ARM-Pro V3.x compiled
Port: JTAG Max Clock: 2MHz
Auto Clk

JTAG Device Chain

	IDCODE	Device Name	IR len
TDO	● 0x3BA00477	ARM CoreSight JTAG-DP	4
TDI	0x06414041	Unknown JTAG device	5

Move: Up | Down
● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
Add | Delete | Update IR len:

Debug
Connect & Reset Options
Reset: Normal
☑ Reset after Connect
Cache Options
☑ Cache Code
☑ Cache Memory
Download Options
☑ Verify Code Download
☑ Download to Flash

Interface
● USB ○ TCP/IP
Scan
State: ready

TCP/IP
Network Settings
IP-Address: 127 . 0 . 0 . 1 Port (Auto: 0): 0
Autodetect | Ping

Misc
JLink Info | JLink Cmd

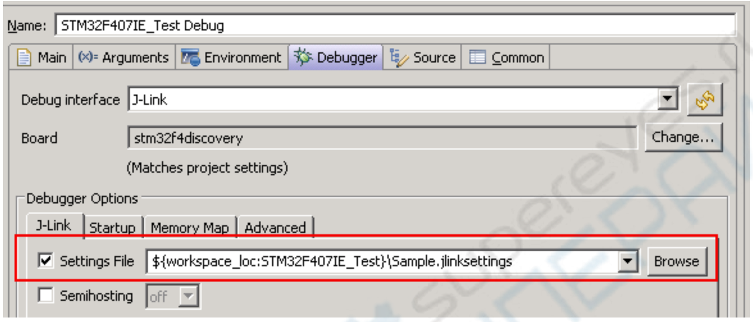
OK | Cancel | Help

6.4.3 Mentor Sourcery CodeBench

To use the J-Link flash download feature in Mentor Sourcery CodeBench, the following steps need to be performed:

Current versions of Sourcery CodeBench do not pass the device name selected in CodeBench, to the J-Link DLL. Therefore a device override via J-Link settings file is needed.

- Copy the J-Link settings file template from `$JLINK_INST_DIR$\Samples\JLink\SettingsFiles\Sample.jlinksettings` to the directory where the CodeBench project is located.
- Open the `Sample.jlinksettings` in a text editor and scroll to the `[FLASH]` section
- Change the line `Device="UNSPECIFIED"` to the device name that shall be selected (keep the quotation marks). A list of valid device names can be found here: *http://www.segger.com/jlink_supported_devices.html* (List of known devices)
- Change the line `Override = 0` to `Override = 1`
- Select the settings file to be used in Sourcery CodeBench:



Name: STM32F407IE_Test Debug

Debug interface: J-Link

Board: stm32f4discovery

(Matches project settings)

☑ Settings File: ${workspace_loc:STM32F407IE_Test}\Sample.jlinksettings

□ Semihosting: off

Additional steps for enabling Flash Breakpoints feature

By default, Mentor Sourcery CodeBench does not allow the user to use the J-Link unlimited number of breakpoints in flash feature, since it only allows hardware breakpoints being set, by default. Enabling this feature, requires an additional tweak in the J-Link settings file:

- Make sure that all steps from *Mentor Sourcery CodeBench* on page 198, to enable flash download, have been performed.
- Make sure that Sourcery CodeBench uses a J-Link DLL with version V4.85d or later. If an earlier version is used, this tweak does not work. To update the DLL used by CodeBench, copy the J-Link DLL from the J-Link installation directory to: C:\Tool\C\Mentor\CodeBench\bin\arm-none-eabi-jlinkarm.dll
- Open the settings file in a text editor and scroll to the `[BREAKPOINTS]` section.
- Add the line: `ForceImpTypeAny = 1`
- Make sure that CodeBench uses the settings file.

6.4.4 J-Link GDB Server

The configuration for the J-Link GDB Server is done by the `.gdbinit` file. The following command has to be added to the `.gdbinit` 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 194. For more information about the GDB monitor commands please refer to *J-Link GDB Server* on page 66.

6.4.5 J-Link Commander

J-Link Commander supports downloading bin files into internal flash memory of popular microcontrollers. In the following, it is explained which steps are necessary to prepare J-Link Commander for download into internal flash memory.

6.4.5.1 Preparing J-Link Commander for flash download

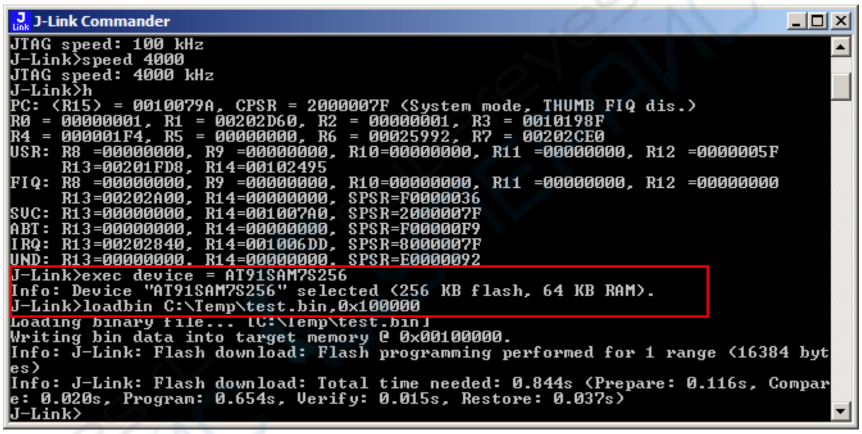
To configure J-Link Commander for flash download simply select the connected device by typing in the following command:

```
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 194. In order to start downloading the binary data file into flash, please type in the following command:

```
loadbin <filename>, <addr>
```

`<Filename>` is the path of the binary data file which should be downloaded into the flash.`<Addr>` is the start address, the data file should be written to.



```
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 = 00202D60, 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=00202A00, R14=00000000, SPSR=F0000036
SVC: R13=00000000, 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... [C:\Temp\test.bin]
Writing bin data into target memory @ 0x00100000.
Info: J-Link: Flash download: Flash programming performed for 1 range (16384 byt
es)
Info: J-Link: Flash download: Total time needed: 0.844s (Prepare: 0.116s, Compar
e: 0.020s, Program: 0.654s, Verify: 0.015s, Restore: 0.037s)
J-Link>
```

6.4.5.2 Converting non-bin files

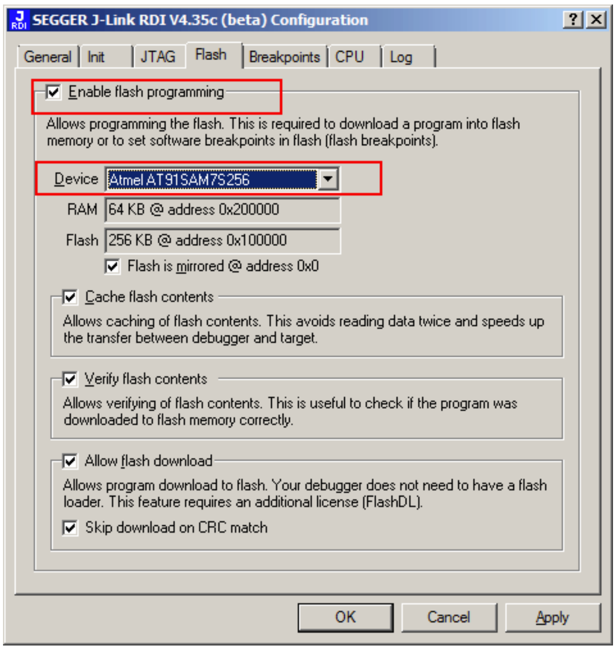
As previously mentioned, J-Link Commander supports programming of bin files into flash memory. If the data file to be programmed is in non-bin format (.hex, .mot, .srec), the file needs to be converted into binary first. There are various free software utilities available which allow data file conversion. Another possibility is to use J-Flash which is part of the J-Link software and documentation package. For converting data files, no J-Flash license is needed.

In order to convert a hex/mot/... file into a bin file using J-Flash, the following needs to be done:

1. Start J-Flash
2. Click **File | Open data file...** to open the data file to be converted in J-Flash.
3. Click **File | Save data file as...**
4. Select Binary file (*.bin) as file type so J-Flash knows that the file shall be saved as binary (bin) file.
5. Click **Save**

6.4.6 J-Link RDI

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



☑ 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 AT91SAM7S256

RAM: 64 KB @ address 0x200000

Flash: 256 KB @ address 0x100000

☑ Flash is mirrored @ address 0x0

☑ Cache flash contents

Allows caching of flash contents. This avoids reading data twice and speeds up the transfer between debugger and target.

☑ Verify flash contents

Allows verifying of flash contents. This is useful to check if the program was downloaded to flash memory correctly.

☑ 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).

☑ Skip download on CRC match

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

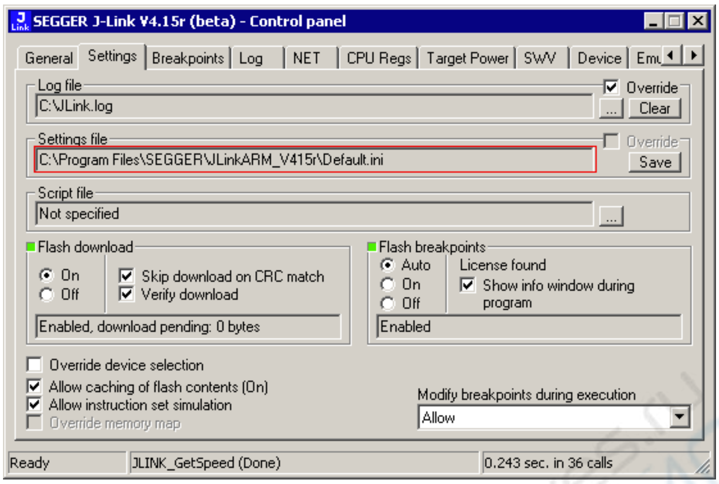
6.5 Setup for various debuggers (CFI flash)

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.



Log file: C:\JLink.log ☑ Override

Settings file: C:\Program Files\SEGGER\JLinkARM_V415r\Default.ini ☐ Override

Script file: Not specified

Flash download: ● On ○ Off ☑ Skip download on CRC match ☑ Verify download
Enabled, download pending: 0 bytes

Flash breakpoints: ● Auto ○ On ○ Off
License found
☑ Show info window during program
Enabled

☐ Override device selection
☑ Allow caching of flash contents (On)
☑ Allow instruction set simulation
☐ Override memory map

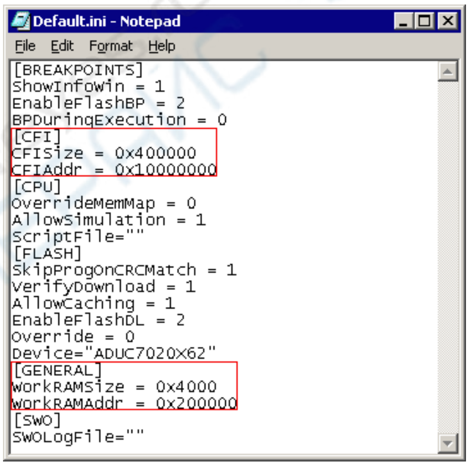
Modify breakpoints during execution: Allow

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:

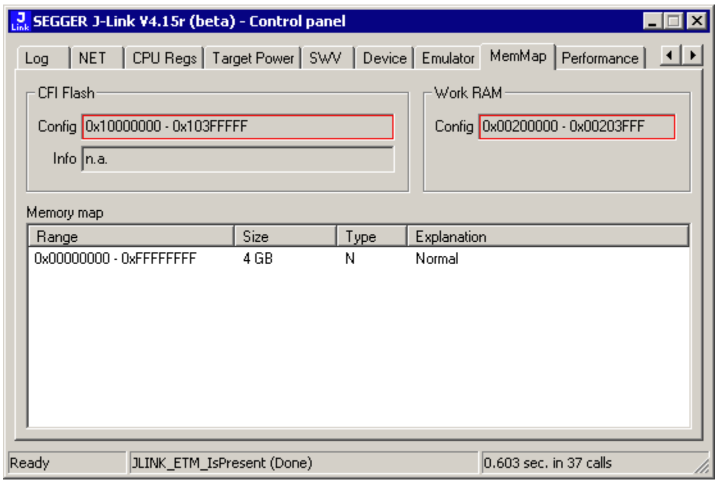
```
[CFI]
CFISize = <FlashSize>
CFIAddr = <FlashAddr>
[GENERAL]
WorkRAMSize = <RAMSize>
WorkRAMAddr = <RAMAddr>
```

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



```
[BREAKPOINTS]
ShowInfoWin = 1
EnableFlashBP = 2
BPDuringExecution = 0
[CFI]
CFISize = 0x400000
CFIAddr = 0x10000000
[CPU]
OverrideMemMap = 0
AllowSimulation = 1
ScriptFile=""
[FLASH]
SkipProgOnCRCMatch = 1
VerifyDownload = 1
AllowCaching = 1
EnableFlashDL = 2
Override = 0
Device="ADUC7020X62"
[GENERAL]
WorkRAMSize = 0x4000
WorkRAMAddr = 0x200000
[SWO]
SWOLogFile=""
```

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.



CFI Flash
Config 0x10000000 - 0x103FFFFF
Info n.a.

Work RAM
Config 0x00200000 - 0x00203FFF

Memory map

Range	Size	Type	Explanation
0x00000000 - 0xFFFFFFFF	4 GB	N	Normal

Ready | JLINK_ETM_IsPresent (Done) | 0.603 sec. in 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 *J-Link GDB Server* on page 66.

6.5.3 J-Link commander

J-Link Commander supports downloading bin files into external CFI flash memory. In the following, it is explained which steps are necessary to prepare J-Link Commander for download into external CFI flash memory based on a sample sequence for a ST STM32F103ZE device:

```
r
speed 1000
exec setcfiflash 0x64000000 - 0x64FFFFFF
exec setworkram 0x20000000 - 0x2000FFFF
w4 0x40021014, 0x00000114 // RCC_AHBENR, FSMC clock enable
w4 0x40021018, 0x000001FD // GPIOD~G clock enable
w4 0x40011400, 0xB4BB44BB // 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 0x40012000, 0x44BBBBBB // GPIOG low config, A10-A15
w4 0x40012004, 0x444B4BB4 // GPIOG high config, NE2 => output
w4 0xA0000008, 0x00001059 // CS control reg 2, 16-bit, write enable, Type: NOR flash
w4 0xA000000C, 0x10000505 // CS2 timing reg (read access)
w4 0xA000010C, 0x10000505 // CS2 timing reg (write access)
speed 4000
mem 0x64000000,100
loadbin C:\STMB672_STM32F103ZE_TestBlinky.bin,0x64000000
mem 0x64000000,100
```

6.5.3.1 Converting non-bin files

As previously mentioned, J-Link Commander supports programming of bin files into flash memory. If the data file to be programmed is in non-bin format (.hex, .mot, .srec), the file needs to be converted into binary first. There are various free software utilities available which allow data file conversion. Another possibility is to use J-Flash which is part of the J-Link software and documentation package. For converting data files, no J-Flash license is needed.

In order to convert a hex/mot/... file into a bin file using J-Flash, the following needs to be done:

1. Start J-Flash
2. Click **File | Open data file...** to open the data file to be converted in J-Flash.
3. Click **File | Save data file as...**
4. Select Binary file (*.bin) as file type so J-Flash knows that the file shall be saved as binary (bin) file.
5. Click **Save**

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 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.

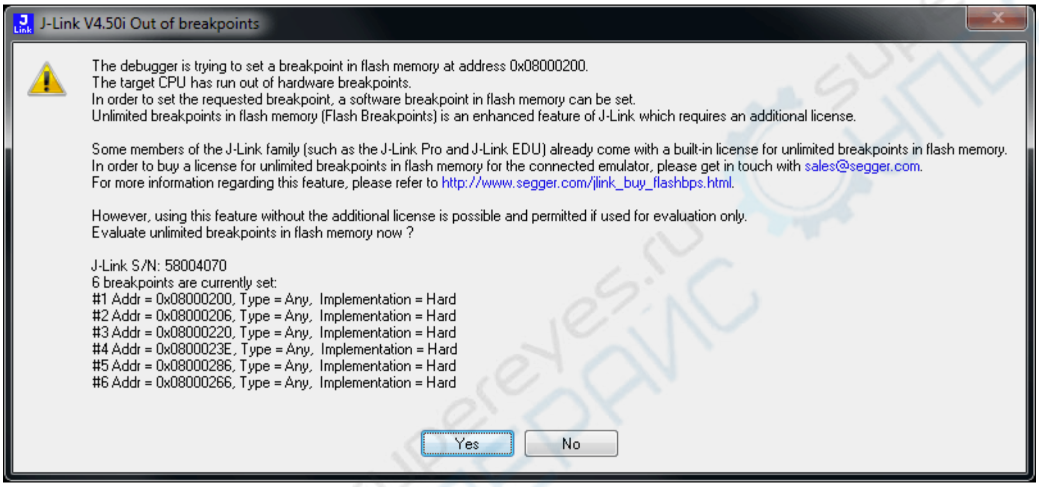
7.2 Licensing

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 Free for evaluation and non-commercial use

In general, the unlimited flash breakpoints feature of the J-Link DLL can be used free of charge for evaluation and non-commercial use.

If used in a commercial project, a license needs to be purchased when the evaluation is complete. There is no time limit on the evaluation period. This feature allows setting an unlimited number of breakpoints even if the application program is located in flash memory, thereby utilizing the debugging environment to its fullest.



The debugger is trying to set a breakpoint in flash memory at address 0x08000200.
The target CPU has run out of hardware breakpoints.
In order to set the requested breakpoint, a software breakpoint in flash memory can be set.
Unlimited breakpoints in flash memory (Flash Breakpoints) is an enhanced feature of J-Link which requires an additional license.

Some members of the J-Link family (such as the J-Link Pro and J-Link EDU) already come with a built-in license for unlimited breakpoints in flash memory.
In order to buy a license for unlimited breakpoints in flash memory for the connected emulator, please get in touch with sales@segger.com.
For more information regarding this feature, please refer to http://www.segger.com/jlink_buy_flashbps.html.

However, using this feature without the additional license is possible and permitted if used for evaluation only.
Evaluate unlimited breakpoints in flash memory now ?

J-Link S/N: 58004070
6 breakpoints are currently set:
#1 Addr = 0x08000200, Type = Any, Implementation = Hard
#2 Addr = 0x08000206, Type = Any, Implementation = Hard
#3 Addr = 0x08000220, Type = Any, Implementation = Hard
#4 Addr = 0x0800023E, Type = Any, Implementation = Hard
#5 Addr = 0x08000286, Type = Any, Implementation = Hard
#6 Addr = 0x08000266, Type = Any, Implementation = Hard

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

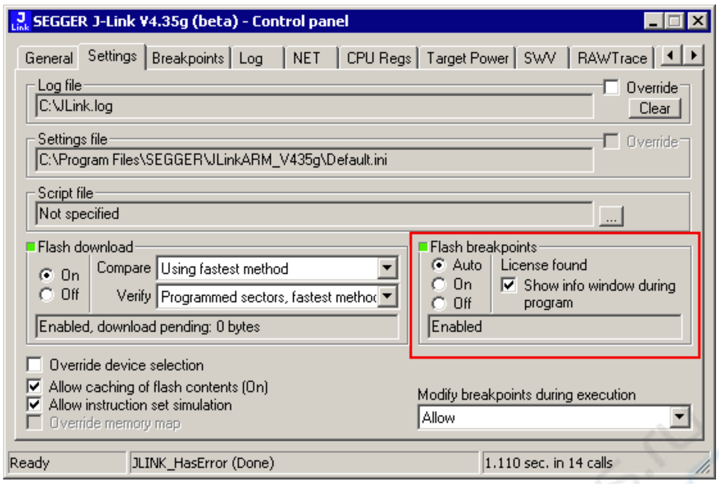
In general, J-Link can be used with any ARM7/9/11, Cortex-M0/M1/M3/M4 and Cortex-A5/A8/R4 core even if it does not provide internal flash.

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 195. If flash breakpoints are available can be verified using the J-Link control panel:



SEGGER J-Link V4.35g (beta) - Control panel

Log file: C:\JLink.log — □ Override — Clear

Settings file: C:\Program Files\SEGGER\JLinkARM_V435g\Default.ini — □ Override

Script file: Not specified

Flash download: ● On ○ Off — Compare: Using fastest method — Verify: Programmed sectors, fastest methoc — Enabled, download pending: 0 bytes

Flash breakpoints: ● Auto ○ On ○ Off — License found — ☑ Show info window during program — Enabled

□ Override device selection
☑ Allow caching of flash contents (On)
☑ Allow instruction set simulation
□ Override memory map

Modify breakpoints during execution: Allow

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
- Freescale Codewarrior
- Mentor Graphics Sourcery CodeBench
- 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.

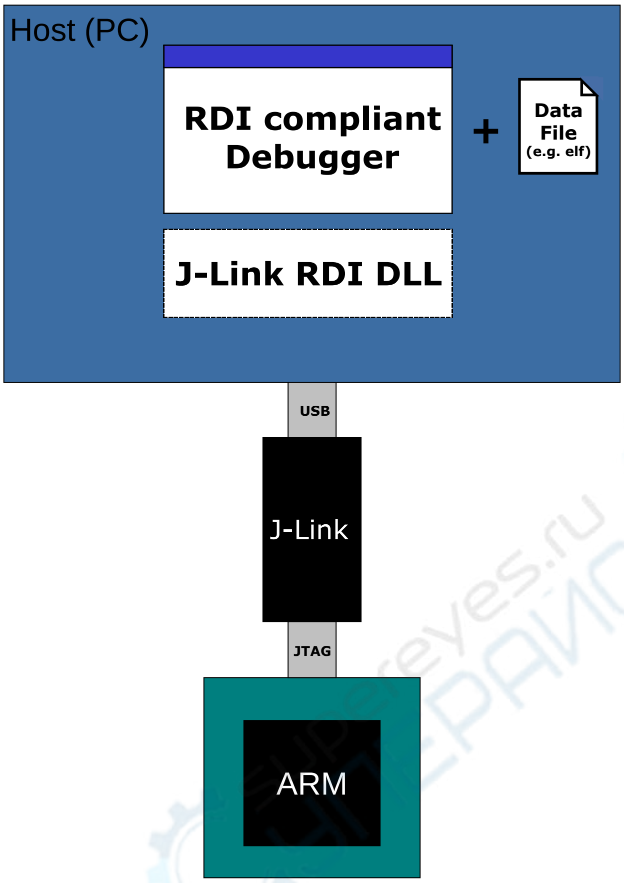
Chapter 8

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.

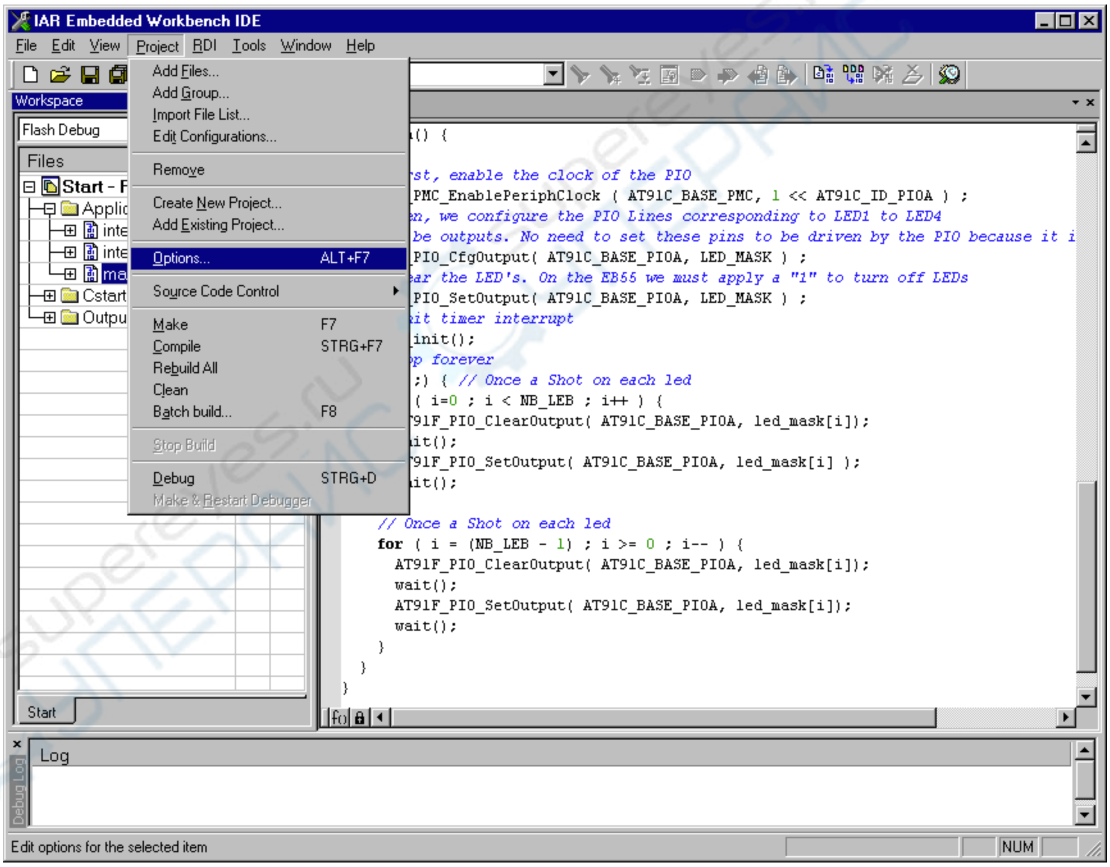
8.3.1.1 Supported software versions

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.

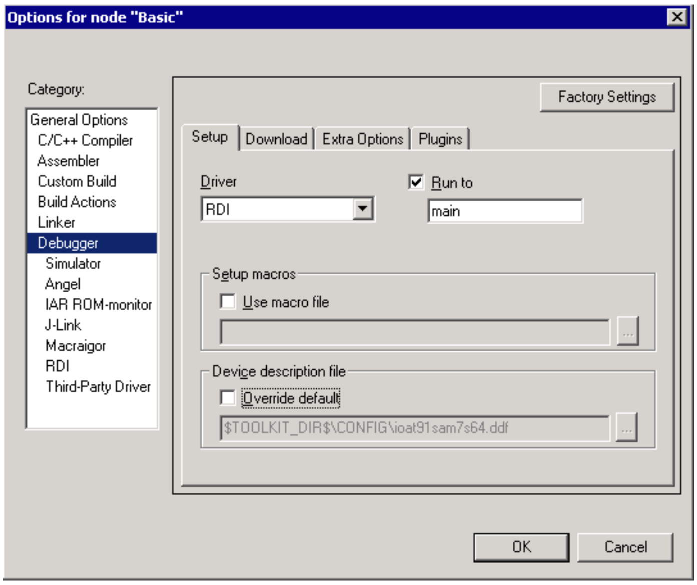


```
() {

st, enable the clock of the PIO
PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_PIOA ) ;
n, we configure the PIO Lines corresponding to LED1 to LED4
be outputs. No need to set these pins to be driven by the PIO because it i
PIO_CfgOutput( AT91C_BASE_PIOA, LED_MASK ) ;
ar the LED's. On the EB55 we must apply a "1" to turn off LEDs
PIO_SetOutput( AT91C_BASE_PIOA, LED_MASK ) ;
it timer interrupt
init();
p forever
;) { // Once a Shot on each led
( i=0 ; i < NB_LEB ; i++ ) {
91F_PIO_ClearOutput( AT91C_BASE_PIOA, led_mask[i]);
it();
91F_PIO_SetOutput( AT91C_BASE_PIOA, led_mask[i] );
it();

    // Once a Shot on each led
    for ( i = (NB_LEB - 1) ; i >= 0 ; i-- ) {
      AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, led_mask[i]);
      wait();
      AT91F_PIO_SetOutput( AT91C_BASE_PIOA, led_mask[i]);
      wait();
    }
  }
}
```

Edit options for the selected item

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

Driver option to RDI.



Driver: RDI

☑ Run to: main

Setup macros

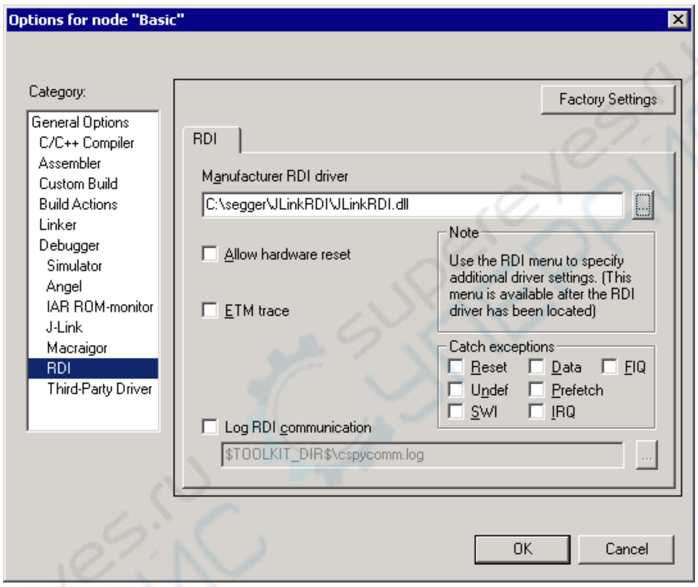
□ Use macro file

Device description file

□ Override default

$TOOLKIT_DIR$\CONFIG\ioat91sam7s64.ddf

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



Manufacturer RDI driver: C:\segger\JLinkRDI\JLinkRDI.dll

□ Allow hardware reset

□ ETM trace

Note

Use the RDI menu to specify additional driver settings. (This menu is available after the RDI driver has been located)

Catch exceptions

□ Reset □ Data □ FIQ

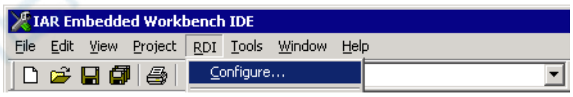
□ Undef □ Prefetch

□ SWI □ IRQ

□ Log RDI communication

$TOOLKIT_DIR$\cspycomm.log

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 230.



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:

Register Index	Assigned register
0	R0
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
25	FAULTMASK
26	BASEPRI
27	BASEPRI_MAX
28	CFBP (CONTROL/FAULT/BASEPRI/PRIMASK)

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

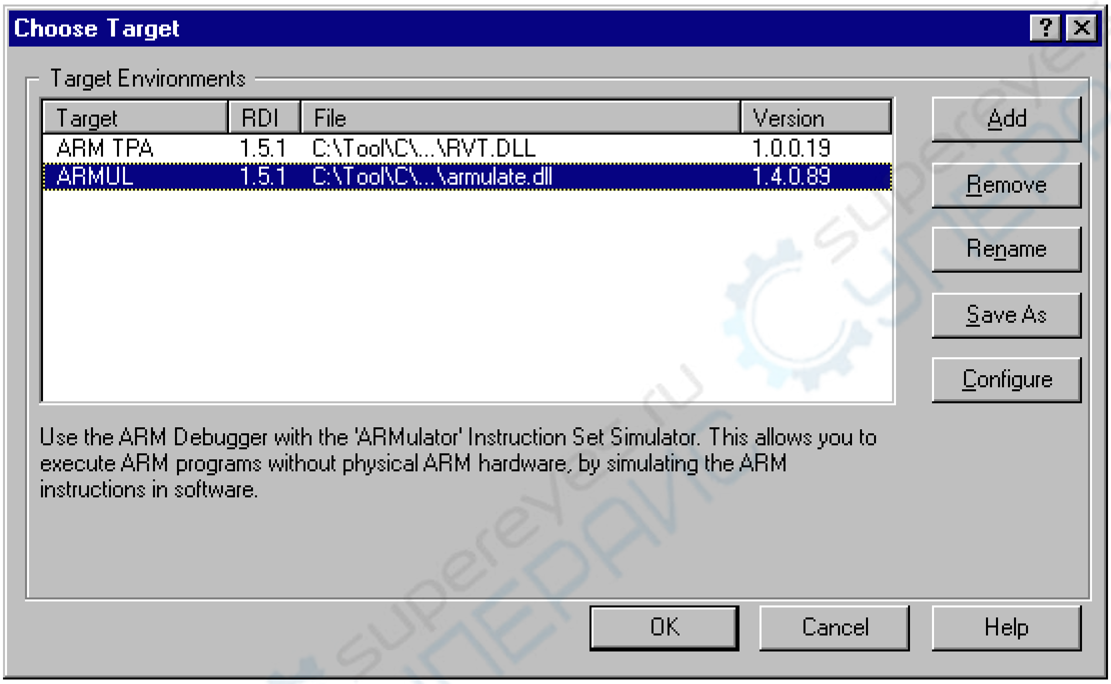
8.3.2 ARM AXD (ARM Developer Suite, ADS)

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

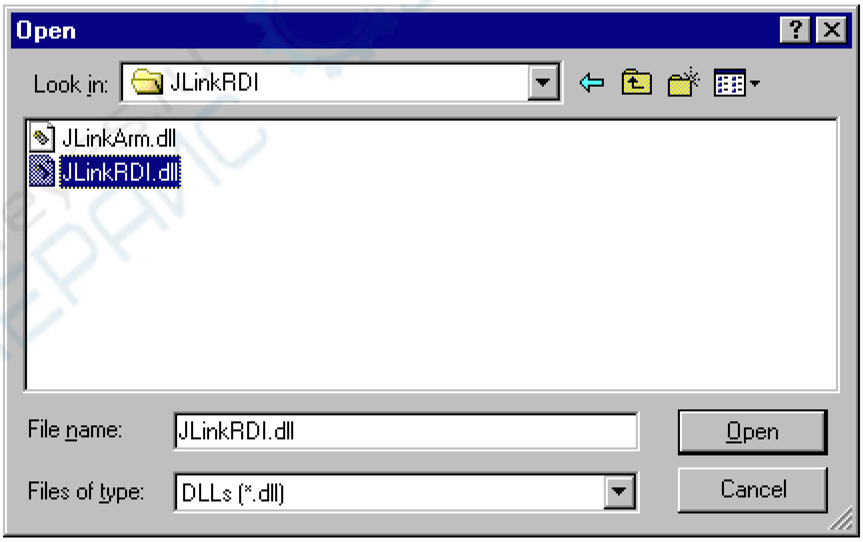
1. Start the ARM debugger and select **Options** | **Configure Target...**. This opens the **Choose Target** dialog box:



Target	RDI	File	Version
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

Use the ARM Debugger with the 'ARMulator' Instruction Set Simulator. This allows you to execute ARM programs without physical ARM hardware, by simulating the ARM instructions in software.

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



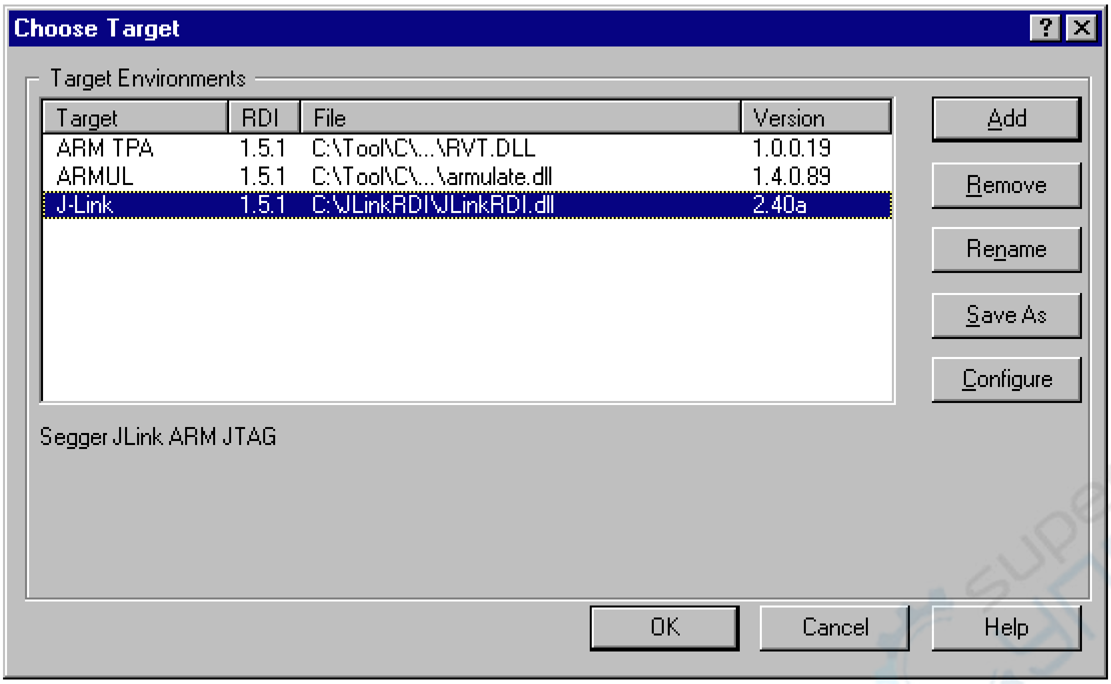
Look in: JLinkRDI

JLinkArm.dll
JLinkRDI.dll

File name: JLinkRDI.dll

Files of type: DLLs (*.dll)

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



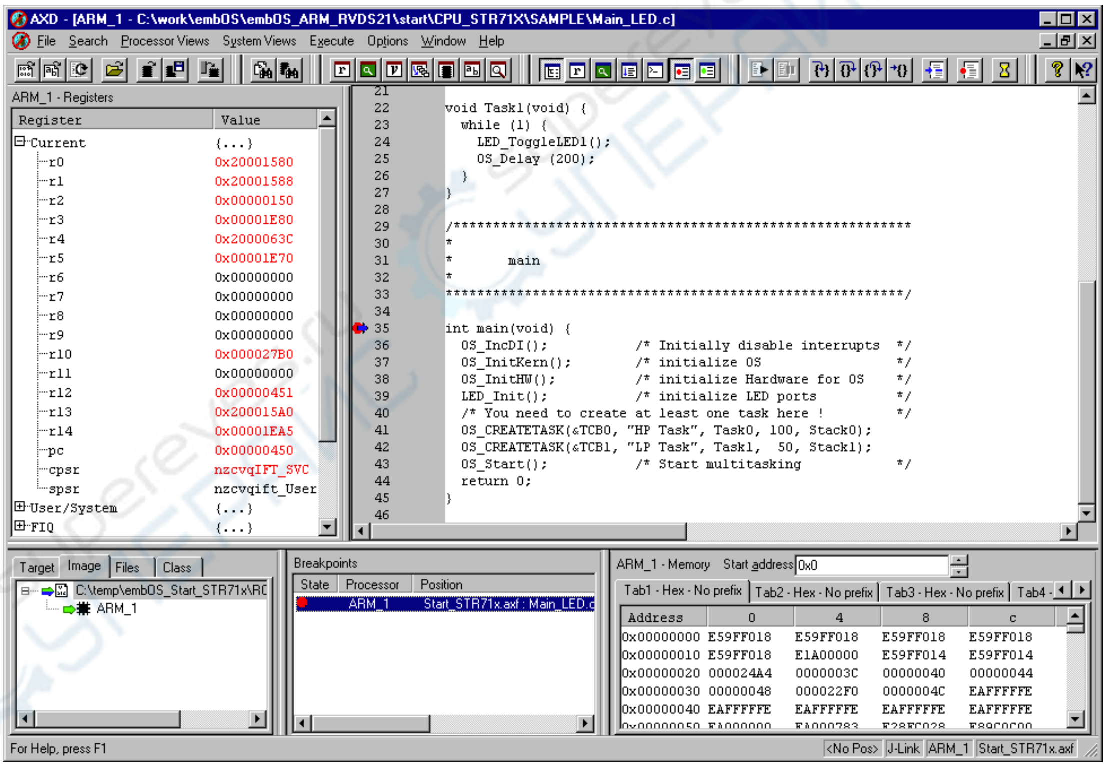
Choose Target

Target Environments

Target	RDI	File	Version
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
J-Link	1.5.1	C:\JLinkRDI\JLinkRDI.dll	2.40a

Segger JLink ARM JTAG

4. Select J-Link and press **OK** to connect to the target via J-Link. For more information about the generic setup of J-Link RDI, please refer to *Configuration* on page 230. After downloading an image to the target board, the debugger window looks as follows:



AXD - [ARM_1 - C:\work\embOS\embOS_ARM_RVDS21\start\CPU_STR71X\SAMPLE\Main_LED.c]

ARM_1 - Registers

Register	Value
Current	{...}
r0	0x20001580
r1	0x20001588
r2	0x00000150
r3	0x00001E80
r4	0x2000063C
r5	0x00001E70
r6	0x00000000
r7	0x00000000
r8	0x00000000
r9	0x00000000
r10	0x000027B0
r11	0x00000000
r12	0x00000451
r13	0x200015A0
r14	0x00001EA5
pc	0x00000450
cpsr	nzcvqIFT_SVC
spsr	nzcvqift_User
User/System	{...}
FIQ	{...}

```

void Task1(void) {
  while (1) {
    LED_ToggleLED1();
    OS_Delay (200);
  }
}

/*********************************************************************
*
*       main
*
*********************************************************************/

int main(void) {
  OS_IncDI();              /* Initially disable interrupts  */
  OS_InitKern();           /* initialize OS                 */
  OS_InitHW();             /* initialize Hardware for OS    */
  LED_Init();              /* initialize LED ports          */
  /* You need to create at least one task here !           */
  OS_CREATETASK(&TCB0, "HP Task", Task0, 100, Stack0);
  OS_CREATETASK(&TCB1, "LP Task", Task1,  50, Stack1);
  OS_Start();              /* Start multitasking            */
  return 0;
}

```

C:\temp\embOS_Start_STR71x\RC
ARM_1

Breakpoints

State	Processor	Position
●	ARM_1	Start_STR71x.axf : Main_LED.c

ARM_1 - Memory Start address 0x0

Address	0	4	8	c
0x00000000	E59FF018	E59FF018	E59FF018	E59FF018
0x00000010	E59FF018	E1A00000	E59FF014	E59FF014
0x00000020	000024A4	0000003C	00000040	00000044
0x00000030	00000048	000022F0	0000004C	EAFFFFFE
0x00000040	EAFFFFFE	EAFFFFFE	EAFFFFFE	EAFFFFFE
0x00000050	EA000000	EA000783	E28FC028	E89C0C00

For Help, press F1 <No Pos> J-Link ARM_1 Start_STR71x.axf

8.3.3 ARM RVDS (RealView developer suite)

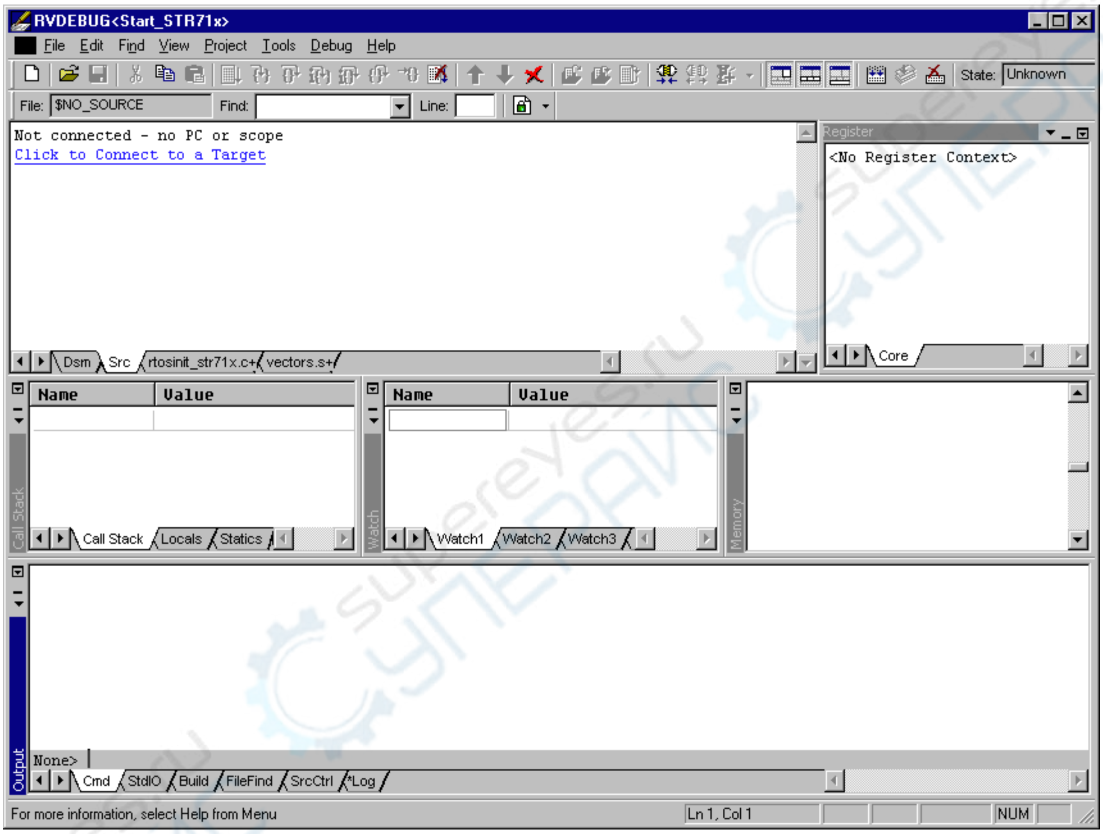
8.3.3.1 Software version

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:

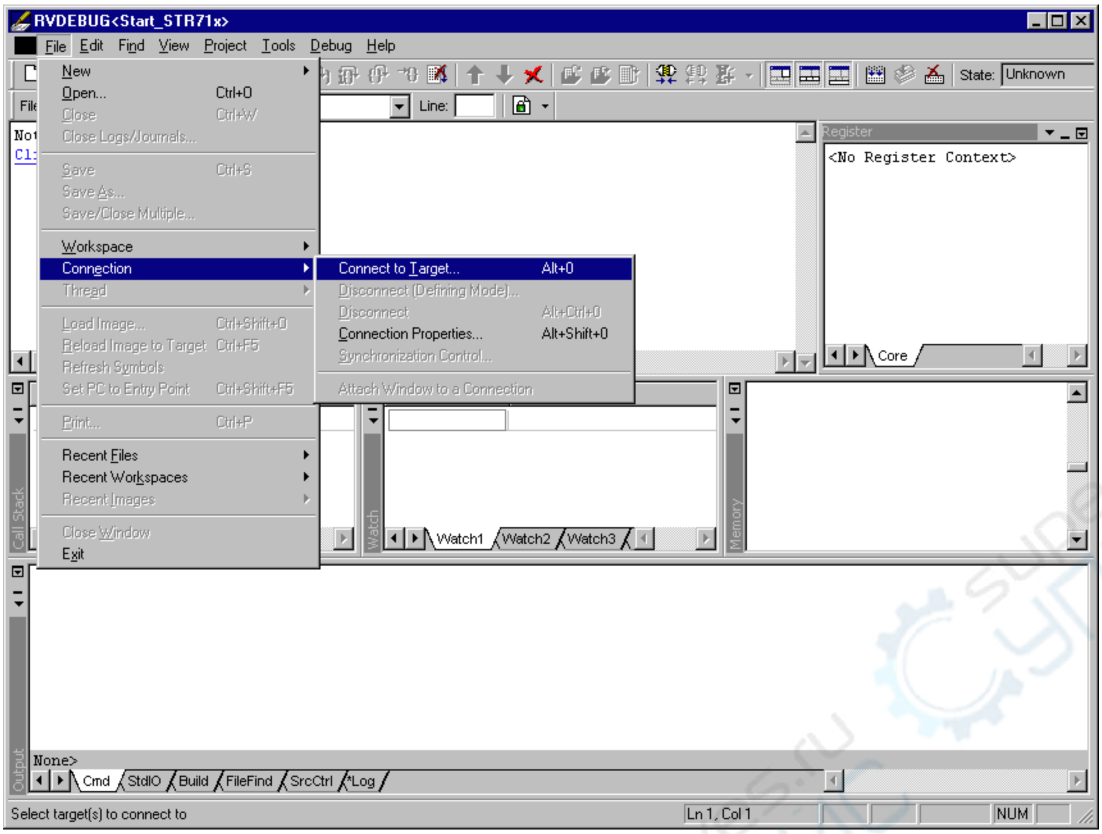


Not connected - no PC or scope
Click to Connect to a Target

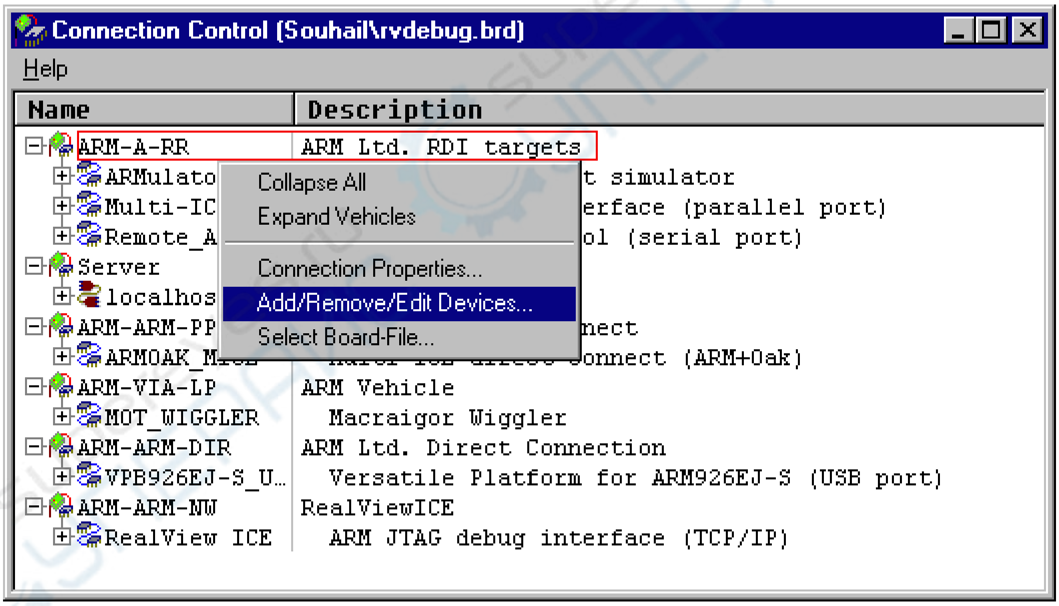
<No Register Context>

For more information, select Help from Menu

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

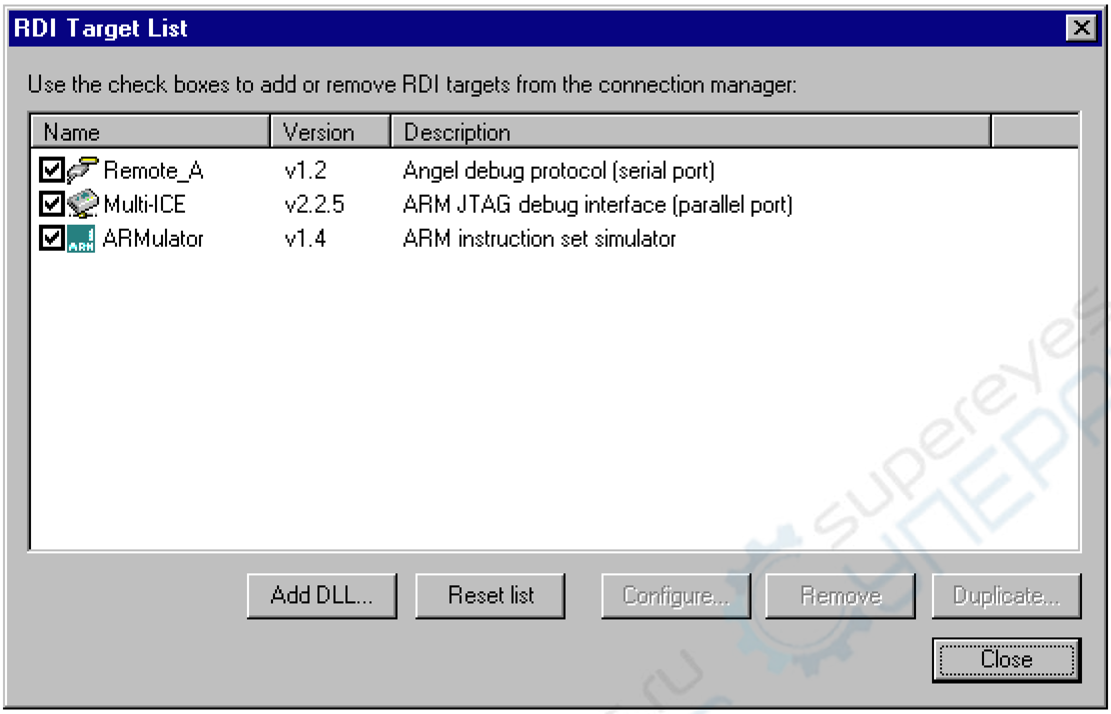


3. In the **Connection Control** dialog use the right mouse click on the first item and select **Add/Remove/Edit Devices**



Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulato	t simulator
Multi-IC	erface (parallel port)
Remote_A	ol (serial port)
Server	
localhos	
ARM-ARM-PP	nect
ARMOAK_M	onnect (ARM+Oak)
ARM-VIA-LP	ARM Vehicle
MOT_WIGGLER	Macraigor Wiggler
ARM-ARM-DIR	ARM Ltd. Direct Connection
VPB926EJ-S_U...	Versatile Platform for ARM926EJ-S (USB port)
ARM-ARM-NW	RealViewICE
RealView ICE	ARM JTAG debug interface (TCP/IP)

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

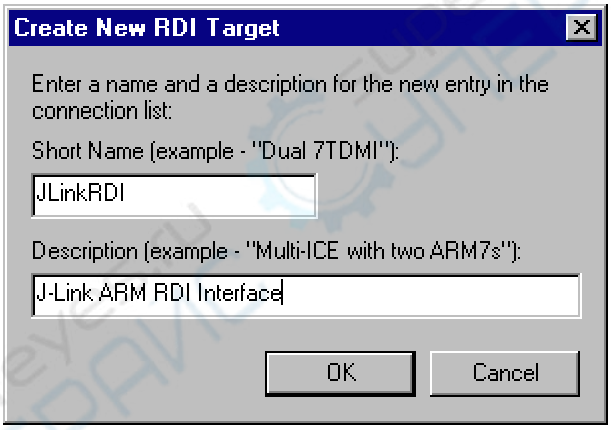


RDI Target List

Use the check boxes to add or remove 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

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 RDI Interface`.



Create New RDI Target

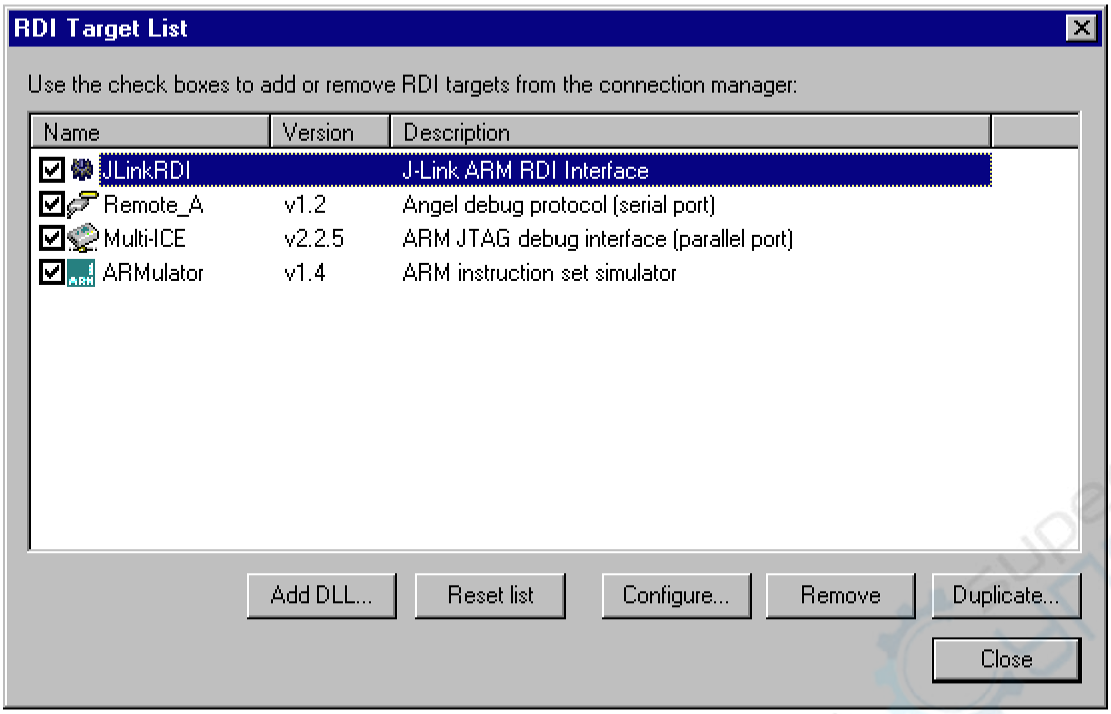
Enter a name and a description for the new entry in the connection list:

Short Name (example - "Dual 7TDMI"): JLinkRDI

Description (example - "Multi-ICE with two ARM7s"): J-Link ARM RDI Interface

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 230.

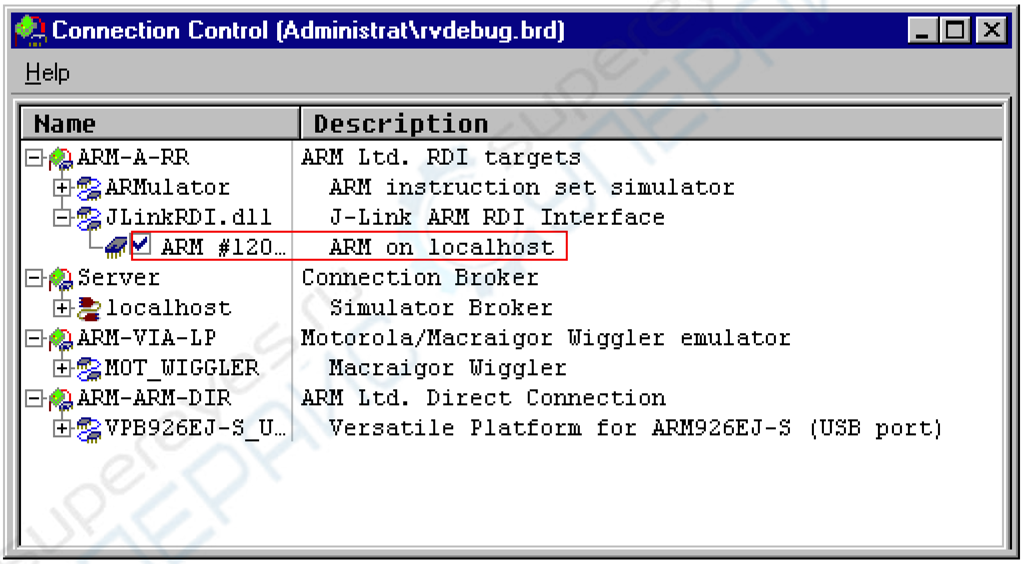


RDI Target List

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)
☑ Multi-ICE	v2.2.5	ARM JTAG debug interface (parallel port)
☑ ARMulator	v1.4	ARM instruction set simulator

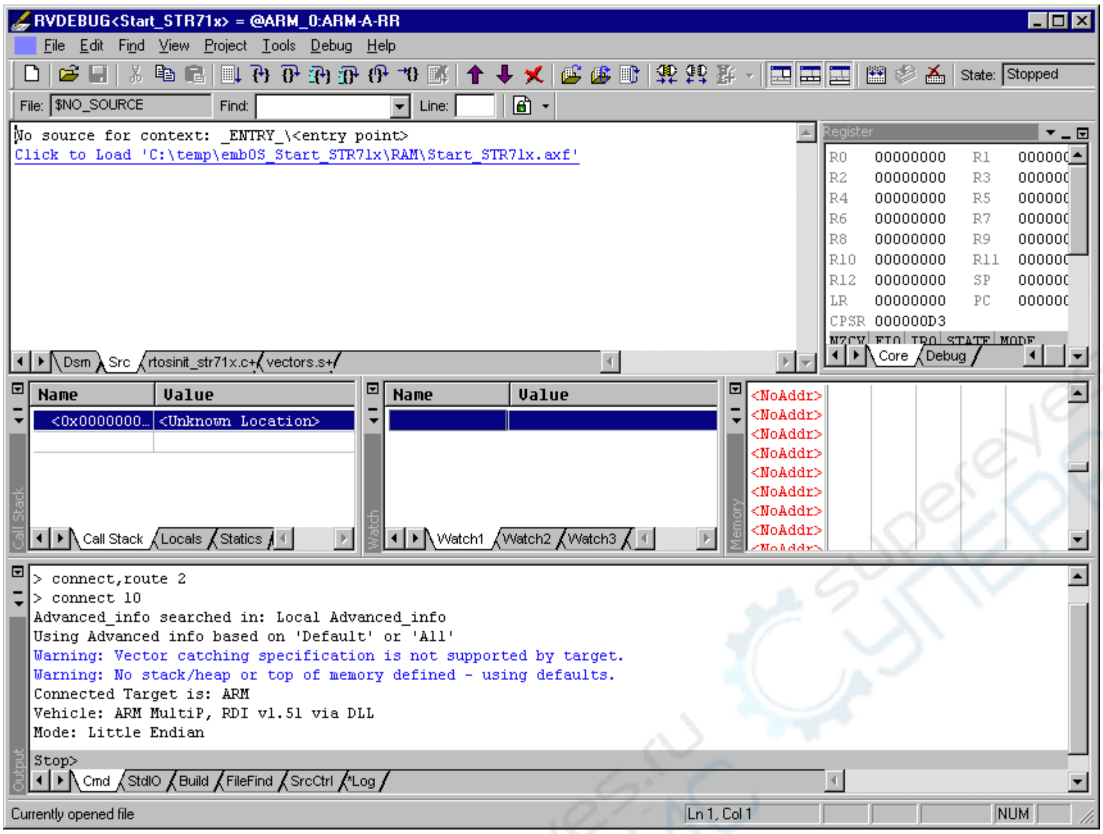
7. Click the **OK** button in the configuration dialog. Now close the **RDI Target List** dialog. Make sure your target hardware is already connected to J-Link.
8. In the **Connection control** dialog, expand the **JLink ARM RDI Interface** and select the ARM_0 Processor. Close the **Connection Control** Window.



Connection Control (Administrat\rvdebug.brd)

Name	Description
⊟ ARM-A-RR	ARM Ltd. RDI targets
⊞ ARMulator	ARM instruction set simulator
⊟ JLinkRDI.dll	J-Link ARM RDI Interface
☑ ARM #120…	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
⊞ VPB926EJ-S_U…	Versatile Platform for ARM926EJ-S (USB port)

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



```
No source for context: _ENTRY_\<entry point>
Click to Load 'C:\temp\embOS_Start_STR71x\RAM\Start_STR71x.axf'
```

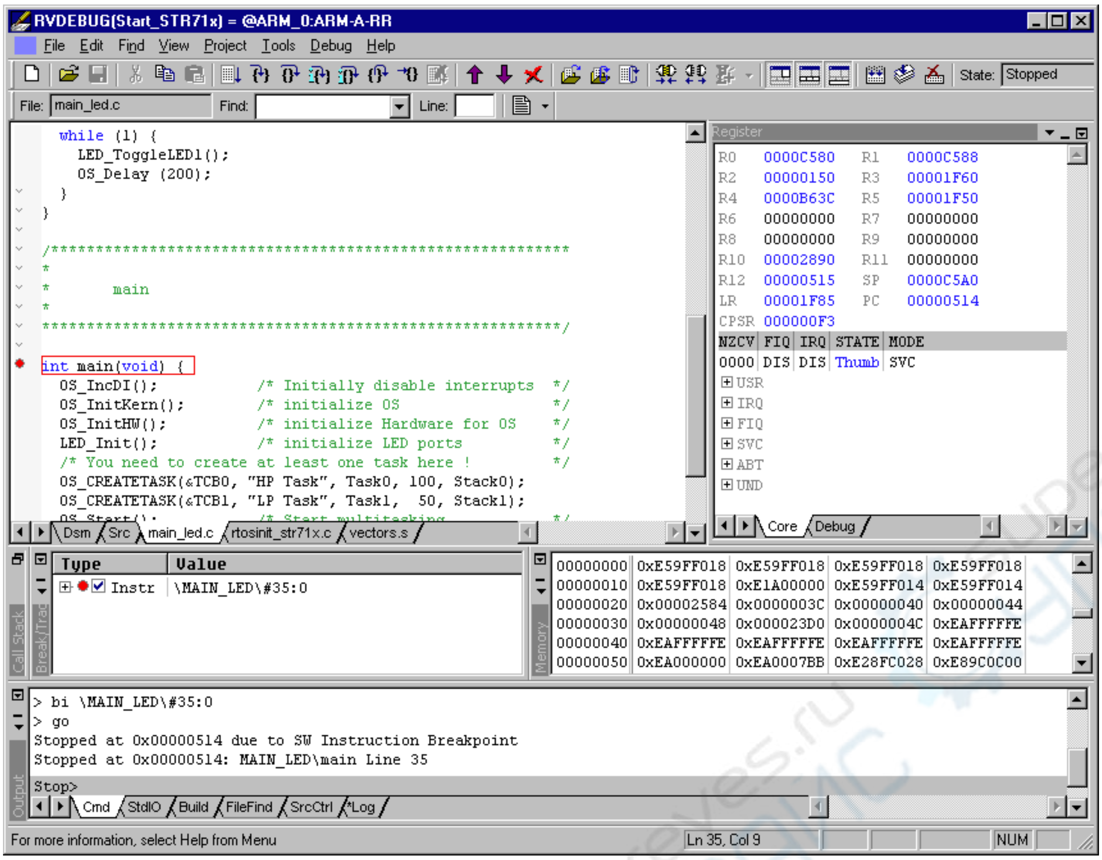
```
R0   00000000   R1   000000
R2   00000000   R3   000000
R4   00000000   R5   000000
R6   00000000   R7   000000
R8   00000000   R9   000000
R10  00000000   R11  000000
R12  00000000   SP   000000
LR   00000000   PC   000000
CPSR 000000D3
NZCV FIQ IRQ STATE MODE
```

Name	Value
<0x0000000…	<Unknown Location>

```
<NoAddr>
<NoAddr>
<NoAddr>
<NoAddr>
<NoAddr>
<NoAddr>
<NoAddr>
<NoAddr>
```

```
> connect,route 2
> connect 10
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
Stop>
```

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



```
  while (1) {
    LED_ToggleLED1();
    OS_Delay (200);
  }
}

/*********************************************************************
*
*       main
*
**********************************************************************/

int main(void) {
  OS_IncDI();             /* Initially disable interrupts  */
  OS_InitKern();          /* initialize OS                 */
  OS_InitHW();            /* initialize Hardware for OS    */
  LED_Init();             /* initialize LED ports          */
  /* You need to create at least one task here !           */
  OS_CREATETASK(&TCB0, "HP Task", Task0, 100, Stack0);
  OS_CREATETASK(&TCB1, "LP Task", Task1,  50, Stack1);
  OS_Start();             /* Start multitasking            */
```

```
R0   0000C580   R1   0000C588
R2   00000150   R3   00001F60
R4   0000B63C   R5   00001F50
R6   00000000   R7   00000000
R8   00000000   R9   00000000
R10  00002890   R11  00000000
R12  00000515   SP   0000C5A0
LR   00001F85   PC   00000514
CPSR 000000F3
NZCV FIQ IRQ STATE MODE
0000 DIS DIS Thumb SVC
⊞USR
⊞IRQ
⊞FIQ
⊞SVC
⊞ABT
⊞UND
```

Type	Value
⊞ ● ☑ Instr	\MAIN_LED\#35:0

```
00000000  0xE59FF018  0xE59FF018  0xE59FF018  0xE59FF018
00000010  0xE59FF018  0xE1A00000  0xE59FF014  0xE59FF014
00000020  0x00002584  0x0000003C  0x00000040  0x00000044
00000030  0x00000048  0x000023D0  0x0000004C  0xEAFFFFFE
00000040  0xEAFFFFFE  0xEAFFFFFE  0xEAFFFFFE  0xEAFFFFFE
00000050  0xEA000000  0xEA0007BB  0xE28FC028  0xE89C0C00
```

```
> bi \MAIN_LED\#35:0
> go
Stopped at 0x00000514 due to SW Instruction Breakpoint
Stopped at 0x00000514: MAIN_LED\main Line 35
Stop>
```

For more information, select Help from Menu   Ln 35, Col 9

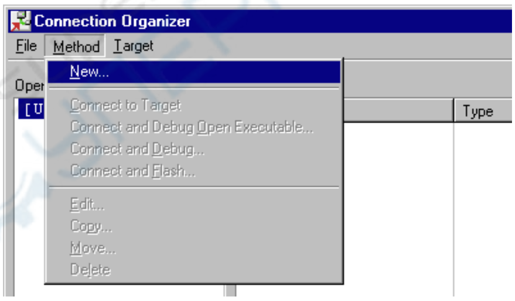
8.3.4 GHS MULTI

8.3.4.1 Software version

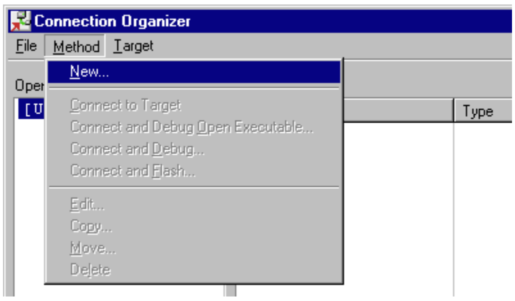
J-Link RDI has been tested with GHS MULTI version 4.07. There should be no problems with other versions of GHS MULTI. All screenshots are taken from GHS MULTI version 4.07.

8.3.4.2 Configuring to use J-Link RDI

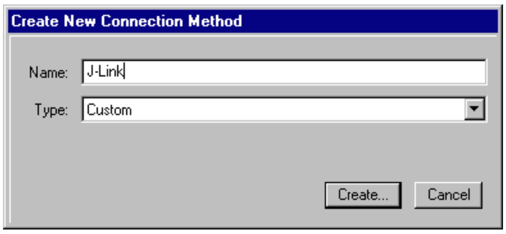
1. Start Green Hills Software MULTI integrated development environment. Click **Connect | Connection Organizer** to open the **Connection Organizer**.



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



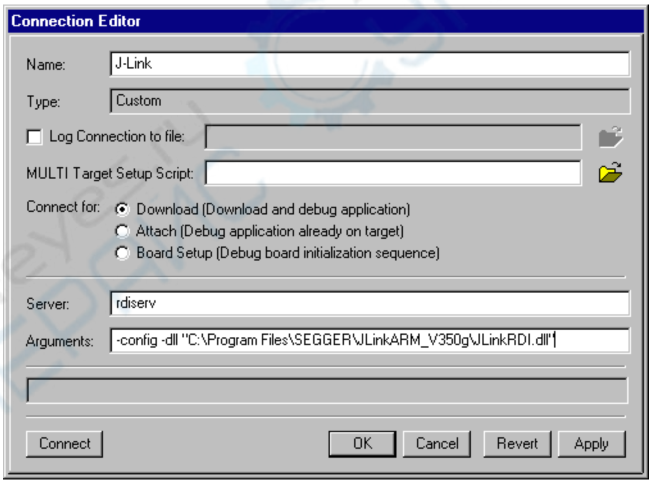
3. The **Create a new Connection Method** will be opened. Enter a name for your configuration in the **Name** field and select **Custom** in the **Type** list. Confirm your choice with the **Create...** button.



Name: J-Link

Type: Custom

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.



Name: J-Link

Type: Custom

□ Log Connection to file:

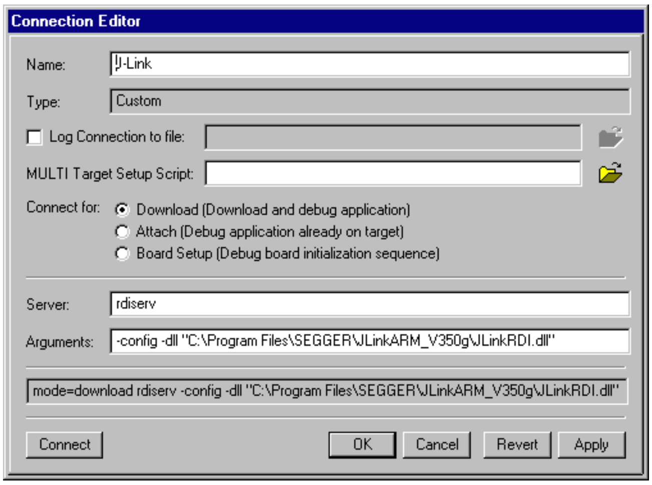
MULTI Target Setup Script:

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"

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



Name: J-Link

Type: Custom

□ Log Connection to file:

MULTI Target Setup Script:

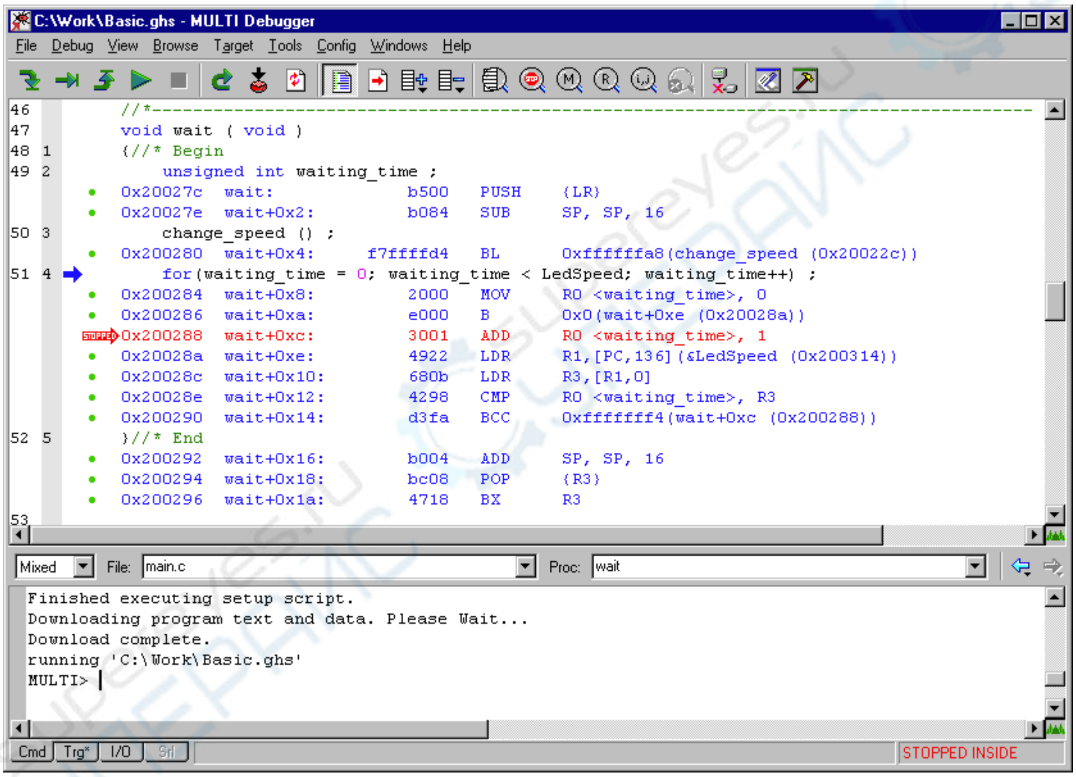
Connect for:
● Download (Download and debug application)
○ Attach (Debug application already on target)
○ Board Setup (Debug board initialization sequence)

Server: rdiserv

Arguments: -config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll"

mode=download rdiserv -config -dll "C:\Program Files\SEGGER\JLinkARM_V350g\JLinkRDI.dll"

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 230.
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.



C:\Work\Basic.ghs - MULTI Debugger
File Debug View Browse Target Tools Config Windows Help

```
//*-------------------------------------------------------------------------------
void wait ( void )
{//* Begin
    unsigned int waiting_time ;
0x20027c  wait:          b500    PUSH   (LR)
0x20027e  wait+0x2:      b084    SUB    SP, SP, 16
    change_speed () ;
0x200280  wait+0x4:  f7ffffd4    BL     0xffffffa8(change_speed (0x20022c))
    for(waiting_time = 0; waiting_time < LedSpeed; waiting_time++) ;
0x200284  wait+0x8:      2000    MOV    R0 <waiting_time>, 0
0x200286  wait+0xa:      e000    B      0x0(wait+0xe (0x20028a))
0x200288  wait+0xc:      3001    ADD    R0 <waiting_time>, 1
0x20028a  wait+0xe:      4922    LDR    R1,[PC,136](&LedSpeed (0x200314))
0x20028c  wait+0x10:     680b    LDR    R3,[R1,0]
0x20028e  wait+0x12:     4298    CMP    R0 <waiting_time>, R3
0x200290  wait+0x14:     d3fa    BCC    0xfffffff4(wait+0xc (0x200288))
}//* End
0x200292  wait+0x16:     b004    ADD    SP, SP, 16
0x200294  wait+0x18:     bc08    POP    (R3)
0x200296  wait+0x1a:     4718    BX     R3
```

Mixed File: main.c Proc: wait

```
Finished executing setup script.
Downloading program text and data. Please Wait...
Download complete.
running 'C:\Work\Basic.ghs'
MULTI>
```

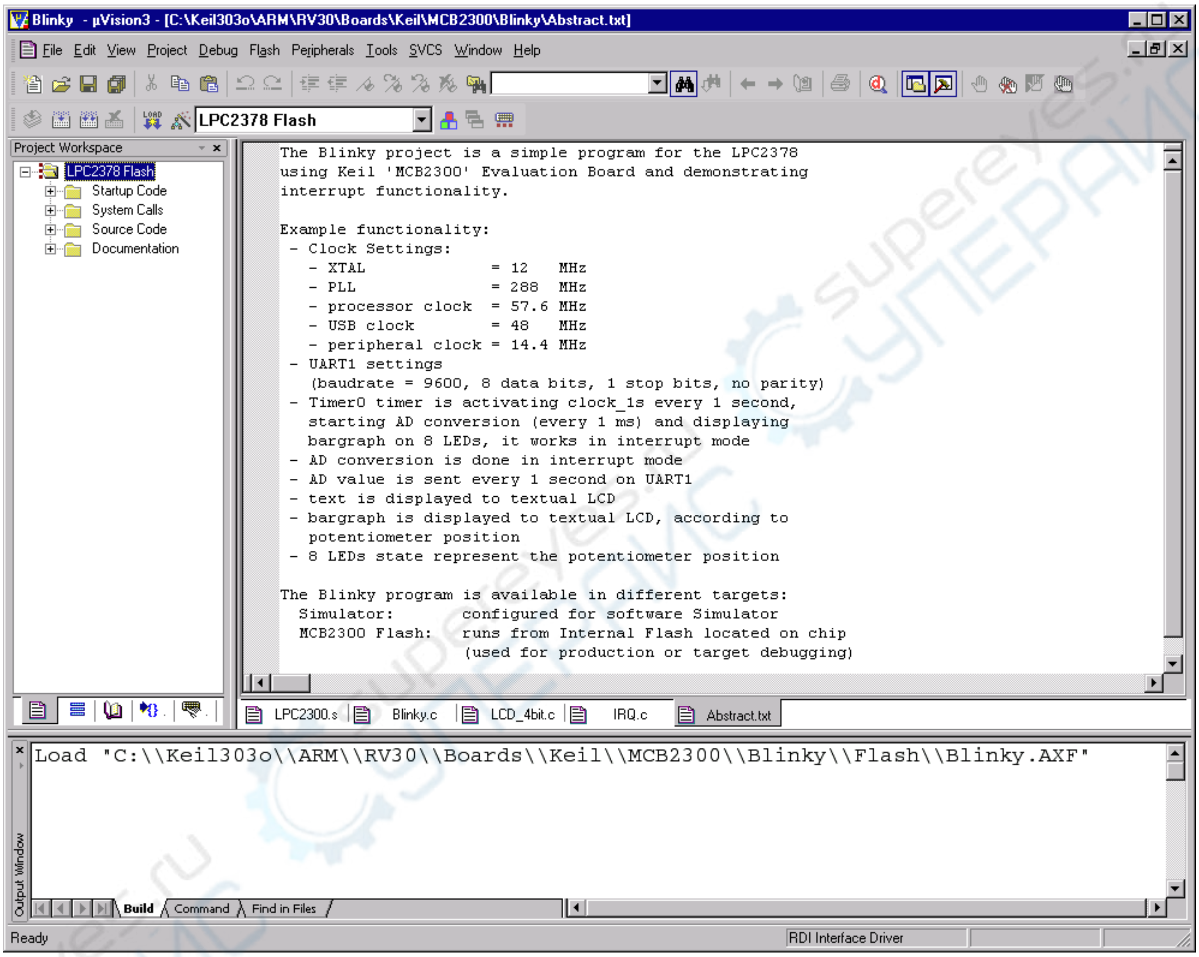
8.3.5 KEIL MDK (µVision IDE)

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 µVision. All screenshots are taken from MDK 3.34.

8.3.5.2 Configuring to use J-Link RDI

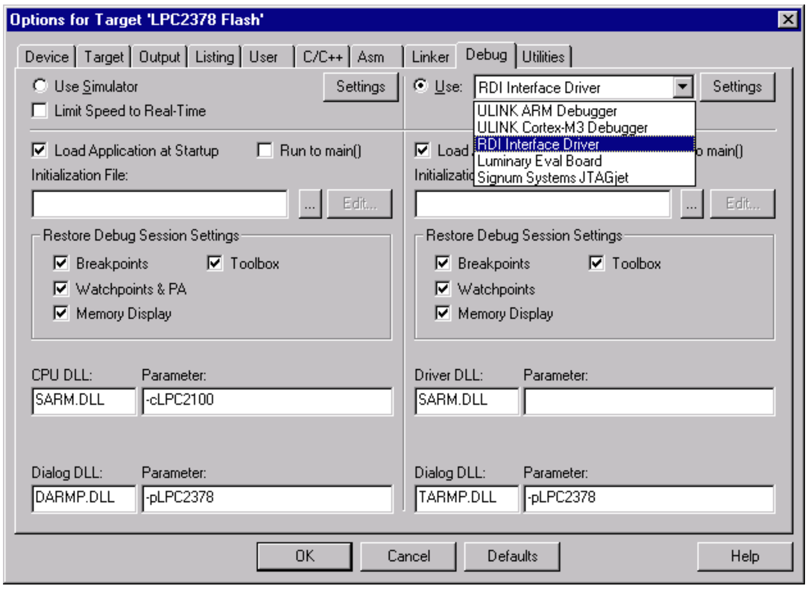
Start KEIL uVision and open the project.



```
The Blinky project is a simple program for the LPC2378
using Keil 'MCB2300' Evaluation Board and demonstrating
interrupt functionality.

Example functionality:
 - Clock Settings:
   - XTAL             = 12   MHz
   - PLL              = 288  MHz
   - processor clock  = 57.6 MHz
   - USB clock        = 48   MHz
   - peripheral clock = 14.4 MHz
 - UART1 settings
   (baudrate = 9600, 8 data bits, 1 stop bits, no parity)
 - Timer0 timer is activating clock_1s every 1 second,
   starting AD conversion (every 1 ms) and displaying
   bargraph on 8 LEDs, it works in interrupt mode
 - AD conversion is done in interrupt mode
 - AD value is sent every 1 second on UART1
 - text is displayed to textual LCD
 - bargraph is displayed to textual LCD, according to
   potentiometer position
 - 8 LEDs state represent the potentiometer position

The Blinky program is available in different targets:
  Simulator:        configured for software Simulator
  MCB2300 Flash:    runs from Internal Flash located on chip
                    (used for production or target debugging)
```

```
Load "C:\\Keil303o\\ARM\\RV30\\Boards\\Keil\\MCB2300\\Blinky\\Flash\\Blinky.AXF"
```

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

○ Use Simulator [Settings]
□ Limit Speed to Real-Time

● Use: RDI Interface Driver [Settings]

☑ Load Application at Startup □ Run to main()
Initialization File:

Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watchpoints & PA
☑ Memory Display

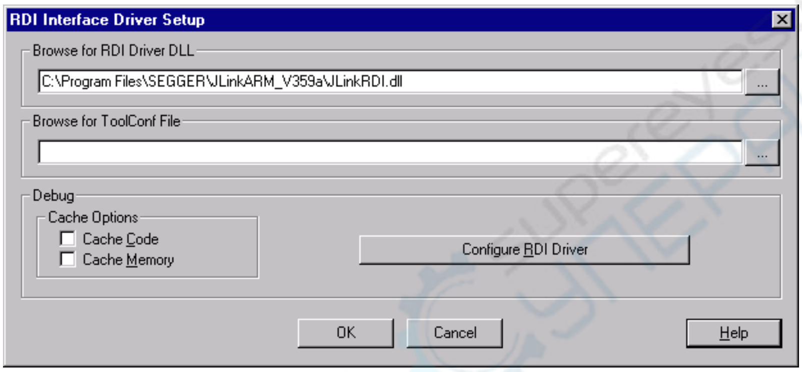
☑ Load ... o main()
Initializatio

Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watchpoints
☑ Memory Display

CPU DLL: SARM.DLL Parameter: -cLPC2100
Dialog DLL: DARMP.DLL Parameter: -pLPC2378

Driver DLL: SARM.DLL Parameter:
Dialog DLL: TARMP.DLL Parameter: -pLPC2378

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

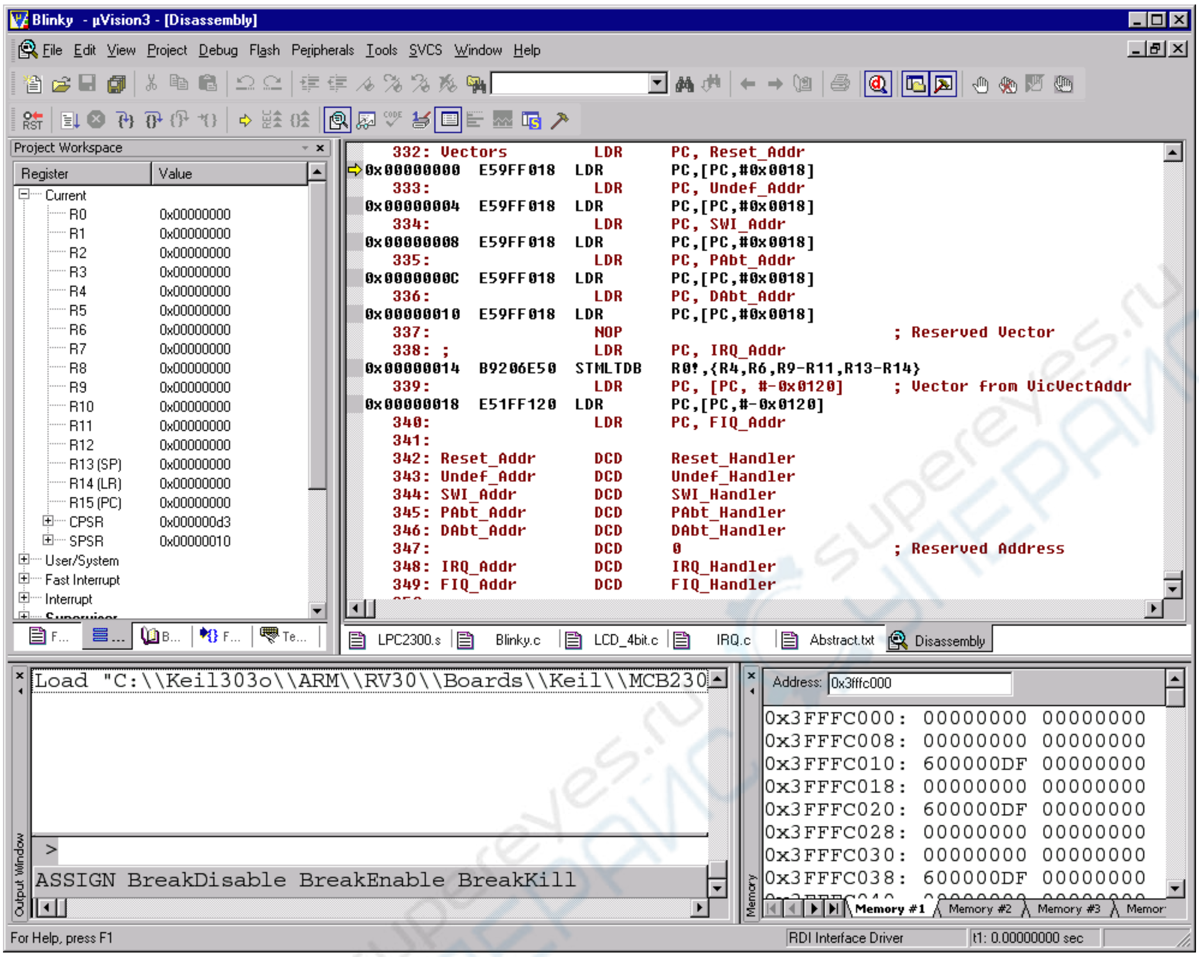
Browse for RDI Driver DLL
C:\Program Files\SEGGER\JLinkARM_V359a\JLinkRDI.dll

Browse for ToolConf File

Debug
Cache Options
□ Cache Code
□ Cache Memory

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 230.

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


Blinky - µVision3 - [Disassembly]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
F... B... F... Te...
LPC2300.s Blinky.c LCD_4bit.c IRQ.c Abstract.txt Disassembly
Output Window
Memory
Memory #1 Memory #2 Memory #3 Memor

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000d3
SPSR	0x00000010
User/System	
Fast Interrupt	
Interrupt	
Supervisor	

```
   332: Vectors        LDR     PC, Reset_Addr
0x00000000  E59FF018  LDR     PC,[PC,#0x0018]
   333:                LDR     PC, Undef_Addr
0x00000004  E59FF018  LDR     PC,[PC,#0x0018]
   334:                LDR     PC, SWI_Addr
0x00000008  E59FF018  LDR     PC,[PC,#0x0018]
   335:                LDR     PC, PAbt_Addr
0x0000000C  E59FF018  LDR     PC,[PC,#0x0018]
   336:                LDR     PC, DAbt_Addr
0x00000010  E59FF018  LDR     PC,[PC,#0x0018]
   337:                NOP                            ; Reserved Vector
   338: ;              LDR     PC, IRQ_Addr
0x00000014  B9206E50  STMLTDB R0!,{R4,R6,R9-R11,R13-R14}
   339:                LDR     PC, [PC, #-0x0120]     ; Vector from VicVectAddr
0x00000018  E51FF120  LDR     PC,[PC,#-0x0120]
   340:                LDR     PC, FIQ_Addr
   341:
   342: Reset_Addr     DCD     Reset_Handler
   343: Undef_Addr     DCD     Undef_Handler
   344: SWI_Addr       DCD     SWI_Handler
   345: PAbt_Addr      DCD     PAbt_Handler
   346: DAbt_Addr      DCD     DAbt_Handler
   347:                DCD     0                      ; Reserved Address
   348: IRQ_Addr       DCD     IRQ_Handler
   349: FIQ_Addr       DCD     FIQ_Handler
```

```
Load "C:\\Keil303o\\ARM\\RV30\\Boards\\Keil\\MCB230
>
ASSIGN BreakDisable BreakEnable BreakKill
```

Address: 0x3fffc000

```
0x3FFFC000: 00000000 00000000
0x3FFFC008: 00000000 00000000
0x3FFFC010: 600000DF 00000000
0x3FFFC018: 00000000 00000000
0x3FFFC020: 600000DF 00000000
0x3FFFC028: 00000000 00000000
0x3FFFC030: 00000000 00000000
0x3FFFC038: 600000DF 00000000
```

For Help, press F1 | RDI Interface Driver | t1: 0.00000000 sec

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 `JLinkRDI.ini`. This file is located in the same directory as `JLinkRDI.dll`.

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 `JLinkARM.dll` as well as the `JLinkRDI.dll` 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 `$PROJ_DIR$\RDI\`), it can make sense to create the directory for the DLLs and configuration file in a subdirectory of the project.

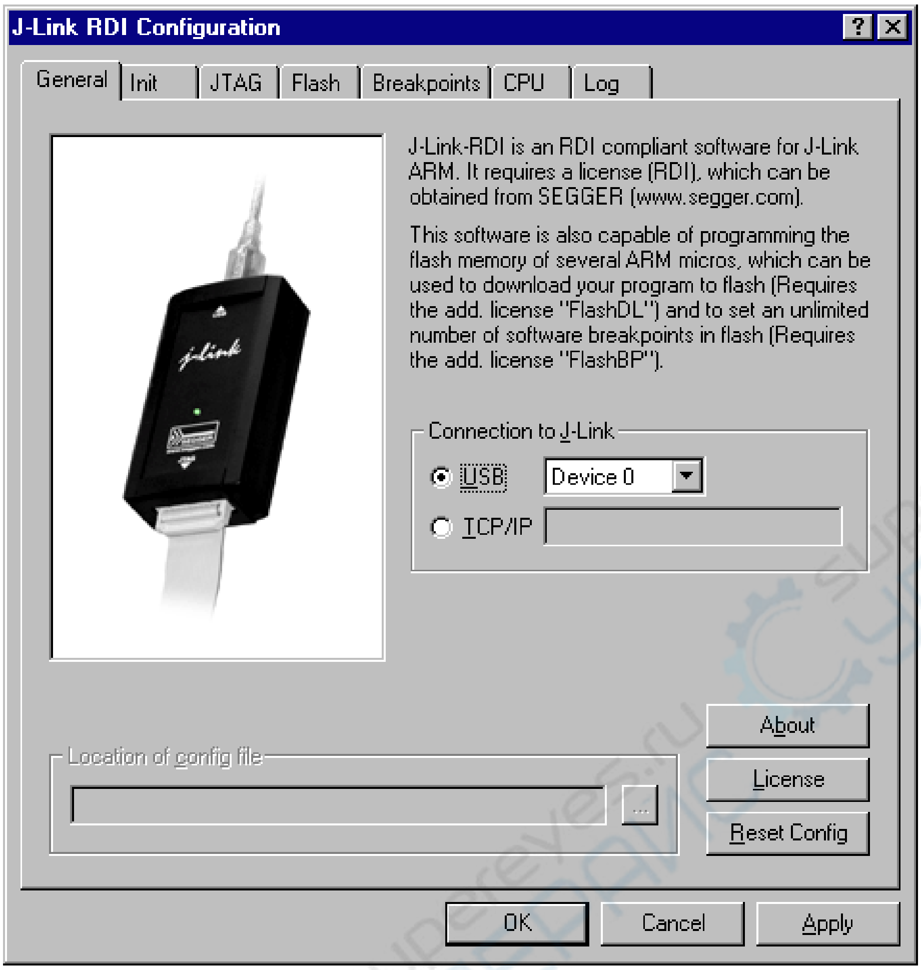
8.4.3 Using mutliple J-Links simulatenously

Same procedure as using different configurations. Each debugger session will use their own instance of the `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 is an RDI compliant software for J-Link ARM. It requires a license (RDI), which can be obtained from SEGGER (www.segger.com).

This software is also capable of programming the flash memory of several ARM micros, which can be used to download your program to flash (Requires the add. license "FlashDL") and to set an unlimited number of software breakpoints in flash (Requires the add. license "FlashBP").

Connection to J-Link

● USB Device 0

○ TCP/IP

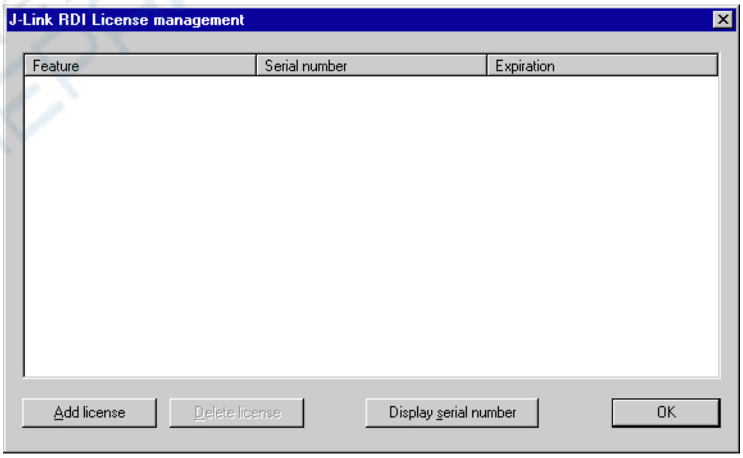
Location of config file

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)

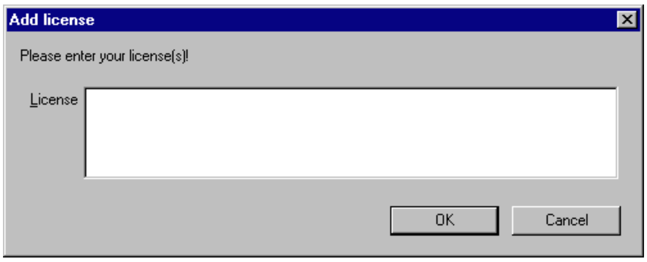
1. The **License** button opens the **J-Link RDI License management** dialog. J-Link RDI requires a valid license.



Feature	Serial number	Expiration

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

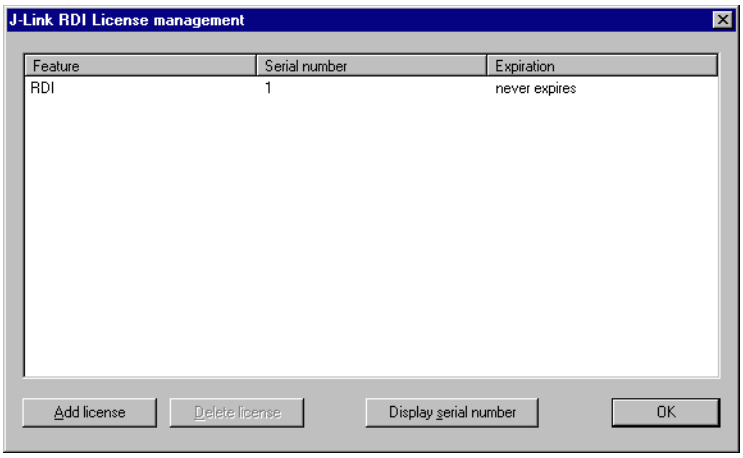
ing the **OK** button.



Please enter your license(s)!

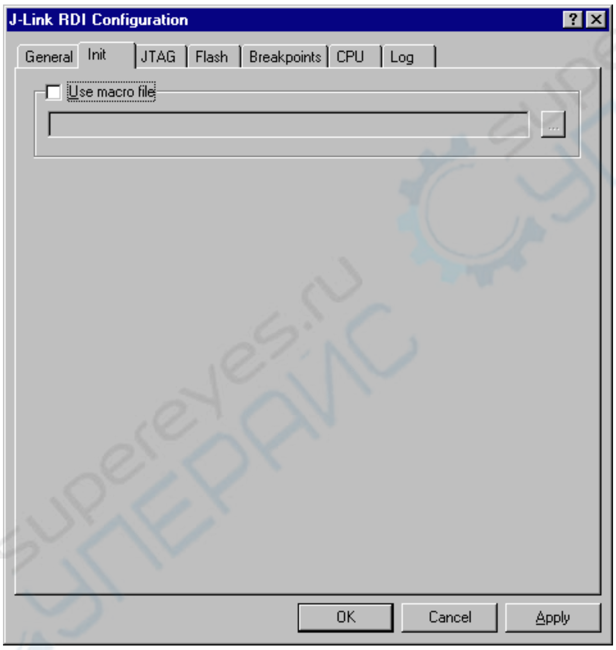
License

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



Feature	Serial number	Expiration
RDI	1	never expires

8.4.4.2 Init tab



☐ Use macro file

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

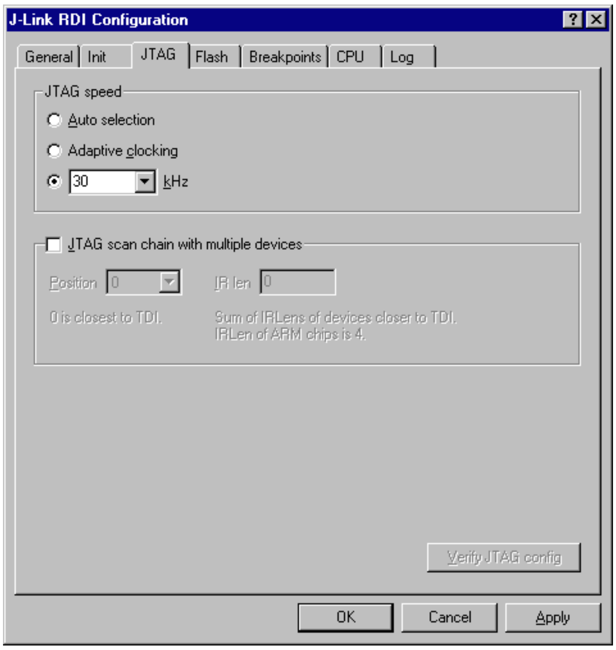
Command	Description
`SetJTAGSpeed(x);`	Sets the JTAG speed, `x` = speed in kHz (0=Auto)
`Delay(x);`	Waits a given time, `x` = delay in milliseconds
`Reset(x);`	Resets the target, `x` = delay in milliseconds
`Go();`	Starts the ARM core
`Halt();`	Halts the ARM core
`Read8(Addr);`	Reads a 8/16/32 bit value,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

Table 8.2: Macro file commands

Example of macro file

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

8.4.4.3 JTAG tab



JTAG speed

○ Auto selection

○ Adaptive clocking

● 30 kHz

□ JTAG scan chain with multiple devices

Position 0 IR len 0

0 is closest to TDI.

Sum of IRLens of devices closer to TDI.
IRLen of ARM chips is 4.

JTAG speed

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

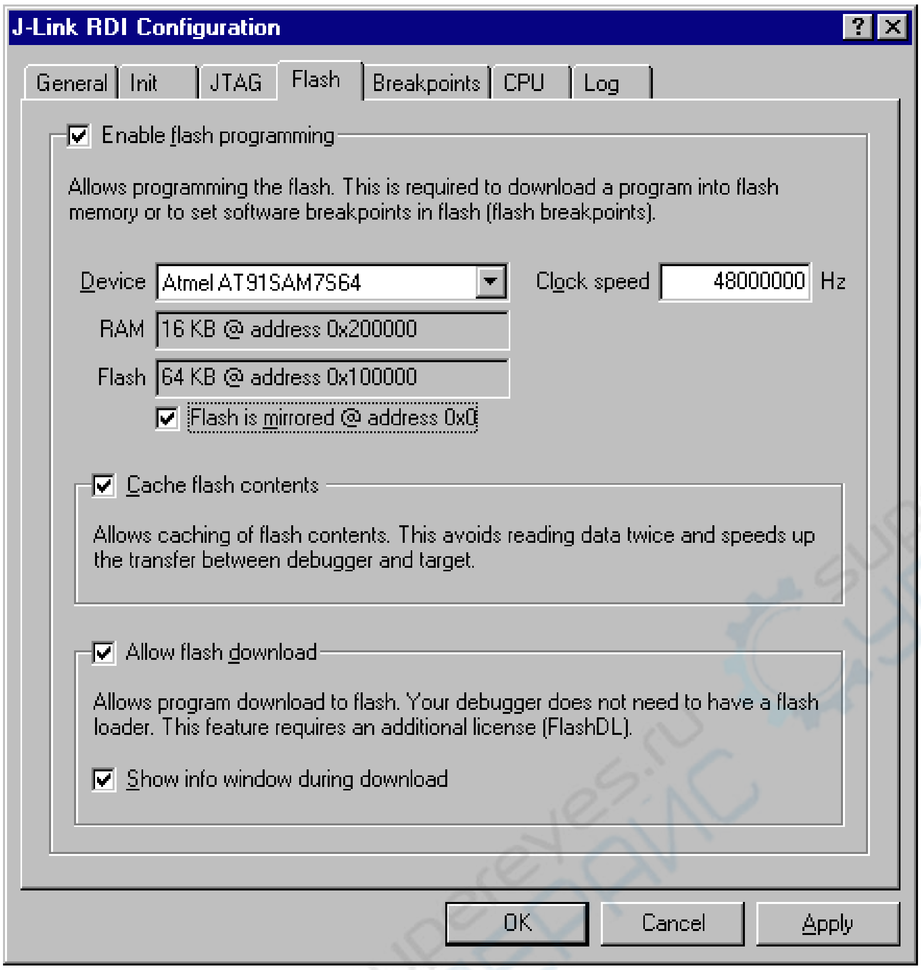
- Fixed JTAG speed
- Automatic JTAG speed
- Adaptive clocking

For more information about the different speed settings supported by J-Link, please refer to *JTAG Speed* on page 148.

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 at once.

8.4.4.4 Flash tab



J-Link RDI Configuration

☑ Enable flash programming

Allows programming the flash. This is required to download a program into flash memory or to set software breakpoints in flash (flash breakpoints).

Device: Atmel AT91SAM7S64

Clock speed: 48000000 Hz

RAM: 16 KB @ address 0x200000

Flash: 64 KB @ address 0x100000

☑ Flash is mirrored @ address 0x0

☑ Cache flash contents

Allows caching of flash contents. This avoids reading data twice and speeds up the transfer between debugger and target.

☑ Allow flash download

Allows program download to flash. Your debugger does not need to have a flash loader. This feature requires an additional license (FlashDL).

☑ Show info window during download

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.

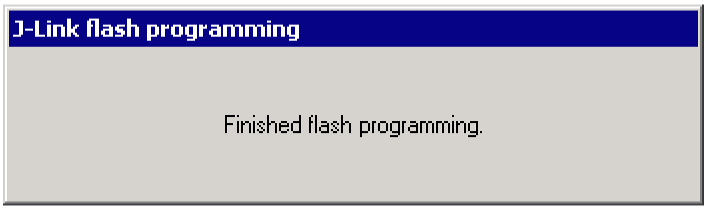
Cache flash contents

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.

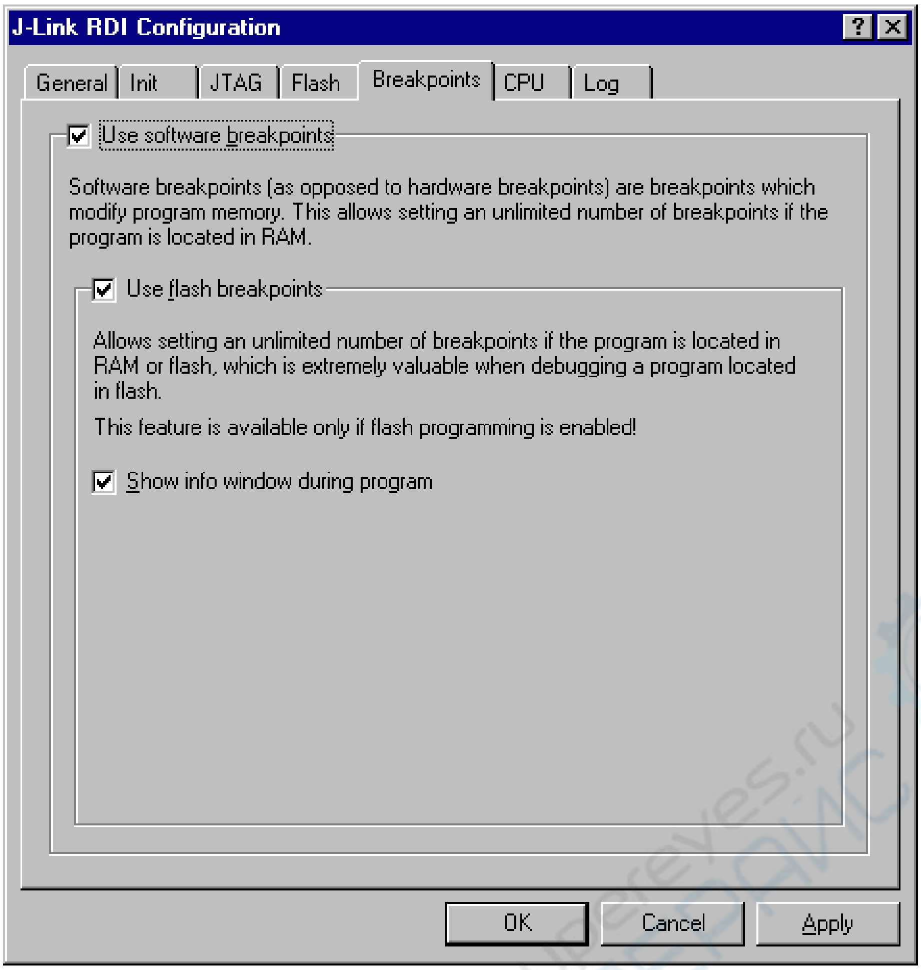
An info window can be shown during download displaying the current operation. Depending on your JTAG speed you may see the info window only very short.



J-Link flash programming

Finished flash programming.

8.4.4.5 Breakpoints tab



☑ 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

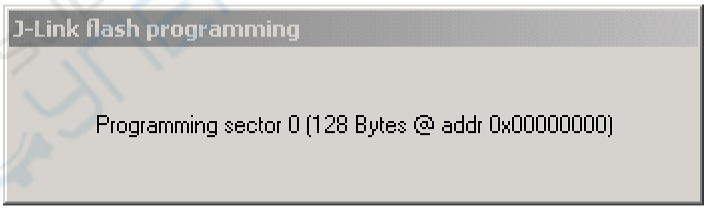
Use software breakpoints

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

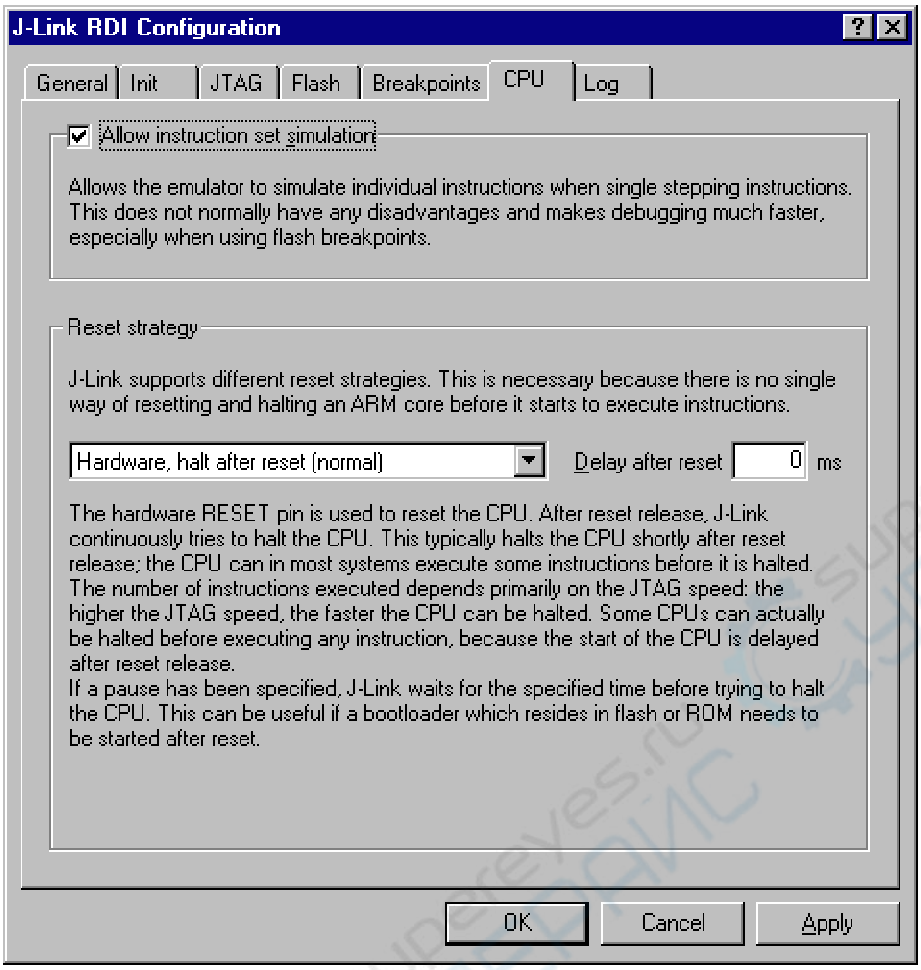
This allows to set an unlimited number of breakpoints if the program is located either in RAM or in flash by setting and resetting breakpoints according to program code.

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.



Programming sector 0 (128 Bytes @ addr 0x00000000)

8.4.4.6 CPU tab



☑ Allow 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)   Delay 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.

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 163.

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

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 241)

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

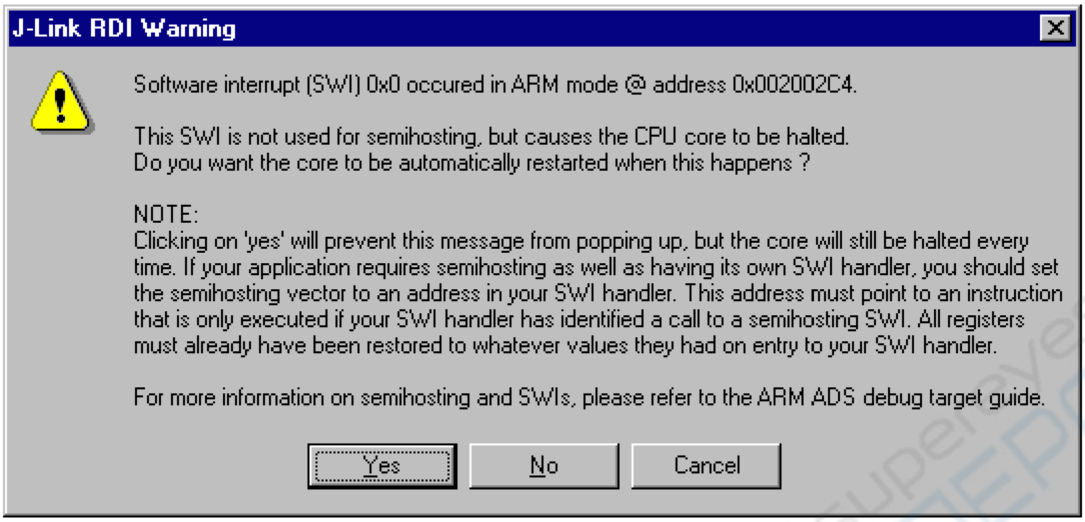
This variable controls the location of the breakpoint set by J-Link RDI to detect a semihosted SWI. It is set to the SWI entry in the exception vector table () by default.

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.



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.

Chapter 9

RTT

SEGGER’s Real Time Terminal (RTT) is a technology for interactive user I/O in embedded applications. It combines the advantages of SWO and semihosting at very high performance.

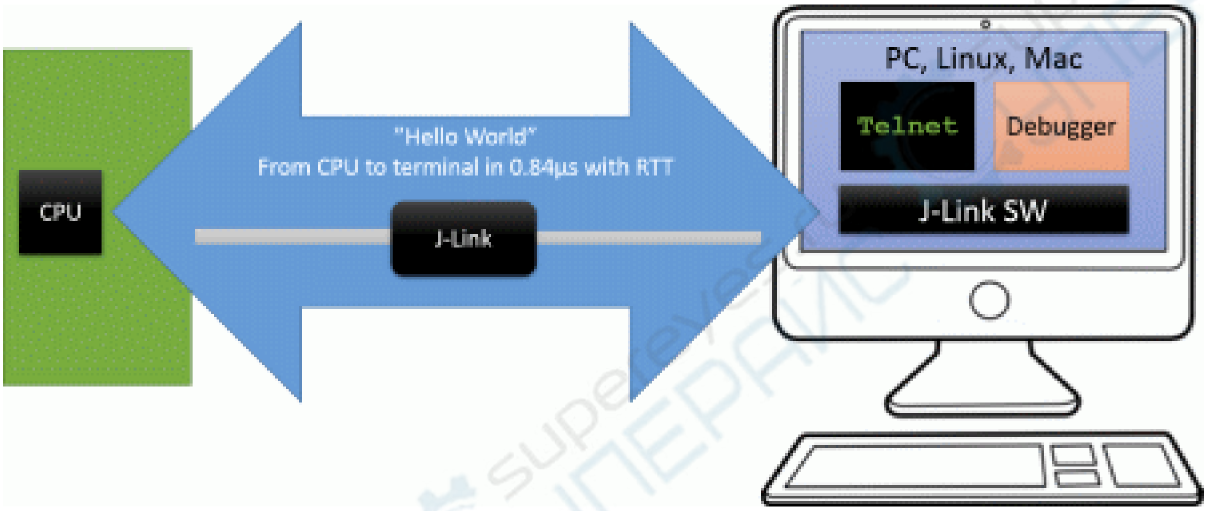
9.1 Introduction

With RTT it is possible to output information from the target microcontroller as well as sending input to the application at a very high speed without affecting the target's real time behavior.

SEGGER RTT can be used with any J-Link model and any supported target processor which allows background memory access, which are Cortex-M and RX targets.

RTT supports multiple channels in both directions, up to the host and down to the target, which can be used for different purposes and provide the most possible freedom to the user.

The default implementation uses one channel per direction, which are meant for printable terminal input and output. With the J-Link RTT Viewer this channel can be used for multiple "virtual" terminals, allowing to print to multiple windows (e.g. one for standard output, one for error output, one for debugging output) with just one target buffer. An additional up (to host) channel can for example be used to send profiling or event tracing data.



9.2 How RTT works

9.2.1 Target implementation

Real Time Terminal uses a SEGGER RTT Control Block structure in the target's memory to manage data reads and writes.

The control block contains an ID to make it findable in memory by a connected J-Link and a ring buffer structure for each available channel, describing the channel buffer and its state.

The maximum number of available channels can be configured at compile time and each buffer can be configured and added by the application at run time. Up and down buffers can be handled separately.

Each channel can be configured to be blocking or non-blocking. In blocking mode the application will wait when the buffer is full, until all memory could be written, resulting in a blocked application state but preventing data from getting lost. In non-blocking mode only data which fits into the buffer, or none at all, will be written and the rest will be discarded. This allows running in real-time, even when no debugger is connected. The developer does not have to create a special debug version and the code can stay in place in a release application.

9.2.2 Locating the Control Block

When RTT is active on the host computer, either by using RTT directly via an application like RTT Viewer or by connecting via Telnet to an application which is using J-Link, like a debugger, J-Link automatically searches for the SEGGER RTT Control Block in the target's known RAM regions. The RAM regions or the specific address of the Control Block can also be set via the host applications to speed up detection or it the block cannot be found automatically.

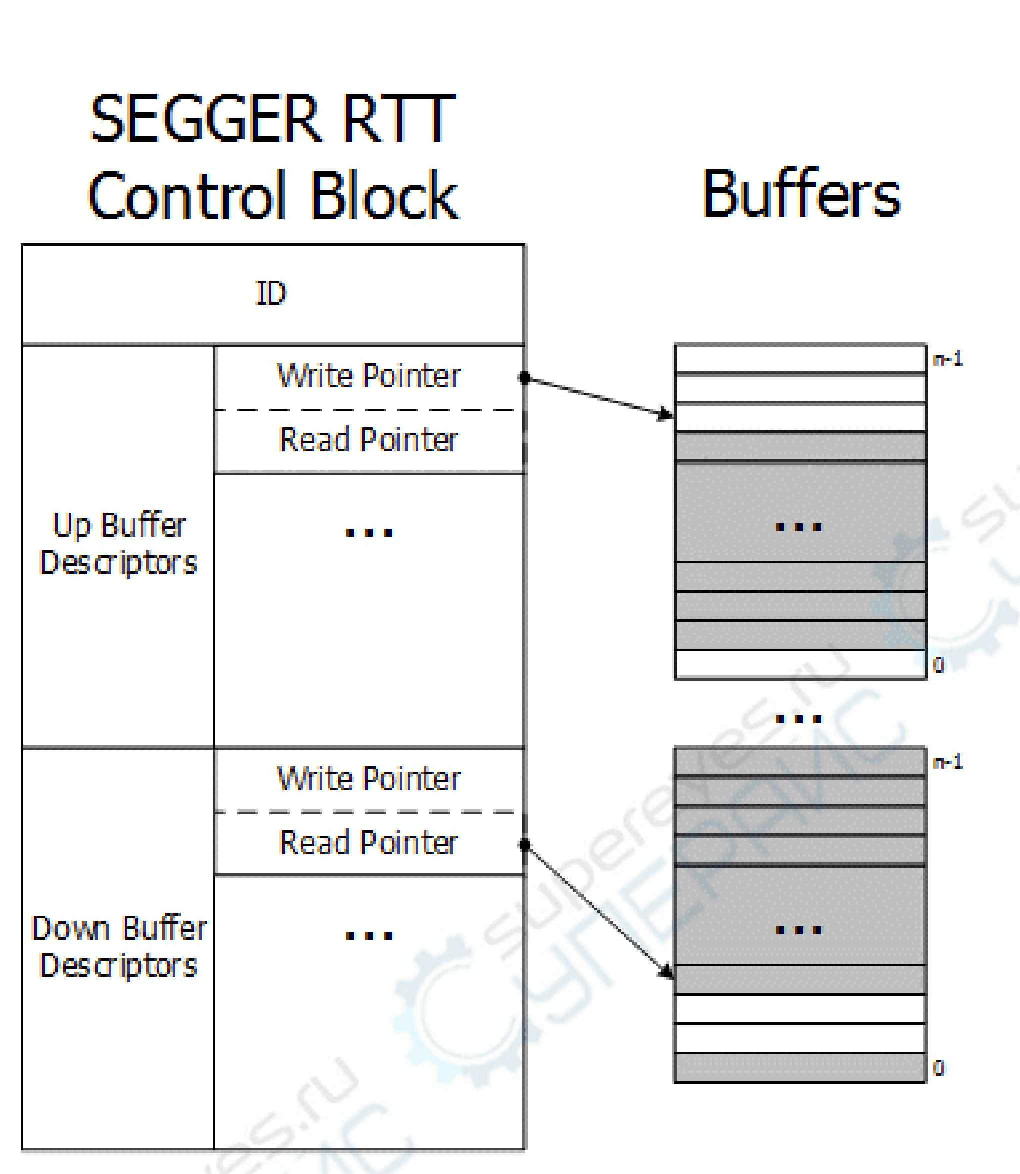
9.2.3 Internal structures

There may be any number of "Up Buffer Descriptors" (Target -> Host), as well as any number of "Down Buffer Descriptors" (Host -> Target). Each buffer size can be configured individually.

The gray areas in the buffers are the areas that contain valid data.

For Up buffers, the Write Pointer is written by the target, the Read Pointer is written by the debug probe (J-Link, Host).

When Read and Write Pointers point to the same element, the buffer is empty. This assures there is never a race condition. The image shows the simplified structure in the target.



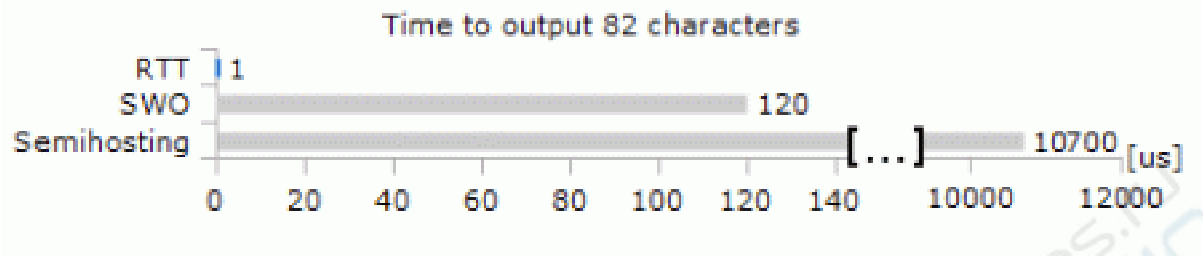
9.2.4 Requirements

SEGGER RTT dies not need any additional pin or hardware, despite a J-Link connected via the standard debug port to the target. It does not require any configuration of the target or in the debugging environment and can even be used with varying target speeds.

RTT can be used in parallel to a running debug session, without intrusion, as well as without any IDE or debugger at all.

9.2.5 Performance

The performance of SEGGER RTT is significantly higher than any other technology used to output data to a host PC. An average line of text can be output in one microsecond or less. Basically only the time to do a single memcopy().



9.2.6 Memory footprint

The RTT implementation code uses ~500 Bytes of ROM and 24 Bytes ID + 24 Bytes per channel for the control block in RAM. Each channel requires some memory for the buffer. The recommended sizes are 1 kByte for up channels and 16 to 32 Bytes for down channels depending on the load of in- / output.

9.3 RTT Communication

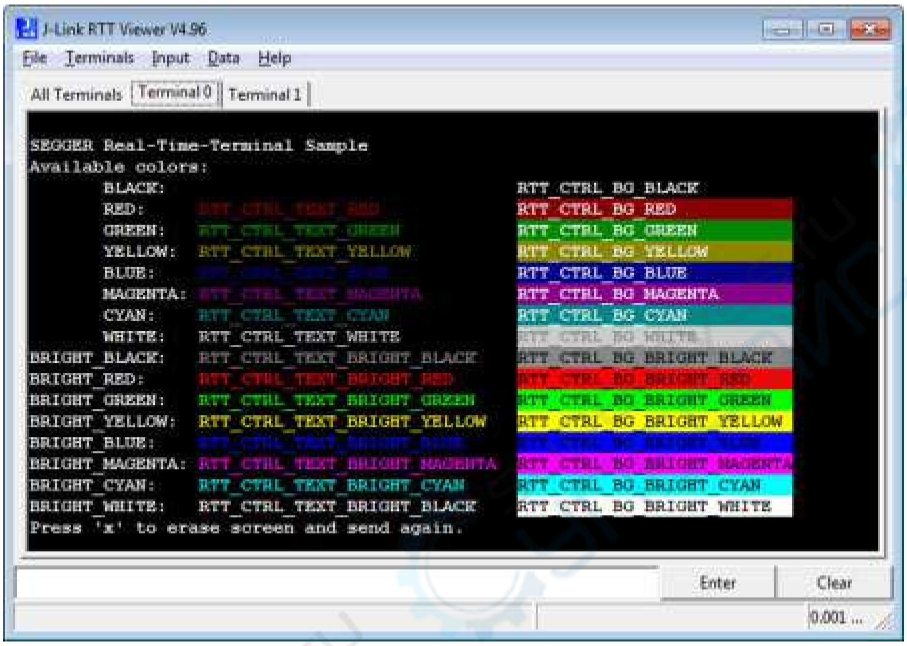
Communication with the RTT implementation on the target can be done with different applications. The functionality can even be integrated into custom applications using the J-Link SDK.

Using RTT in the target application is made easy. The implementation code is freely available for download and can be integrated into any existing application. To communicate via RTT any J-Link can be used.

The simple way to communicate via the Terminal (Channel 0) is to create a connection to localhost:19021 with a Telnet client or similar, when a connection to J-Link (e.g. via a debug session) is active.

The J-Link Software Package comes with some more advanced applications for different purposes.

9.3.1 J-Link RTT Viewer



```
SEGGER Real-Time-Terminal Sample
Available colors:
        BLACK:                                 RTT_CTRL_BG_BLACK
        RED:       RTT_CTRL_TEXT_RED           RTT_CTRL_BG_RED
        GREEN:     RTT_CTRL_TEXT_GREEN         RTT_CTRL_BG_GREEN
        YELLOW:    RTT_CTRL_TEXT_YELLOW        RTT_CTRL_BG_YELLOW
        BLUE:      RTT_CTRL_TEXT_BLUE          RTT_CTRL_BG_BLUE
        MAGENTA:   RTT_CTRL_TEXT_MAGENTA       RTT_CTRL_BG_MAGENTA
        CYAN:      RTT_CTRL_TEXT_CYAN          RTT_CTRL_BG_CYAN
        WHITE:     RTT_CTRL_TEXT_WHITE         RTT_CTRL_BG_WHITE
BRIGHT_BLACK:      RTT_CTRL_TEXT_BRIGHT_BLACK  RTT_CTRL_BG_BRIGHT_BLACK
BRIGHT_RED:        RTT_CTRL_TEXT_BRIGHT_RED    RTT_CTRL_BG_BRIGHT_RED
BRIGHT_GREEN:      RTT_CTRL_TEXT_BRIGHT_GREEN  RTT_CTRL_BG_BRIGHT_GREEN
BRIGHT_YELLOW:     RTT_CTRL_TEXT_BRIGHT_YELLOW RTT_CTRL_BG_BRIGHT_YELLOW
BRIGHT_BLUE:       RTT_CTRL_TEXT_BRIGHT_BLUE   RTT_CTRL_BG_BRIGHT_BLUE
BRIGHT_MAGENTA:    RTT_CTRL_TEXT_BRIGHT_MAGENTA RTT_CTRL_BG_BRIGHT_MAGENTA
BRIGHT_CYAN:       RTT_CTRL_TEXT_BRIGHT_CYAN   RTT_CTRL_BG_BRIGHT_CYAN
BRIGHT_WHITE:      RTT_CTRL_TEXT_BRIGHT_BLACK  RTT_CTRL_BG_BRIGHT_WHITE
Press 'x' to erase screen and send again.
```

J-Link RTT Viewer is a Windows GUI application to use all features of RTT in one application. It supports:

- Displaying terminal output of Channel 0.
- Up to 16 virtual Terminals on Channel 0.
- Sending text input to Channel 0.
- Interpreting text control codes for colored text and controling the Terminal.
- Logging data on Channel 1.

9.3.1.1 RTT Viewer Startup

Make sure J-Link and target device are connected and powered up.

Start RTT Viewer by opening the executable (JLinkRTTViewer.exe) from the installation folder of the J-Link Software or the start menu.

The Configuration Dialog will pop up.

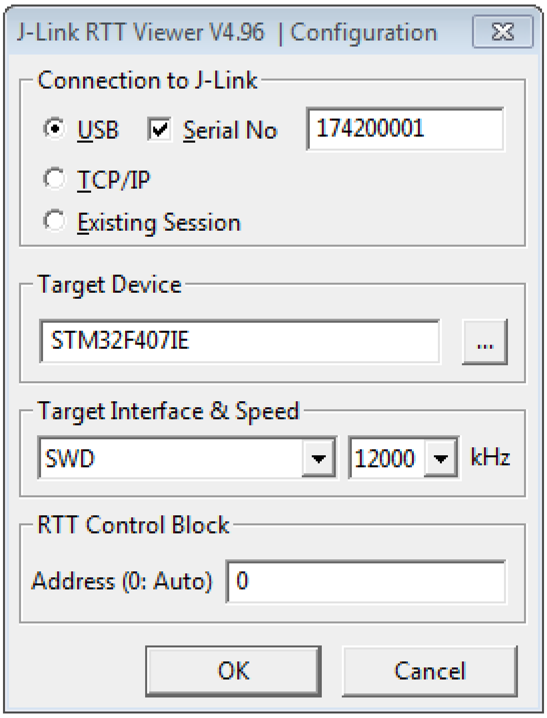
Configure the Connection Settings as described below and click OK. The connection settings and all in app configuration will be saved for the next start of RTT Viewer.

9.3.1.2 Connection Settings

RTT Viewer can be used in two modes:

- Stand-alone, opening an own connection to J-Link and target

- In attach mode, connecting to an existing J-Link connection of a debugger.



J-Link RTT Viewer V4.96 | Configuration

Connection to J-Link

● USB ☑ Serial No 174200001

○ TCP/IP

○ Existing Session

Target Device

STM32F407IE

Target Interface & Speed

SWD 12000 kHz

RTT Control Block

Address (0: Auto) 0

Stand-alone connection settings

In stand-alone mode RTT Viewer needs to know some settings of J-Link and target device.

Select USB or TCP/IP as the connection to J-Link. For USB a specific J-Link serial number can optionally be entered, for TCP/IP the IP or hostname of the J-Link has to be entered.

Select the target device to connect to. This allows J-Link to search in the known RAM of the target.

Select the target interface and its speed.

If known, enter the address of the RTT Control Block in the target application. Otherwise leave as 0 for auto detection.

Attaching to a connection

In attach mode RTT Viewer does not need any settings. Select Existing Session.

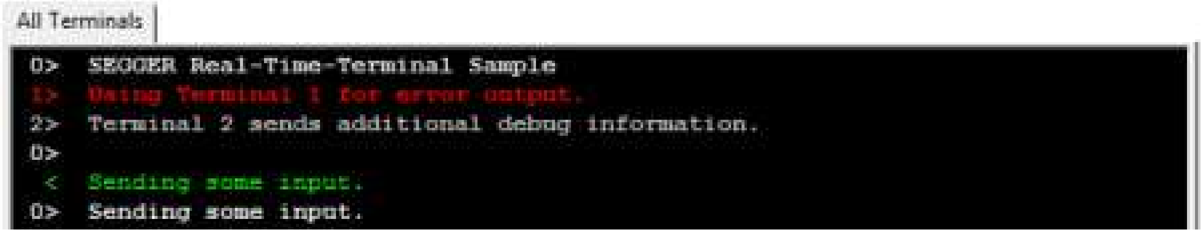
For attach mode a connection to J-Link has to be opened and configured by another application like a debugger or simply J-Link Commander. If the RTT Control Block cannot be found automatically, configuration of its location has to be done by the debugger / application.

9.3.1.3 The Terminal Tabs

RTT Viewer allows displaying the output of Channel 0 in different "virtual" Terminals.

The target application can switch between terminals with SEGGER_RTT_SetTerminal() and SEGGER_RTT_TerminalOut().

RTT Viewer displays the Terminals in different tabs.



```
0> SEGGER Real-Time-Terminal Sample
1> Using Terminal 1 for error output.
2> Terminal 2 sends additional debug information.
0>
 < Sending some input.
0> Sending some input.
```

All Terminals

The All Terminals tab displays the complete output of RTT Channel 0 and can display the user input (Check Input -> Echo input... -> Echo to "All Terminals").

Each output line is prefixed by the Terminal it has been sent to. Additionally output on Terminal 1 is shown in red, output on Terminals 2 - 15 in grey.

Terminal 0 - 15

Each tab Terminal 0 - Terminal 15 displays the output which has been sent to this Terminal. The Terminal tabs interpret and display Text Control Codes as sent by the application to show colored text or erase the screen.

By default, if the RTT application does not set a Terminal Id, the output is displayed in Terminal 0.

The Teminal 0 tab can additionally display the user input. (Check Input -> Echo input... -> Echo to "Terminal 0")

Each Terminal tab can be shown or hidden via the menu Terminals -> Terminals... or their respective shortcuts as described below.

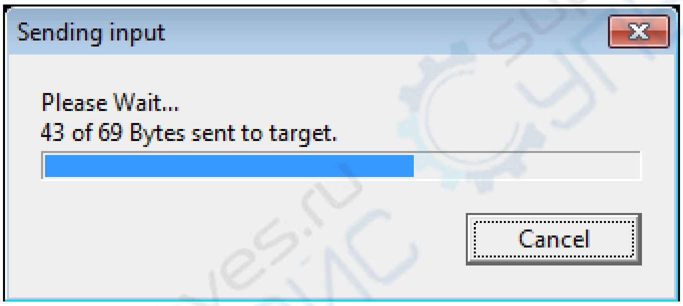
9.3.1.4 Sending Input

RTT Viewer supports sending user input to RTT Down Channel 0 which can be read by the target application with SEGGER_RTT_GetKey() and SEGGER_RTT_Read().

Input can be entered in the text box below the Teminal Tabs.

RTT Viewer can be configured to directly send each character while typing or buffer it until Enter is pressed (Menu Input -> Sending...).

In stand-alone mode RTT Viewer can retry to send input, in case the target input buffer is full, until all data could be sent to the target via Input -> Sending... -> Block if FIFO full.



Sending input

Please Wait...
43 of 69 Bytes sent to target.

9.3.1.5 Logging Data

Additionally to displaying output of Channel 0, RTT Viewer can log data which is sent on RTT Channel 1 into a file. This can for example be used to sent instrumentalized event tracing data. The data log file contains header and footer and the binary data as received from the application.

Logging can be started via Data -> Start Logging....

Note: Logging is only available in stand-alone mode.

9.3.1.6 Menus and Shortcuts

Menu entry	Contents	Shortcut
File		
-> Exit	Close connection and exit RTT Viewer.	Alt-Q
Terminals		
-> Add next terminal	Opens the next available Terminal Tab.	Alt-A
-> Close active terminal	Closes the active Terminal Tab.	Alt-C
-> Show Log	Opens or closes the Log Tab.	Alt-L
Terminals -> Terminals...		
--> Terminal 0 - 9	Opens or closes the Terminal Tab.	Alt-0 - Alt-9
--> Terminal 10 - 15	Opens or closes the Terminal Tab.	
Input		
-> Clear input field	Clears the input field without sending entered data.	Button 'Clear'
Input -> Sending...		
--> Send on Input	If selected, entered input will be sent directly to the target while typing.	
--> Send on Enter	If selected, entered input will be sent when pressing Enter.	
--> Block if FIFO full	If checked, RTT Viewer will retry to send all input to the target when the target buffer is full.	
Input -> End of line...		
--> Windows format (CR+LF) **--> Unix format (LF)** **--> Mac format (CR)** **--> None**	Select the end of line character to be sent on Enter.	
Input -> Echo input...		
--> Echo to "All Terminals"	If checked, sent input will be displayed in the All Terminals Tab.	
--> Echo to "Terminal 0"	If checked, sent input will be displayed in the Terminal Tab 0.	
Data		
-> Start logging...	Start logging data of Channel 1 to a file.	F5
-> Stop logging	Stop loggind data and close the file.	Shift-F5
Help		
-> About...	Show version info of RTT Viewer.	F12

Table 9.1: RTT Viewer Menus and Shortcuts

Menu entry	Contents	Shortcut
-> J-Link Manual...	Open the J-Link Manual PDF file.	F11
-> RTT Webpage...	Open the RTT webpage.	F10
Right-Click on Tab		
-> Clear Terminal	Clear the displayed output of this Terminal Tab.	

Table 9.1: RTT Viewer Menus and Shortcuts

9.3.1.7 Using "virtual" Terminals in RTT

For virtual Terminals the target application needs only Up Channel 0. This is especially important on targets with low RAM.

If nothing is configured, all data is sent to Terminal 0.

The Teminal to output all following via Write, WriteString or printf can be set with SEGGER_RTT_SetTerminal() .

Output of only one string via a specific Terminal can be done with SEGGER_RTT_TerminalOut().

The sequences sent to change the Terminal are interpreted by RTT Viewer. Other applications like a Telnet Client will ignore them.

9.3.1.8 Using Text Control Codes

RTT allows using Text Control Codes (ANSI escape codes) to configure the display of text.

RTT Viewer supports changing the text color and background color and can erase the Terminal.

These Control Codes are pre-defined in the RTT application and can easily be used in the application.

Example 1:

```
SEGGER_RTT_WriteString(0,
                       RTT_CTRL_RESET"Red: "                                      \
                       RTT_CTRL_TEXT_BRIGHT_RED"This text is red. "               \
                       RTT_CTRL_TEXT_BLACK""                                      \
                       RTT_CTRL_BG_BRIGHT_RED"This background is red. "           \
                       RTT_CTRL_RESET"Normal text again.");
```

Example 2:

```
SEGGER_RTT_printf(0, "%sTime:%s%s %.7d\n",
                      RTT_CTRL_RESET,
                      RTT_CTRL_BG_BRIGHT_RED,
                      RTT_CTRL_TEXT_BRIGHT_WHITE,
                      1111111
                      );
//
// Clear the terminal.
// The first line will not be shown after this command.
//
SEGGER_RTT_WriteString(0, RTT_CTRL_CLEAR);

SEGGER_RTT_printf(0, "%sTime: %s%s%.7d\n",
                      RTT_CTRL_RESET,
                      RTT_CTRL_BG_BRIGHT_RED,
                      RTT_CTRL_TEXT_BRIGHT_WHITE,
                      2222222
                      );
```

9.3.2 RTT Client

J-Link RTT Client acts as a Telnet client, but automatically tries to reconnect to a J-Link connection when a debug session is closed.

The J-Link RTT Client is part of the J-Link Software and Documentation Pack for Windows, Linux and OS X and can be used for simple RTT use cases.

9.3.3 RTT Logger

With J-Link RTT Logger, data from Up-Channel 1 can be read and logged to a file. This channel can for example be used to send performance analysis data to the host.

J-Link RTT Logger opens a dedicated connection to J-Link and can be used stand-alone, without running a debugger.

The application is part of the J-Link Software and Documentation Pack for Windows, Linux and OS X..

The source of J-Link RTT Logger can be used as a starting point to integrate RTT in other PC applications, like debuggers and is part of the J-Link SDK.

9.3.4 RTT in other host applications

RTT can also be integrated in any other PC application like a debugger or a data visualizer in either of two ways.

1. The application can establish a socket connection to the RTT Telnet Server which is opened on localhost:19021 when a J-Link connection is active.
2. The application creates its own connection to J-Link and uses the J-Link RTT API which is part of the J-Link SDK to directly configure and use RTT.

9.4 Implementation

The SEGGER RTT implementation code is written in ANSI C and can be integrated into any embedded application by simply adding the available sources.

RTT can be used via a simple and easy to use API. It is even possible to override the standard printf() functions to be used with RTT. Using RTT reduces the time taken for output to a minimum and allows printing debug information to the host computer while the application is performing time critical real time tasks.

The implementation code also includes a simple version of printf() which can be used to write formatted strings via RTT. It is smaller than most standard library printf() implementations and does not require heap and only a configureable ammount of stack.

The SEGGER RTT implementation is fully configureable at compile time with pre-processor defines. The number of channels, the size of the default channels can be set. Reading and writing can be made task-safe with definable Lock() and Unlock() routines.

9.4.1 API functions

The following API functions are available in the RTT Implementation. To use them SEGGER_RTT.h has to be included in the calling sources.

9.4.1.1 SEGGER_RTT_ConfigDownBuffer()

Description

Configure or add a down buffer by specifying its name, size and flags.

Prototype

```
int SEGGER_RTT_ConfigDownBuffer (unsigned BufferIndex, const char* sName,
char* pBuffer, int BufferSize, int Flags);
```

Parameters

Parameter	Meaning
BufferIndex	Index of the buffer to configure. Must be lower than SEGGER_RTT_MAX_NUM_DOWN_CHANNELS.
sName	Pointer to a 0-terminated string to be displayed as the name of the channel.
pBuffer	Pointer to a buffer used by the channel.
BufferSize	Size of the buffer in Bytes.
Flags	Flags of the channel (blocking or non-blocking).

Table 9.2: SEGGER_RTT_ConfigDownBuffer() parameter list

Return value

>= 0	O.K.
< 0	Error.

Example

```
//
// Configure down channel 1
//
SEGGER_RTT_ConfigDownChannel(1, "DataIn", &abDataIn[0], sizeof(abDataIn),
                                          SEGGER_RTT_MODE_NO_BLOCK_SKIP);
```

Additional information

Once a channel is configured only the flags of the channel should be changed.

9.4.1.2 SEGGER_RTT_ConfigUpBuffer()

Description

Configure or add an up buffer by specifying its name, size and flags.

Prototype

```
int SEGGER_RTT_ConfigUpBuffer (unsigned BufferIndex, const char* sName,
char* pBuffer, int BufferSize, int Flags);
```

Parameters

Parameter	Meaning
BufferIndex	Index of the buffer to configure. Must be lower than SEGGER_RTT_MAX_NUM_UP_CHANNELS.
sName	Pointer to a 0-terminated string to be displayed as the name of the channel.
pBuffer	Pointer to a buffer used by the channel.
BufferSize	Size of the buffer in Bytes.
Flags	Flags of the channel (blocking or non-blocking).

Table 9.3: SEGGER_RTT_ConfigUpBuffer() parameter list

Return value

>= 0	O.K.
< 0	Error.

Example

```
//
// Configure up channel 1 to work in blocking mode
//
SEGGER_RTT_ConfigUpChannel(1, "DataOut", &abDataOut[0], sizeof(abDataOut),
                           SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL);
```

Additional information

Once a channel is configured only the flags of the channel should be changed.

9.4.1.3 SEGGER_RTT_GetKey()

Description

Reads one character from SEGGER RTT buffer 0. Host has previously stored data there.

Prototype

```
int SEGGER_RTT_GetKey (void);
```

Return value

< 0	No character available (empty buffer).
>= 0	Character which has been read (0 - 255).

Example

```
int c;
c = SEGGER_RTT_GetKey();
if (c == ’q’) {
  exit();
}
```

9.4.1.4 SEGGER_RTT_HasKey()

Description

Checks if at least one character for reading is available in SEGGER RTT buffer. 0

Prototype

```
int SEGGER_RTT_HasKey (void);
```

Return value

0	No characters are available to be read.
1	At least one character is available in the buffer.

Example

```
if (SEGGER_RTT_HasKey()) {
  int c = SEGGER_RTT_GetKey();
}
```

9.4.1.5 SEGGER_RTT_Init()

Description

Initializes the RTT Control Block.

Prototype

```
void SEGGER_RTT_Init (void);
```

Additional information

Should be used in RAM targets, at start of the application.

9.4.1.6 SEGGER_RTT_printf()

Description

Send a formatted string to the host.

Prototype

```
int SEGGER_RTT_printf (unsigned BufferIndex, const char * sFormat, ...)
```

Parameters

Parameter	Meaning
BufferIndex	Index of the up channel to sent the string to.
sFormat	Pointer to format string, followed by arguments for conversion.

Table 9.4: SEGGER_RTT_printf() parameter list

Return value

>= 0	Number of bytes which have been sent.
< 0	Error.

Example

```
SEGGER_RTT_printf(0, "SEGGER RTT Sample. Uptime: %.10dms.", /*OS_Time*/ 890912);
// Formatted output on channel 0: SEGGER RTT Sample. Uptime: 890912ms.
```

Additional information

(1) Conversion specifications have following syntax:

- %[flags][FieldWidth][.Precision]ConversionSpecifier

(2) Supported flags:

- -: Left justify within the field width
- +: Always print sign extension for signed conversions
- 0: Pad with 0 instead of spaces. Ignored when using '-'-flag or precision

(3) Supported conversion specifiers:

- c: Print the argument as one char
- d: Print the argument as a signed integer
- u: Print the argument as an unsigned integer
- x: Print the argument as an hexadecimal integer
- s: Print the string pointed to by the argument
- p: Print the argument as an 8-digit hexadecimal integer. (Argument shall be a pointer to void.)

9.4.1.7 SEGGER_RTT_Read()

Description

Read characters from any RTT down channel which have been previously stored by the host.

Prototype

```
int SEGGER_RTT_Read (unsigned BufferIndex, char* pBuffer, unsigned
BufferSize);
```

Parameters

Parameter	Meaning
BufferIndex	Index of the down channel to read from.
pBuffer	Pointer to a character buffer to store the read characters.
BufferSize	Number of bytes available in the buffer.

Table 9.5: SEGGER_RTT_Read() parameter list

Return value

>= 0	Number of bytes that have been read.

Example

```
char acIn[4];
int  NumBytes = sizeof(acIn);
NumBytes = SEGGER_RTT_Read(0, &acIn[0], NumBytes);
if (NumBytes) {
  AnalyzeInput(acIn);
}
```

9.4.1.8 SEGGER_RTT_SetTerminal()

Description

Set the "virtual" terminal to send following data on channel 0 to.

Prototype

```
void SEGGER_RTT_SetTerminal(char TerminalId);
```

Parameters

Parameter	Meaning
TerminalId	Id of the virtual terminal (0-9).

Table 9.6: SEGGER_RTT_SetTerminal() parameter list

Example

```
//
// Send a string to terminal 1 which is used as error out.
//
SEGGER_RTT_SetTerminal(1);  // Select terminal 1
SEGGER_RTT_WriteString(0, "ERROR: Buffer overflow");
SEGGER_RTT_SetTerminal(0);  // Reset to standard terminal
```

Additional information

All following data which is sent via channel 0 will be printed on the set terminal until the next change.

9.4.1.9 SEGGER_RTT_TerminalOut()

Description

Sent one string to a specific "virtual" terminal

Prototype

```
int SEGGER_RTT_TerminalOut (char TerminalID, const char* s);
```

Parameters

Parameter	Meaning
TerminalId	Id of the virtual terminal (0-9).
s	Pointer to 0-terminated string to be sent.

Table 9.7: SEGGER_RTT_TerminalOut() parameter list

Return value

>= 0	Number of bytes sent to the terminal.

Example

```
//
// Sent a string to terminal 1 without changing the standard terminal.
//
SEGGER_RTT_TerminalOut(1, "ERROR: Buffer overflow.");
```

Additional information

SEGGER_RTT_TerminalOut does not affect following data which is sent via channel 0.

9.4.1.10 SEGGER_RTT_Write()

Description

Send data to the host on an RTT channel.

Prototype

```
int SEGGER_RTT_Write (unsigned BufferIndex, const char* pBuffer, unsigned
NumBytes);
```

Parameters

Parameter	Meaning
BufferIndex	Index of the up channel to send data to.
pBuffer	Pointer to data to be sent.
NumBytes	Number of bytes to send.

Table 9.8: SEGGER_RTT_Write() parameter list

Return value

>= 0	Number of bytes which have been sent.

Additional information

With SEGGER_RTT_Write() all kinds of data, not only printable one can be sent.

9.4.1.11 SEGGER_RTT_WaitKey()

Description

Waits until at least one character is avaible in SEGGER RTT buffer 0. Once a character is available, it is read and returned.

Prototype

```
int SEGGER_RTT_WaitKey (void);
```

Return value

>= 0	Character which has been read (0 - 255).

Example

```
int c = 0;
do {
  c = SEGGER_RTT_WaitKey();
} while (c != ’c’);
```

9.4.1.12 SEGGER_RTT_WriteString()

Description

Write a 0-terminated string to an up channel via RTT.

Prototype

```
int SEGGER_RTT_WriteSting (unsigned BufferIndex, const char* s);
```

Parameters

Parameter	Meaning
BufferIndex	Index of the up channel to send string to.
s	Pointer to 0-terminated string to be sent.

Table 9.9: SEGGER_RTT_WriteString() parameter list

Return value

>= 0	Number of bytes which have been sent.

Example

```
SEGGER_RTT_WriteString(0, "Hello World from your target.\n");
```

9.4.2 Configuration defines

9.4.2.1 RTT configuration

SEGGER_RTT_MAX_NUM_DOWN_BUFFERS

Maximum number of down (to target) channels.

SEGGER_RTT_MAX_NUM_UP_BUFFERS

Maximum number of up (to host) cahnnels.

BUFFER_SIZE_DOWN

Size of the buffer for default down channel 0.

BUFFER_SIZE_UP

Size of the buffer for default up channel 0.

SEGGER_RTT_PRINT_BUFFER_SIZE

Size of the buffer for SEGGER_RTT_printf to bulk-send chars.

SEGGER_RTT_LOCK()

Locking routine to prevent interrupts and task switches from within an RTT operation.

SEGGER_RTT_UNLOCK()

Unlocking routine to allow interrupts and task switches after an RTT operation.

SEGGER_RTT_IN_RAM

Indicate the whole application is in RAM to prevent falsly identifying the RTT Control Block in the init segment by defining as 1.

9.4.2.2 Channel buffer configuration

SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL

SEGGER_RTT_NO_BLOCK_SKIP

SEGGER_RTT_NO_BLOCK_TRIM

9.4.2.3 Color control sequences

RTT_CTRL_RESET

Reset the text color and background color.

RTT_CTRL_TEXT_*

Set the text color to one of the following colors.

- BLACK,
- RED,
- GREEN,
- YELLOW,
- BLUE,
- MAGENTA,
- CYAN,
- WHITE (light grey),
- BRIGHT_BLACK (dark grey),
- BRIGHT_RED,

- BRIGHT_GREEN,
- BRIGHT_YELLOW,
- BRIGHT_BLUE,
- BRIGHT_MAGENTA,
- BRIGHT_CYAN,
- BRIGHT_WHITE

RTT_CTRL_BG_*

Set the background color to one of the following colors.

- BLACK,
- RED,
- GREEN,
- YELLOW,
- BLUE,
- MAGENTA,
- CYAN,
- WHITE (light grey),
- BRIGHT_BLACK (dark grey),
- BRIGHT_RED,
- BRIGHT_GREEN,
- BRIGHT_YELLOW,
- BRIGHT_BLUE,
- BRIGHT_MAGENTA,
- BRIGHT_CYAN,
- BRIGHT_WHITE

9.5 Example code

```
/*********************************************************************
*                SEGGER MICROCONTROLLER GmbH & Co KG                 *
*       Solutions for real time microcontroller applications         *
**********************************************************************
*                                                                    *
*       (c) 2014  SEGGER Microcontroller GmbH & Co KG                *
*                                                                    *
*       www.segger.com     Support: support@segger.com               *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File    : RTT.c
Purpose : Simple implementation for output via RTT.
          It can be used with any IDE.
--------  END-OF-HEADER  ---------------------------------------------
*/

#include "SEGGER_RTT.h"

static void _Delay(int period) {
  int i = 100000*period;
  do { ; } while (i--);
}

int main(void) {
  int Cnt = 0;

  SEGGER_RTT_WriteString(0, "Hello World from SEGGER!\n");
  do {
    SEGGER_RTT_printf("%sCounter: %s%d\n",
                      RTT_CTRL_TEXT_BRIGHT_WHITE,
                      RTT_CTRL_TEXT_BRIGHT_GREEN,
                      Cnt);
    if (Cnt > 100) {
      SEGGER_RTT_TerminalOut(1, RTT_CTRL_TEXT_BRIGHT_RED"Counter overflow!");
      Cnt = 0;
    }
    _Delay(100);
    Cnt++;
  } while (1);
  return 0;
}

/*************************** End of file ****************************/
```

9.6 FAQ

Q: How does J-Link find the RTT buffer?

A: There are two ways: If the debugger (IDE) knows the address of the SEGGER RTT Control Block, it can pass it to J-Link. This is for example done by J-Link Debugger. If another application that is not SEGGER RTT aware is used, then J-Link searches for the ID in the known target RAM during execution of the application in the background. This process normally takes just fractions of a second and does not delay program execution.

Q: I am debugging a RAM-omly application. J-Link finds an RTT buffer, but I get no output. What can I do?

A: In case the init section af an application is stored in RAM, J-Link mit falsely identify the block in the init section instead of the actual one in the data section.
To prevent this, set the define SEGGER_RTT_IN_RAM to 1. Now J-Link will find the correct RTT buffer, but only after calling the first SEGGER_RTT function in the application. A call to SEGGER_RTT_Init() at the beginning of the application is recommended.

Q: Can this also be used on targets that do not have the SWO pin?

A: Yes, the debug interface is used. This can be JTAG or SWD (2pins only!) on most Cortex-M devices, or even the FINE interface on some Renesas devices, just like the Infineon SPD interface (single pin!)

Q: Can this also be used on Cortex-M0 and M0+?

A: Yes.

Q: Some terminal output (printf) Solutions "crash" program execution when executed outside of the debug environment, because they use a Software breakpoint that triggers a hardfault without debugger or halt because SWO is not initialized. That makes it impossible to run a Debug-build in stand-alone mode.
What about SEGGER-RTT?

A: SEGGER-RTT uses non-blocking mode per default, which means it does not halt program execution if no debugger is present and J-Link is not even connected. The application program will continue to work.

Q: I do not see any output, although the use of RTT in my application is correct. What can I do?

A: In some cases J-Link cannot locate the RTT buffer in the known RAM region.
In this case the possible region or the exact address can be set manually via a J-Link exec command:

- Set ranges to be searched for RTT buffer: SetRTTSearchRanges <RangeStart [Hex]> <RangeSize >[, <Range1Start [Hex]> <Range1Size>, ...] (e.g. "SetRTTSearchRanges 0x10000000 0x1000, 0x2000000 0x1000")
- Set address of the RTT buffer: SetRTTAddr <RTTBufferAddress [Hex]> (e.g. "SetRTTAddr 0x20000000")
- Set address of the RTT buffer via J-Link Control Panel -> RTTerminal

Note: J-Link exec commands can be executed in most applications, for example in J-Link Commander via "exec <Command>", in J-Link GDB Server via "monitor exec <Command>" or in IAR EW via "__jlinkExecCommand("<Command>");" from a macro file.

Chapter 10

Device specifics

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

10.1 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
- ADuC7129
- ADuC7229x126
- ADuCRF02
- ADuCRF101

10.1.1 ADuC7xxx

10.1.1.1 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
- Analog ADuC7027x62
- Analog ADuC7030
- Analog ADuC7031
- Analog ADuC7032
- Analog ADuC7033
- Analog ADuC7128
- Analog ADuC7129
- Analog ADuC7229x126

10.2 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

10.2.1 AT91SAM7

10.2.1.1 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 0xfffffd00.

This information is applicable to the following devices:

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

10.2.1.2 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)

10.2.1.3 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 0xFFFFFD00 = 0xA5000004
monitor sleep 10
# Disable watchdog
monitor long 0xFFFFFD44 = 0x00008000
monitor sleep 10
# Initialize PLL
monitor long 0xFFFFFC20 = 0x00000601
monitor sleep 10
monitor long 0xFFFFFC2C = 0x00480a0e
monitor sleep 10
monitor long 0xFFFFFC30 = 0x00000007
monitor sleep 10
monitor long 0xFFFFFF60 = 0x00480100
monitor sleep 100
monitor speed 12000
break main
load
continue
```

IAR Sample

```
/*******************************************************************
*
*       _Init()
*/
_Init() {
  __emulatorSpeed(30000);                             // Set JTAG speed to 30 kHz
  __writeMemory32(0xA5000004,0xFFFFFD00,"Memory");    // Perform peripheral reset
  __sleep(20000);
  __writeMemory32(0x00008000,0xFFFFFD44,"Memory");    // Disable Watchdog
  __sleep(20000);
  __writeMemory32(0x00000601,0xFFFFFC20,"Memory");    // PLL
  __sleep(20000);
  __writeMemory32(0x10191c05,0xFFFFFC2C,"Memory");    // PLL
  __sleep(20000);
  __writeMemory32(0x00000007,0xFFFFFC30,"Memory");    // PLL
  __sleep(20000);
  __writeMemory32(0x002f0100,0xFFFFFF60,"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);                        // Set JTAG speed to 30 kHz
Reset(0, 0);
Write32(0xFFFFFD00, 0xA5000004);         // Perform peripheral reset
Write32(0xFFFFFD44, 0x00008000);         // Disable watchdog
Write32(0xFFFFFC20, 0x00000601);         // Set PLL
Delay(200);
Write32(0xFFFFFC2C, 0x00191C05);         // Set PLL and divider
Delay(200);
Write32(0xFFFFFC30, 0x00000007);         // Select master clock and processor clock
Write32(0xFFFFFF60, 0x00320300);         // Set flash wait states
SetJTAGSpeed(12000);
```

10.2.2 AT91SAM9

10.2.2.1 JTAG settings

We recommend using adaptive clocking.

This information is applicable to the following devices:

- AT91RM9200
- AT91SAM9260
- AT91SAM9261
- AT91SAM9262
- AT91SAM9263

10.3 DSPGroup

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

- DA56KLF

Currently, there are no specifics for these devices.

10.4 Ember

For more information, please refer to *Silicon Labs* on page 289.

10.5 Energy Micro

For more information, please refer to *Silicon Labs* on page 289.

10.6 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
- MK20DN512
- MK20DX128
- MK20DX256
- MK30DN512
- MK30DX128
- MK30DX256
- MK40N512
- MK40X128
- MK40X256
- MK50DN512
- MK50DX256
- MK50DN512
- MK50DX256
- MK51DX256
- MK51DN512
- MK51DX256
- MK51DN512
- MK51DN256
- MK51DN512
- MK52DN512
- MK53DN512
- MK53DX256
- MK60N256
- MK60N512
- MK60X256

10.6.1 Kinetis family

10.6.1.1 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.

10.6.1.2 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. This bug applies to all devices with mask 0M33Z from the 100MHz series.

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 \Samples\JLink\Projects.

10.6.1.3 Data flash support

Some devices of the Kinetis family have an additional area called FlexNVM, which can be configured as data flash. The size of the FlexNVM to be used as data flash is configurable and needs to be configured first, before this area can be used as data flash.

The sample below shows how to configure the FlexNVM area to be used as data flash out of the target application.

For J-Flash there are also projects that are already preconfigured to configure the data flash size of a Kinetis device. The projects can be found at $JLINK_INST_DIR$\Samples\JFlash\ProjectFiles. One of these sample projects is the MK40DX256xxx10_ConfigureDataFlash.jflash.

For more information about how configuration of the data flash works, please refer to the appropriate user manual of the device.

Configure FlexNVM area as data flash

The follwing sample configures the data flash size of Kinetis device. It is created for a MK40DX256xxx10 device. The sequence is almost the same for all Kinetis devices only the lines which configure size of the data flash may be modified. In this sample the data flash is set to max size. EEPROM size is set to 0 bytes.

```
#define FSTAT   (*(volatile unsigned char*)(0x40020000 + 0x00))
#define FCCOB0  (*(volatile unsigned char*)(0x40020000 + 0x07))
#define FCCOB1  (*(volatile unsigned char*)(0x40020000 + 0x06))
#define FCCOB2  (*(volatile unsigned char*)(0x40020000 + 0x05))
#define FCCOB3  (*(volatile unsigned char*)(0x40020000 + 0x04))
#define FCCOB4  (*(volatile unsigned char*)(0x40020000 + 0x0B))
#define FCCOB5  (*(volatile unsigned char*)(0x40020000 + 0x0A))
#define FCCOB6  (*(volatile unsigned char*)(0x40020000 + 0x09))
#define FCCOB7  (*(volatile unsigned char*)(0x40020000 + 0x08))
#define FCCOB8  (*(volatile unsigned char*)(0x40020000 + 0x0F))

void ConfigureDataFlash(void);

*******************************************************************
*
*       ConfigureDataFlash
*
*  Notes
*    Needs to be located in RAM since it performs flash operations
*    which make instruction fetching from flash temporarily not possible.
*/
void ConfigureDataFlash(void) {
  unsigned char v;

  //
  // Read out current configuration first
  //
  FSTAT  = 0x70;        // Clear error flags in status register
  FCCOB0 = 0x03;        // Read resource
  FCCOB1 = 0x80;        // Read from data flash IFR area with offset 0xFC (0x8000FC)
  FCCOB2 = 0x00;
  FCCOB3 = 0xFC;
  FCCOB8 = 0x00;               // Select IFR area to be read
  FSTAT  = 0x80;               // Start command execution
  while((FSTAT & 0x80) == 0);  // Wait until flash controller has finished
  //
  // Check current data flash & EEPROM config
  //
  v = FCCOB6;                      // IFR offset 0xFD
  if (v != 0xFF) {                 // EEPROM data set size already configured?
    return;
  }
  v = FCCOB7;                      // IFR offset 0xFC
  if (v != 0xFF) {                 // FlexNVM partition code already configured?
    return;
  }
  //
  // Configure EEPROM size and data flash size
  // via the program partition command
  //
  FCCOB0 = 0x80;                   // Program partition
  FCCOB4 = 0x3F;                   // EEPROM data size code: 0 KB EEPROM
  FCCOB5 = 0x00;                   // FlexNVM partition code: 256 KB data flash
  FSTAT  = 0x80;                   // Start command execution
  while((FSTAT & 0x80) == 0);      // Wait until flash controller has finished
}
```

10.7 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

Currently, there are no specifics for these devices.

10.8 Itron

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

- TRIFECTA

Currently, there are no specifics for these devices.

10.9 Infineon

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

- UMF1110
- UMF1120
- UMF5110
- UMF5120
- XMC1100-T016F00xx
- XMC1100-T038F00xx
- XMC1100-T038F0xxx
- XMC1201-T028F0xxx
- XMC1201-T038F0xxx
- XMC1202-T016X00xx
- XMC1202-T028X00xx
- XMC1202-T038X00xx
- XMC1203-T016X0xxx
- XMC1301-T016F00xx
- XMC1302-T038X0xxx
- XMC4100-128
- XMC4104-128
- XMC4104-64
- XMC4200-256
- XMC4400-256
- XMC4400-512
- XMC4402-256
- XMC4500-1024
- XMC4500-768
- XMC4502
- XMC4504

Currently, there are no specifics for these devices.

10.10 Luminary Micro

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

- LM3S101
- LM3S102
- LM3S301
- LM3S310
- LM3S315
- LM3S316
- LM3S317
- LM3S328
- LM3S601
- LM3S610
- LM3S611
- LM3S612
- LM3S613
- 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

10.10.1 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" comnmand in J-Link Commander.

10.11 NXP

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

- LPC1111
- LPC1113
- LPC1311
- LPC1313
- LPC1342
- LPC1343
- LPC1751
- LPC1751
- LPC1752
- LPC1754
- LPC1756
- LPC1758
- LPC1764
- LPC1765
- LPC1766
- LPC1768
- LPC2101
- LPC2102
- LPC2103
- LPC2104
- LPC2105
- LPC2106
- LPC2109
- LPC2114
- LPC2119
- LPC2124
- LPC2129
- LPC2131
- LPC2132
- LPC2134
- LPC2136
- LPC2138
- 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

10.11.1 LPC ARM7-based devices

10.11.1.1Fast 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.

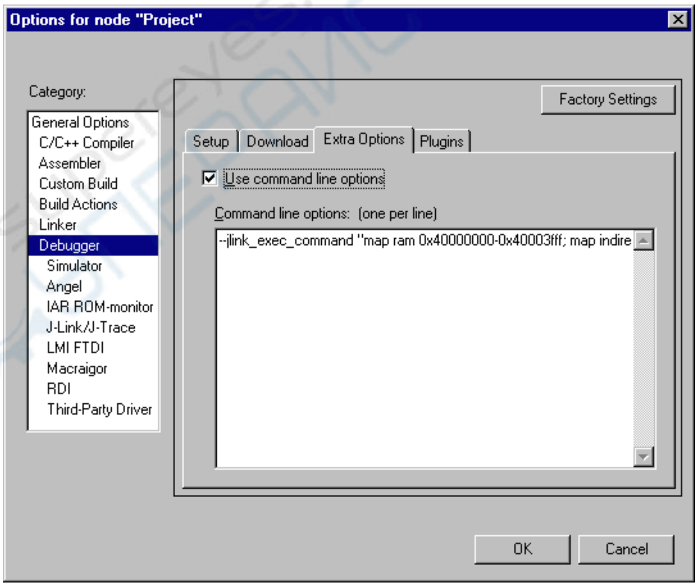
Define a 256 byte aligned area in RAM of the LPC target device with the J-Link command `map ram` and define afterwards the memory area which should be read indirect with the command `map indirectread` to use the indirectly reading feature of J-Link. Note that the data in the defined RAM area is saved 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 **Debugger**. Select **Use command line options** in the **Extra Options** tap and enter in the textfield `--jlink_exec_command "map ram 0x40000000-0x40003fff; map indirectread 0x3fffc000-0x3fffcfff; map exclude 0x3fffd000-0x3fffffff;"` as shown in the screenshot below.



☑ Use command line options

Command line options: (one per line)

```
--jlink_exec_command "map ram 0x40000000-0x40003fff; map indire
```

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 180.

10.11.1.2RDI

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 49.

10.11.2 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.

10.11.3 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 @ addr 0x10400000. Otherwise it is user’s responsibility to ensure that flash is re-mapped to 0x0 in order to debug the application from addr 0x0.

10.11.4 LPC43xx:

All devices of the LPC43xx are dual core devices (One Cortex-M4 core and one Cortex-M0 core). For these devices, a J-Link script file is needed (exact file depends on if the Cortex-M4 or the Cortex-M0 shall be debugged) in order to guarantee proper functionality.

Script file can be found at $JLINK_INST_DIR$\Samples\JLink\Scripts

For more information about how to use J-Link script files, please refer to *Executing J-Link script files* on page 178.

10.12 OKI

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

- ML67Q4002
- ML67Q4003
- ML67Q4050
- ML67Q4051
- ML67Q4060
- ML67Q4061

Currently, there are no specifics for these devices.

10.13 Renesas

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

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

Currently, there are no specifics for these devices.

10.14 Samsung

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

- S3FN60D

10.14.1 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.

10.15 Silicon Labs

J-Link has been tested with the following Silicon Labs devices:

- EFM32G200F16
- EFM32G200F32
- EFM32G200F64
- EFM32G210F128
- EFM32G230F32
- EFM32G230F64
- EFM32G230F128
- EFM32G280F32
- EFM32G280F64
- EFM32G280F128
- EFM32G290F32
- EFM32G290F64
- EFM32G290F128
- EFM32G840F32
- EFM32G840F64
- EFM32G840F128
- EFM32G880F32
- EFM32G880F64
- EFM32G880F128
- EFM32G890F32
- EFM32G890F64
- EFM32G890F128
- EFM32TG108F4
- EFM32TG108F8
- EFM32TG108F16
- EFM32TG108F32
- EFM32TG110F4
- EFM32TG110F8
- EFM32TG110F16
- EFM32TG110F32
- EFM32TG210F8
- EFM32TG210F16
- EFM32TG210F32
- EFM32TG230F8
- EFM32TG230F16
- EFM32TG230F32
- EFM32TG840F8
- EFM32TG840F16
- EFM32TG840F32
- EM351
- EM357

10.15.1 EFM32 series devices

10.15.1.1SWO

Usually, the SWO output frequency of a device is directly dependent on the CPU speed. The SWO speed is calculated as: <CPUFreq> / n. On the EFM32 series this is not the case:
The SWO related units (ITM, TPIU, ...) are chip-internally wired to a fixed 14 MHz clock (AUXHFRCO).
This will cause the auto-detection of J-Link to not work by default for these devices, if the CPU is running at a different speed than AUXHFRCO. All utilities that use SWO speed auto-detection, like the J-Link SWOViewer, need to be told that the CPU is running at 14 MHz, to make SWO speed auto-detection to work, no matter what speed the CPU is really running at.

10.16 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

10.16.1 STR91x

10.16.1.1JTAG settings

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

10.16.1.2Unlocking

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 117.

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.

10.16.1.3Switching 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 117.

10.16.2 STM32F10xxx

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

10.16.2.1ETM init

The following sequence can be used to prepare STM32F10xxx devices for 4-bit ETM tracing:

```
int v;
//
// DBGMCU_CR, enable trace I/O and configure pins for 4-bit trace.
//
v = *((volatile int *)(0xE0042004));
v &= ~(7 << 5); // Preserve all bits except the trace pin configuration
v |= (7 << 5); // Enable trace I/O and configure pins for 4-bit trace
*((volatile int *)(0xE0042004)) = v;
```

10.16.2.2 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

Table 10.1: 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 10.2: Option bytes programming example

For a detailed description of each option byte, please refer to *ST programming manual PM0042, section "Option byte description".*

10.16.2.3Securing/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 Unlock (Command line tool)* on page 118.

Note: Unsecuring a secured device will cause a mass-erase of the internal flash memory.

10.16.2.4Hardware 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 Unlock (Command line tool)* on page 118.

10.16.2.5Debugging with software watchdog enabled

If the device shall be debugged with one of the software watchdogs (independed watchdog / window watchdog) enabled, there is an additional init step necessary to make the watchdog counter stop when the CPU is halted by the debugger. This is configured in the DBGMCU_CR register. The following sequence can be used to enable debugging with software watchdogs enabled:

```
//
// Configure both watchdog timers to be halted if the CPU is halted by the debugger
//
*((volatile int *)(0xE0042004)) |= (1 << 8) | (1 << 9);
```

10.16.3 STM32F2xxx

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

10.16.3.1ETM init

The following sequence can be used to prepare STM32F2xxx devices for 4-bit ETM tracing:

```
int v;
//
// Enable GPIOE clock
//
*((volatile int *)(0x40023830)) = 0x00000010;
//
// Assign trace pins to alternate function in order
// to make them usable as trace pins
// PE2: Trace clock
// PE3: TRACE_D0
// PE4: TRACE_D1
// PE5: TRACE_D2
// PE6: TRACE_D3
//
*((volatile int *)(0x40021000)) = 0x00002AA0;
//
// DBGMCU_CR, enable trace I/O and configure pins for 4-bit trace.
//
v = *((volatile int *)(0xE0042004));
v &= ~(7 << 5); // Preserve all bits except the trace pin configuration
v |= (7 << 5); // Enable trace I/O and configure pins for 4-bit trace
*((volatile int *)(0xE0042004)) = v;
```

10.16.3.2Debugging with software watchdog enabled

If the device shall be debugged with one of the software watchdogs (independed watchdog / window watchdog) enabled, there is an additional init step necessary to make the watchdog counter stop when the CPU is halted by the debugger. This is configured in the DBGMCU_APB1_FZ register. The following sequence can be used to enable debugging with software watchdogs enabled:

```
//
// Configure both watchdog timers to be halted if the CPU is halted by the debugger
//
*((volatile int *)(0xE0042008)) |= (1 << 11) | (1 << 12);
```

10.16.4 STM32F4xxx

These device are Cortex-M4 based.
All devices of this family are supported by J-Link.

10.16.4.1ETM init

The following sequence can be used to prepare STM32F4xxx devices for 4-bit ETM tracing:

```
int v;
//
// Enable GPIOE clock
//
*((volatile int *)(0x40023830)) = 0x00000010;
//
// Assign trace pins to alternate function in order
// to make them usable as trace pins
// PE2: Trace clock
// PE3: TRACE_D0
// PE4: TRACE_D1
// PE5: TRACE_D2
// PE6: TRACE_D3
//
*((volatile int *)(0x40021000)) = 0x00002AA0;
//
// DBGMCU_CR, enable trace I/O and configure pins for 4-bit trace.
//
v = *((volatile int *)(0xE0042004));
v &= ~(7 << 5); // Preserve all bits except the trace pin configuration
v |= (7 << 5); // Enable trace I/O and configure pins for 4-bit trace
*((volatile int *)(0xE0042004)) = v;
```

10.16.4.2Debugging with software watchdog enabled

If the device shall be debugged with one of the software watchdogs (independed watchdog / window watchdog) enabled, there is an additional init step necessary to make the watchdog counter stop when the CPU is halted by the debugger. This is configured in the DBGMCU_APB1_FZ register. The following sequence can be used to enable debugging with software watchdogs enabled:

```
//
// Configure both watchdog timers to be halted if the CPU is halted by the debugger
//
*((volatile int *)(0xE0042008)) |= (1 << 11) | (1 << 12);
```

10.17 Texas Instruments

J-Link has been tested with the following Texas Instruments devices:

- AM3352
- AM3354
- AM3356
- AM3357
- AM3358
- AM3359
- OMAP3530
- OMAP3550
- OMAP4430
- OMAP-L138
- TMS470M
- TMS470R1A64
- TMS470R1A128
- TMS470R1A256
- TMS470R1A288
- TMS470R1A384
- TMS470R1B512
- TMS470R1B768
- TMS470R1B1M
- TMS470R1VF288
- TMS470R1VF688
- TMS470R1VF689

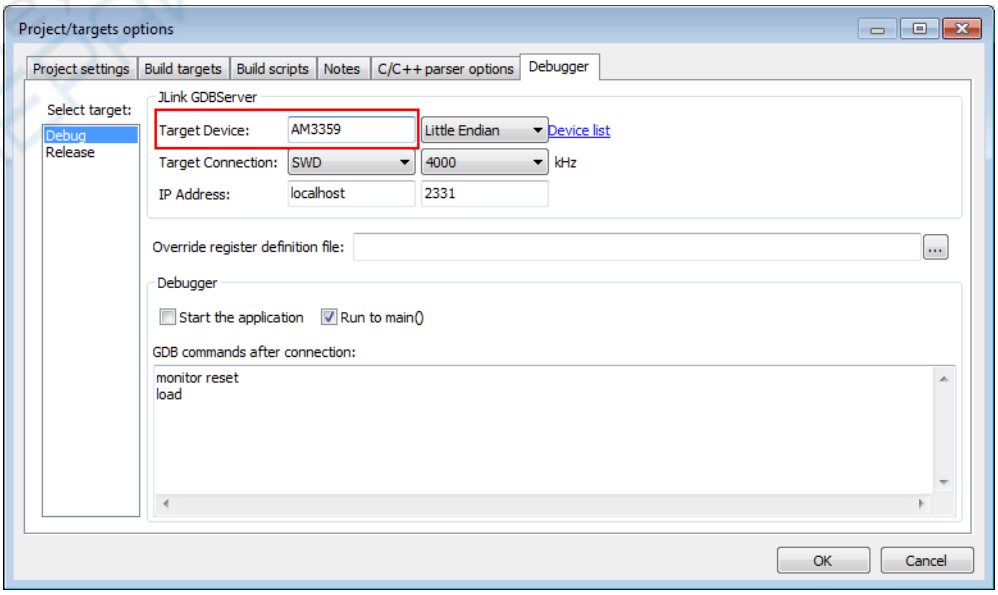
10.17.1 AM335x

The AM335x series devices need some special handling which requires that correct device is selected in the J-Link DLL. When used out of a debugger, this is usually done automatically (see *Software reset* on page 266). For J-Link Commander & J-Link GDBServer this needs to be done automatically

10.17.1.1Selecting the device in the IDE

When using J-Link in an IDE, there is usually a way to directly select the device in the IDE, since it usually also needs this information for peripheral register view etc. The selected device is then usually automatically passed to the J-Link DLL.

The screenshot below is an example for a device selection inside emIDE (http://www.emide.org).



Select target: Debug
Release

JLink GDBServer

Target Device: AM3359 Little Endian

Target Connection: SWD 4000 kHz

IP Address: localhost 2331

Override register definition file:

Debugger

□ Start the application ☑ Run to main()

GDB commands after connection:

```
monitor reset
load
```

10.17.1.2Selecting 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
```

10.17.1.3Selecting 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.

10.17.1.4Known values for <DeviceName>

For a list of all supported devices, please refer to http://www.segger.com/jlink_supported_devices.html

10.17.1.5Required J-Link hardware version

The special handling for the AM335x cannot be supported by some older hardware versions of J-Link, so the device cannot be used with these versions.

The following hardware versions come with AM335x support:

- J-Link V8 or later
- J-Link PRO V3 or later
- J-Link ULTRA V4 or later
- Flasher ARM V4 or later

10.17.2 AM35xx / AM37xx

Script is not needed. Refer to AM335x special handling. Same needs to be done for AM35xx / AM37xx.

10.17.3 OMAP4430

Script is not needed. Refer to AM335x special handling. Same needs to be done for AM35xx / AM37xx.

10.17.4 OMAP-L138

Needs a J-Link script file to guarantee proper functionality.

J-Link script file can be found at `$JLINK_INST_DIR$\Samples\JLink\Scripts`

For more information about how to use J-Link script files, please refer to *Executing J-Link script files* on page 178.

10.17.5 TMS470M

Needs a J-Link script file to guarantee proper functionality.

J-Link script file can be found at `$JLINK_INST_DIR$\Samples\JLink\Scripts`

For more information about how to use J-Link script files, please refer to *Executing J-Link script files* on page 178.

10.17.6 OMAP3530

Needs a J-Link script file to guarantee proper functionality.

J-Link script file can be found at `$JLINK_INST_DIR$\Samples\JLink\Scripts`

For more information about how to use J-Link script files, please refer to *Executing J-Link script files* on page 178.

10.17.7 OMAP3550

Needs a J-Link script file to guarantee proper functionality.

J-Link script file can be found at `$JLINK_INST_DIR$\Samples\JLink\Scripts`

For more information about how to use J-Link script files, please refer to *Executing J-Link script files* on page 178.

10.18 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 11

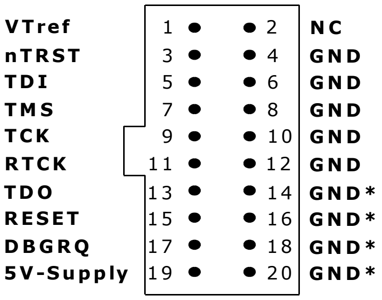
Target interfaces and adapters

This chapter gives an overview about J-Link / J-Trace specific hardware details, such as the pinouts and available adapters.

11.1 20-pin JTAG/SWD connector

11.1.1 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.



VTref	1 ●	● 2	NC
nTRST	3 ●	● 4	GND
TDI	5 ●	● 6	GND
TMS	7 ●	● 8	GND
TCK	9 ●	● 10	GND
RTCK	11 ●	● 12	GND
TDO	13 ●	● 14	GND*
RESET	15 ●	● 16	GND*
DBGRQ	17 ●	● 18	GND*
5V-Supply	19 ●	● 20	GND*

*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.

The following table lists the J-Link / J-Trace JTAG 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 connected	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	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.
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. Connect to RTCK if available, otherwise to GND.
13	TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU.

Table 11.1: 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 compatibility 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-Supply	Output	This pin can be used to supply power to the target hardware. 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 *Target power supply* on page 302.

Table 11.1: 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.

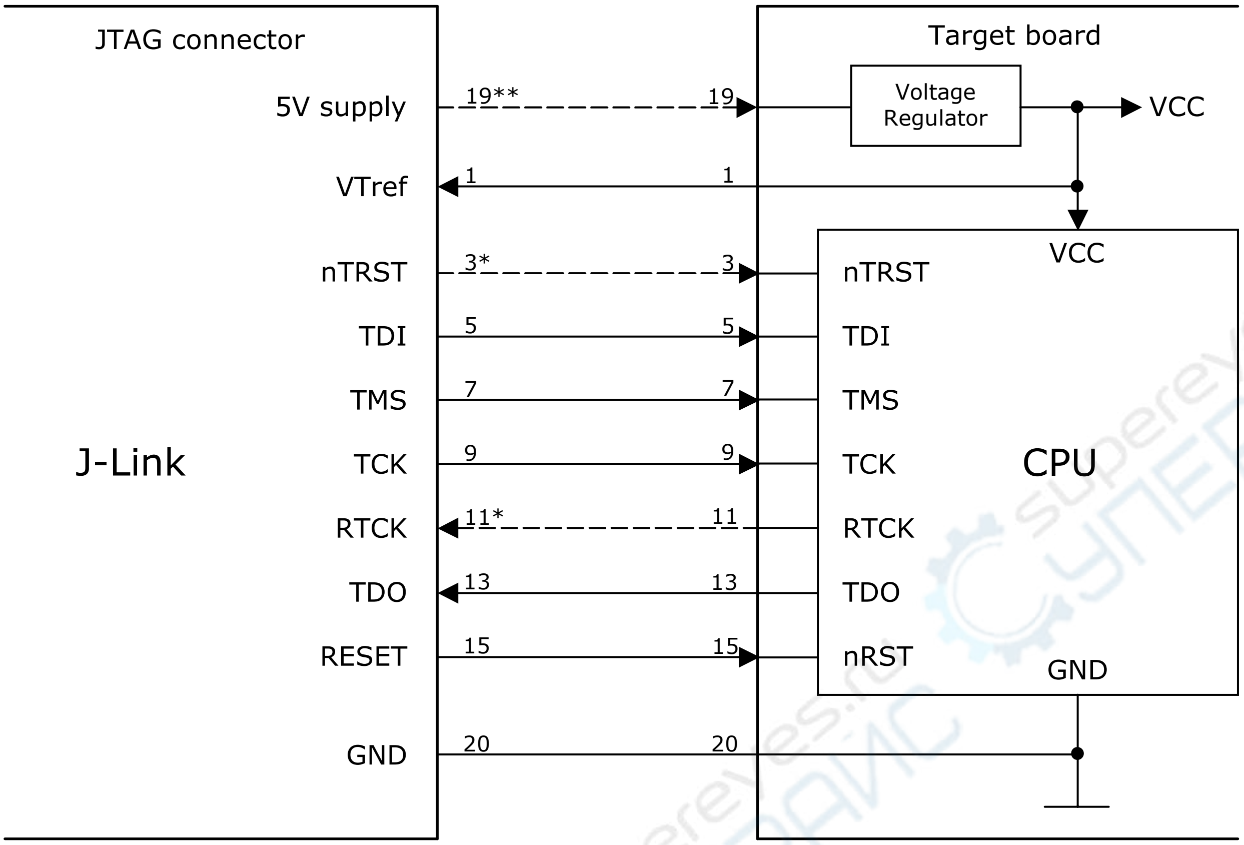
11.1.1.1 Target board design

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 300. 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.

11.1.1.2 Pull-up/pull-down resistors

Unless otherwise specified by developer's manual, pull-ups/pull-downs are recommended to 100 kOhms.

11.1.1.3 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:

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"

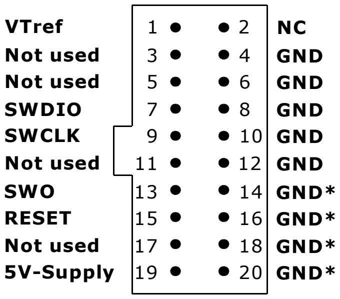
Table 11.2: Command List

11.1.2 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.

The following table lists the J-Link / J-Trace SWD pinout.



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***

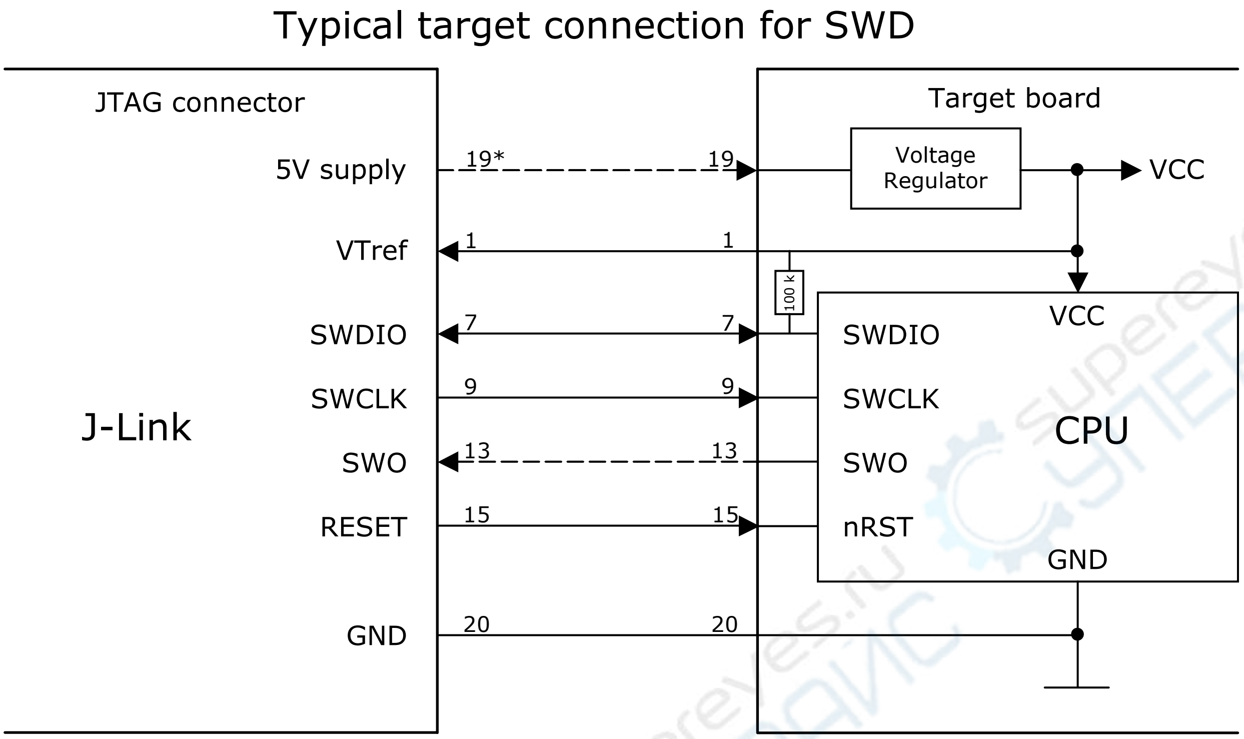
PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	Not connected	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	Input	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-Supply	Output	This pin can be used to supply power to the target hardware. 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 *Target power supply* on page 304.

Table 11.3: 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.

11.1.2.1 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 303. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.



* Optional to supply the target board from J-Link.

11.1.2.2 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.

11.1.2.3 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"

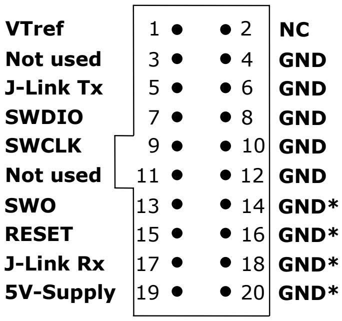
Table 11.4: Command List

11.1.3 Pinout for SWD + Virtual COM Port (VCOM)

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.

The following table lists the J-Link / J-Trace SWD pinout.



VTref	1 ●	● 2	NC
Not used	3 ●	● 4	GND
J-Link Tx	5 ●	● 6	GND
SWDIO	7 ●	● 8	GND
SWCLK	9 ●	● 10	GND
Not used	11 ●	● 12	GND
SWO	13 ●	● 14	GND*
RESET	15 ●	● 16	GND*
J-Link Rx	17 ●	● 18	GND*
5V-Supply	19 ●	● 20	GND*

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 connected	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	J-Link Tx	Output	This pin is used as VCOM Tx (out on J-Link side) in case VCOM functionality of J-Link is enabled. For further information about VCOM, please refer to *Virtual COM Port (VCOM)* on page 189.
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	Input	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	J-Link Rx	input	This pin is used as VCOM Rx (in on J-Link side) in case VCOM functionality of J-Link is enabled. For further information, please refer to *Virtual COM Port (VCOM)* on page 189.
19	5V-Supply	Output	This pin can be used to supply power to the target hardware. 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 *Target power supply* on page 304.

Table 11.5: J-Link / J-Trace SWD pinout

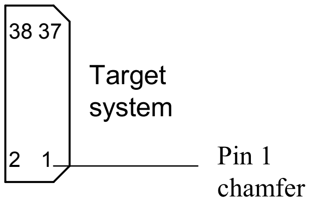
Pins 4, 6, 8, 10, 12, 14, 16, 18, 20 are GND pins connected to GND in J-Link. They should also be connected to GND in the target system.

11.2 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

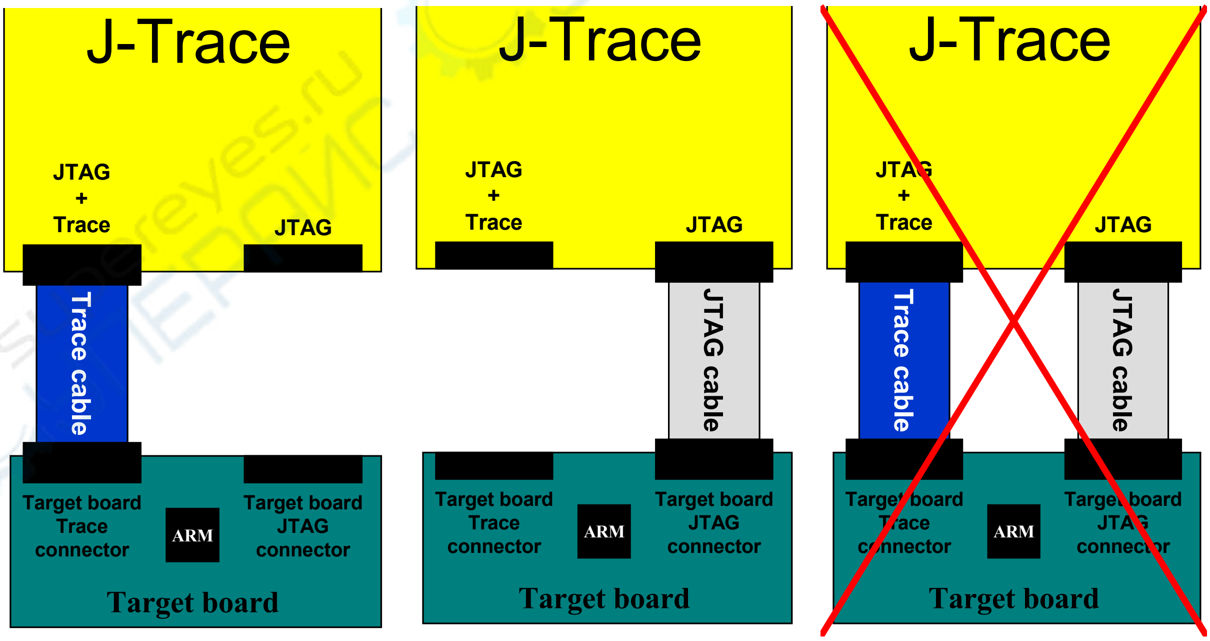


J-Trace can capture the state of signals PIPESTAT[2:0], TRACESYNC and TRACEPKT[n:0] at each rising edge of each TRACECLK or on each alternate rising or falling edge.

11.2.1 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.


J-Trace
JTAG + Trace
JTAG
Trace cable
Target board Trace connector
ARM
Target board JTAG connector
Target board
J-Trace
JTAG + Trace
JTAG
JTAG cable
Target board Trace connector
ARM
Target board JTAG connector
Target board
J-Trace
JTAG + Trace
JTAG
Trace cable
JTAG cable
Target board Trace connector
ARM
Target board JTAG connector
Target board

11.2.2 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	TCK	Test clock to the run control unit from the JTAG port.
16	Trace signal 12	Trace signal. For more information, please refer to *Assignment of trace information pins between ETM architecture versions* on page 309.
17	TMS	Test mode select from run control to the JTAG port.
18	Trace signal 11	Trace signal. For more information, please refer to *Assignment of trace information pins between ETM architecture versions* on page 309.
19	TDI	Test data input from run control to the JTAG port.
20	Trace signal 10	Trace signal. For more information, please refer to *Assignment of trace information pins between ETM architecture versions* on page 309.
21	nTRST	Active-low JTAG reset

Table 11.6: JTAG+Trace connector pinout

PIN	SIGNAL	Description
22	Trace signal 9	Trace signals. For more information, please refer to *Assignment of trace information pins between ETM architecture versions* on page 309.
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	
30	Trace signal 5	
31	Trace signal 16	
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 11.6: JTAG+Trace connector pinout

11.2.3 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 11.7: Assignment of trace information pins between ETM architecture versions

11.2.4 Trace signals

Data transfer is synchronized by TRACECLK.

11.2.4.1 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.

11.2.4.2 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 TRACECLK.

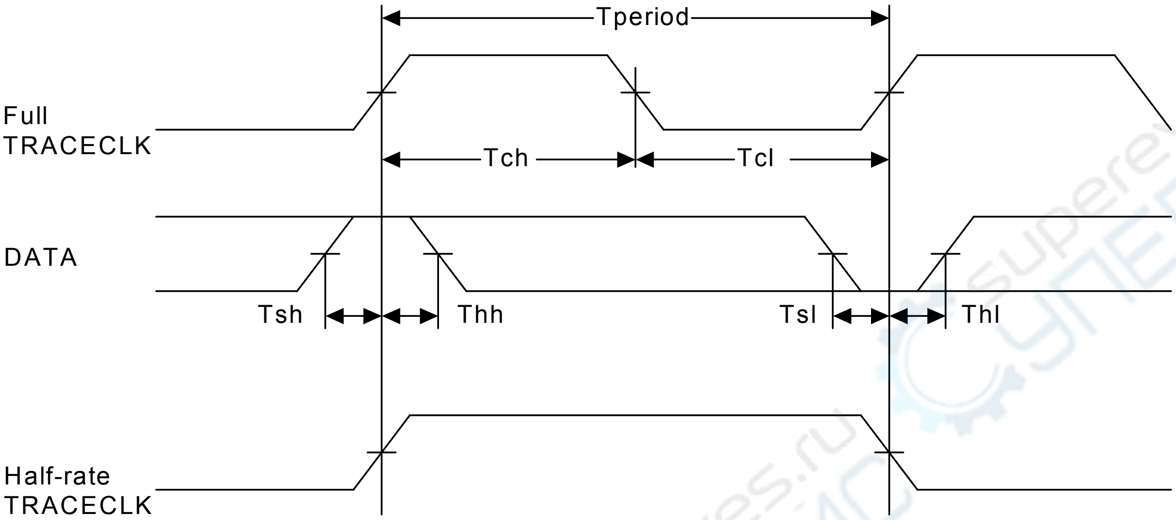
Parameter	Min.	Max.	Explanation
Tperiod	5ns	1000ns	Clock period
Fmax	1MHz	200MHz	Maximum trace frequency
Tch	2.5ns	-	High pulse width
Tcl	2.5ns	-	Low pulse width

Table 11.8: Clock frequency

Parameter	Min.	Max.	Explanation
Tsh	2.5ns	-	Data setup high
Thh	1.5ns	-	Data hold high
Tsl	2.5ns	-	Data setup low
Thl	1.5ns	-	Data hold low

Table 11.8: Clock frequency

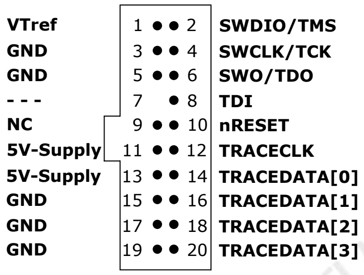
The diagram below shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACECLK.



Note: J-Trace supports half-rate clocking mode. Data is output on each edge of the TRACECLK signal and TRACECLK (max) <= 100MHz. For half-rate clocking, the setup and hold times at the JTAG+Trace connector must be observed.

11.3 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.



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	SWDIO: (Single) bi-directional data pin. 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	SWCLK: 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 target CPU. JTAG clock signal to 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 hardware.
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 hardware. For more information about how to enable/disable the power supply, please refer to *Target power supply* on page 312.
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 hardware. For more information about how to enable/disable the power supply, please refer to *Target power supply* on page 312.
14	TRACE-DATA[0]	Input	Input Trace data pin 0.

Table 11.9: 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 11.9: 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.

11.3.1 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.

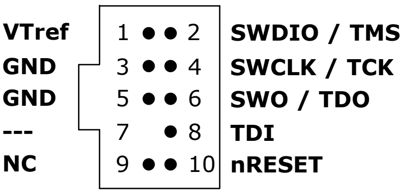
Power can be controlled via the J-Link commander. The following commands are available to control power:

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 11.10: Command List

11.4 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.



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	SWDIO: (Single) bi-directional data pin. 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	SWCLK: 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 target CPU. JTAG clock signal to target CPU.
6	SWO/TDO	Input	When using SWD, this pin is used as Serial Wire Output trace port (optional, not required for SWD communication). 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 hardware.

Table 11.11: 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.

11.5 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 12

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).

12.1 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.

12.1.1 Test access port (TAP)

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

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

Table 12.1: Test access port

12.1.2 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.

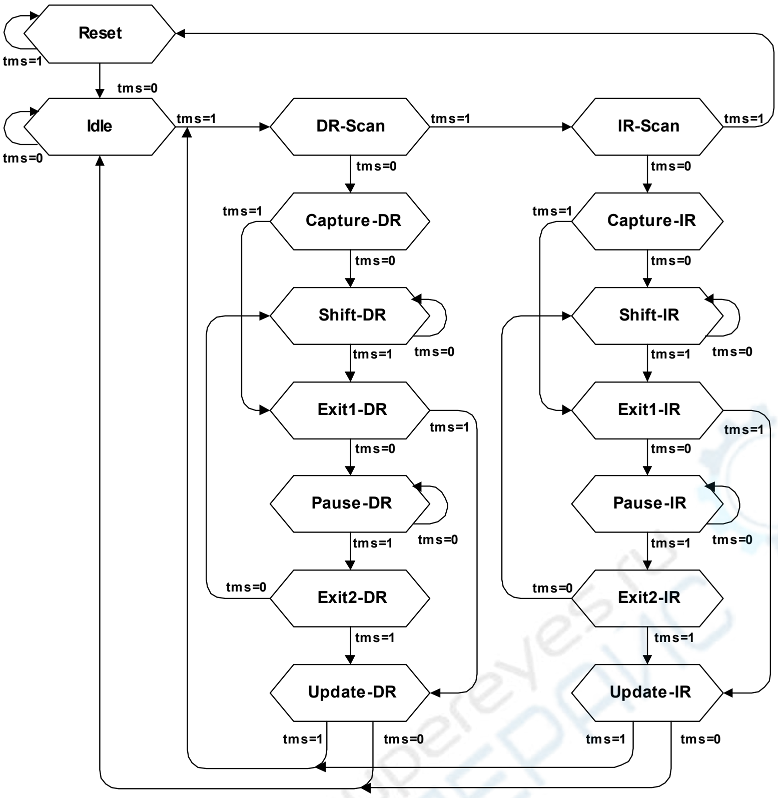
12.1.3 Instruction register

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.

12.1.4 The TAP controller

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

TAP controller state diagram



12.1.4.1 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.

12.2 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.

12.2.1 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.

12.2.2 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.

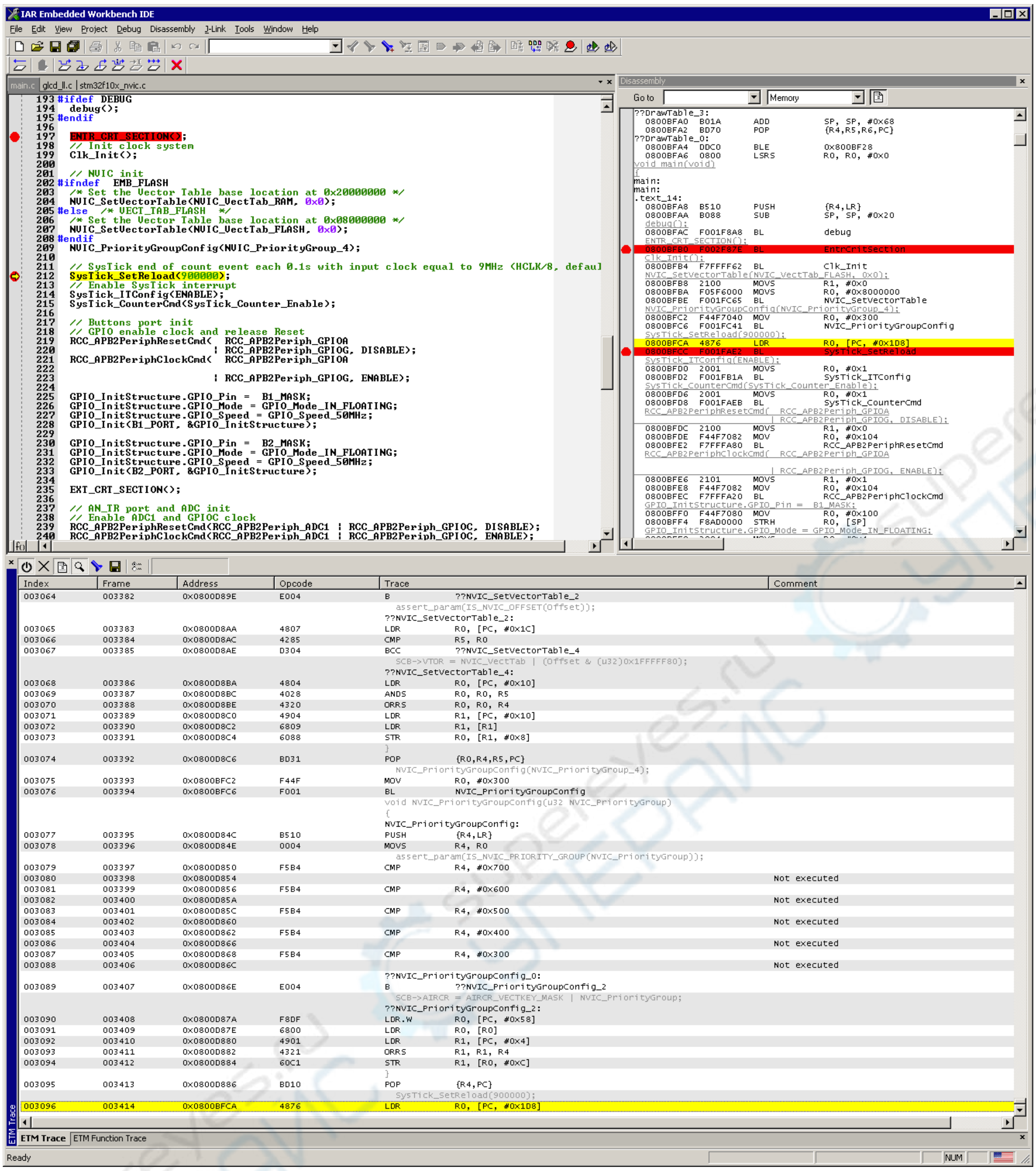
Data trace

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.

12.2.3 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.

12.2.3.1 Code coverage - Disassembly tracing

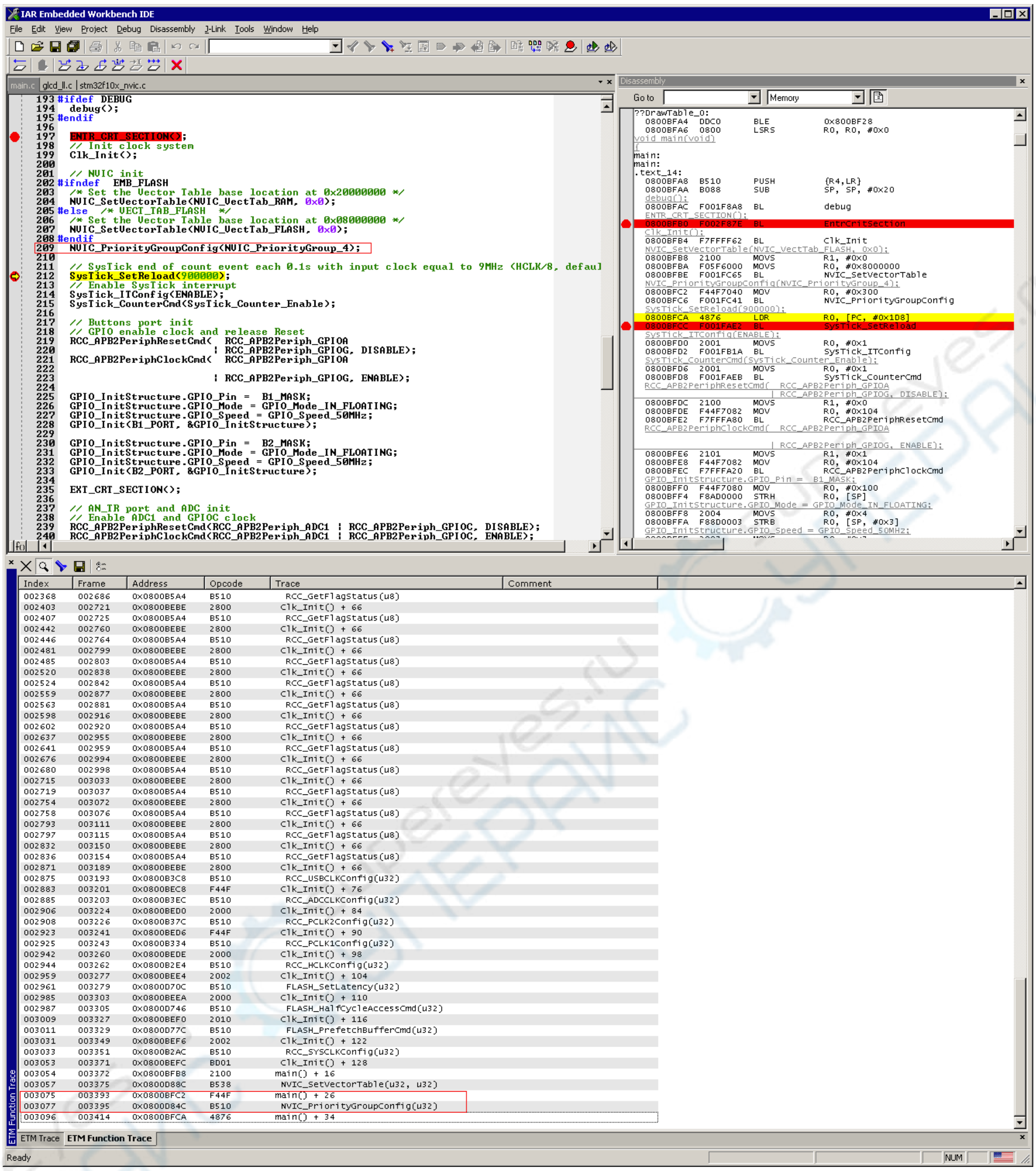

IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly J-Link Tools Window Help
main.c | glcd_ll.c | stm32f10x_nvic.c
Disassembly
Go to
Memory
ETM Trace
ETM Function Trace
Ready
NUM

```
#ifdef DEBUG
  debug();
#endif

  ENTR_CRT_SECTION();
  // Init clock system
  Clk_Init();

  // NVIC init
#ifndef  EMB_FLASH
  /* Set the Vector Table base location at 0x20000000 */
  NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
#else  /* VECT_TAB_FLASH  */
  /* Set the Vector Table base location at 0x08000000 */
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
#endif
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

  // SysTick end of count event each 0.1s with input clock equal to 9MHz (HCLK/8, defaul
  SysTick_SetReload(900000);
  // Enable SysTick interrupt
  SysTick_ITConfig(ENABLE);
  SysTick_CounterCmd(SysTick_Counter_Enable);

  // Buttons port init
  // GPIO enable clock and release Reset
  RCC_APB2PeriphResetCmd(  RCC_APB2Periph_GPIOA
                         | RCC_APB2Periph_GPIOG, DISABLE);
  RCC_APB2PeriphClockCmd(  RCC_APB2Periph_GPIOA

                         | RCC_APB2Periph_GPIOG, ENABLE);

  GPIO_InitStructure.GPIO_Pin =  B1_MASK;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(B1_PORT, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin =  B2_MASK;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(B2_PORT, &GPIO_InitStructure);

  EXT_CRT_SECTION();

  // AN_TR port and ADC init
  // Enable ADC1 and GPIOC clock
  RCC_APB2PeriphResetCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, DISABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, ENABLE);
```

```
??DrawTable_3:
  0800BFA0  B01A      ADD        SP, SP, #0x68
  0800BFA2  BD70      POP        {R4,R5,R6,PC}
??DrawTable_0:
  0800BFA4  DDC0      BLE        0x800BF28
  0800BFA6  0800      LSRS       R0, R0, #0x0
void main(void)
{
main:
main:
.text_14:
  0800BFA8  B510      PUSH       {R4,LR}
  0800BFAA  B088      SUB        SP, SP, #0x20
  debug();
  0800BFAC  F001F8A8  BL         debug
  ENTR_CRT_SECTION();
  0800BFB0  F002F87E  BL         EntrCritSection
  Clk_Init();
  0800BFB4  F7FFFF62  BL         Clk_Init
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
  0800BFB8  2100      MOVS       R1, #0x0
  0800BFBA  F05F6000  MOVS       R0, #0x8000000
  0800BFBE  F001FC65  BL         NVIC_SetVectorTable
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
  0800BFC2  F44F7040  MOV        R0, #0x300
  0800BFC6  F001FC41  BL         NVIC_PriorityGroupConfig
  SysTick_SetReload(900000);
  0800BFCA  4876      LDR        R0, [PC, #0x1D8]
  0800BFCC  F001FAE2  BL         SysTick_SetReload
  SysTick_ITConfig(ENABLE);
  0800BFD0  2001      MOVS       R0, #0x1
  0800BFD2  F001FB1A  BL         SysTick_ITConfig
  SysTick_CounterCmd(SysTick_Counter_Enable);
  0800BFD6  2001      MOVS       R0, #0x1
  0800BFD8  F001FAEB  BL         SysTick_CounterCmd
  RCC_APB2PeriphResetCmd(  RCC_APB2Periph_GPIOA
                         | RCC_APB2Periph_GPIOG, DISABLE);
  0800BFDC  2100      MOVS       R1, #0x0
  0800BFDE  F44F7082  MOV        R0, #0x104
  0800BFE2  F7FFFA80  BL         RCC_APB2PeriphResetCmd
  RCC_APB2PeriphClockCmd(  RCC_APB2Periph_GPIOA

                         | RCC_APB2Periph_GPIOG, ENABLE);
  0800BFE6  2101      MOVS       R1, #0x1
  0800BFE8  F44F7082  MOV        R0, #0x104
  0800BFEC  F7FFFA20  BL         RCC_APB2PeriphClockCmd
  GPIO_InitStructure.GPIO_Pin =  B1_MASK;
  0800BFF0  F44F7080  MOV        R0, #0x100
  0800BFF4  F8AD0000  STRH       R0, [SP]
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
```

```
Index    Frame    Address      Opcode   Trace                                                    Comment
003064   003382   0x0800D89E   E004     B          ??NVIC_SetVectorTable_2
                                          assert_param(IS_NVIC_OFFSET(Offset));
                                        ??NVIC_SetVectorTable_2:
003065   003383   0x0800D8AA   4807     LDR        R0, [PC, #0x1C]
003066   003384   0x0800D8AC   4285     CMP        R5, R0
003067   003385   0x0800D8AE   D304     BCC        ??NVIC_SetVectorTable_4
                                          SCB->VTOR = NVIC_VectTab | (Offset & (u32)0x1FFFFF80);
                                        ??NVIC_SetVectorTable_4:
003068   003386   0x0800D8BA   4804     LDR        R0, [PC, #0x10]
003069   003387   0x0800D8BC   4028     ANDS       R0, R0, R5
003070   003388   0x0800D8BE   4320     ORRS       R0, R0, R4
003071   003389   0x0800D8C0   4904     LDR        R1, [PC, #0x10]
003072   003390   0x0800D8C2   6809     LDR        R1, [R1]
003073   003391   0x0800D8C4   6088     STR        R0, [R1, #0x8]
                                        }
003074   003392   0x0800D8C6   BD31     POP        {R0,R4,R5,PC}
                                          NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
003075   003393   0x0800BFC2   F44F     MOV        R0, #0x300
003076   003394   0x0800BFC6   F001     BL         NVIC_PriorityGroupConfig
                                        void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup)
                                        {
                                        NVIC_PriorityGroupConfig:
003077   003395   0x0800D84C   B510     PUSH       {R4,LR}
003078   003396   0x0800D84E   0004     MOVS       R4, R0
                                          assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));
003079   003397   0x0800D850   F5B4     CMP        R4, #0x700
003080   003398   0x0800D854                                                                     Not executed
003081   003399   0x0800D856   F5B4     CMP        R4, #0x600
003082   003400   0x0800D85A                                                                     Not executed
003083   003401   0x0800D85C   F5B4     CMP        R4, #0x500
003084   003402   0x0800D860                                                                     Not executed
003085   003403   0x0800D862   F5B4     CMP        R4, #0x400
003086   003404   0x0800D866                                                                     Not executed
003087   003405   0x0800D868   F5B4     CMP        R4, #0x300
003088   003406   0x0800D86C                                                                     Not executed
                                        ??NVIC_PriorityGroupConfig_0:
003089   003407   0x0800D86E   E004     B          ??NVIC_PriorityGroupConfig_2
                                          SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
                                        ??NVIC_PriorityGroupConfig_2:
003090   003408   0x0800D87A   F8DF     LDR.W      R0, [PC, #0x58]
003091   003409   0x0800D87E   6800     LDR        R0, [R0]
003092   003410   0x0800D880   4901     LDR        R1, [PC, #0x4]
003093   003411   0x0800D882   4321     ORRS       R1, R1, R4
003094   003412   0x0800D884   60C1     STR        R1, [R0, #0xC]
                                        }
003095   003413   0x0800D886   BD10     POP        {R4,PC}
                                          SysTick_SetReload(900000);
003096   003414   0x0800BFCA   4876     LDR        R0, [PC, #0x1D8]
```

12.2.3.2 Code coverage - Source code tracing

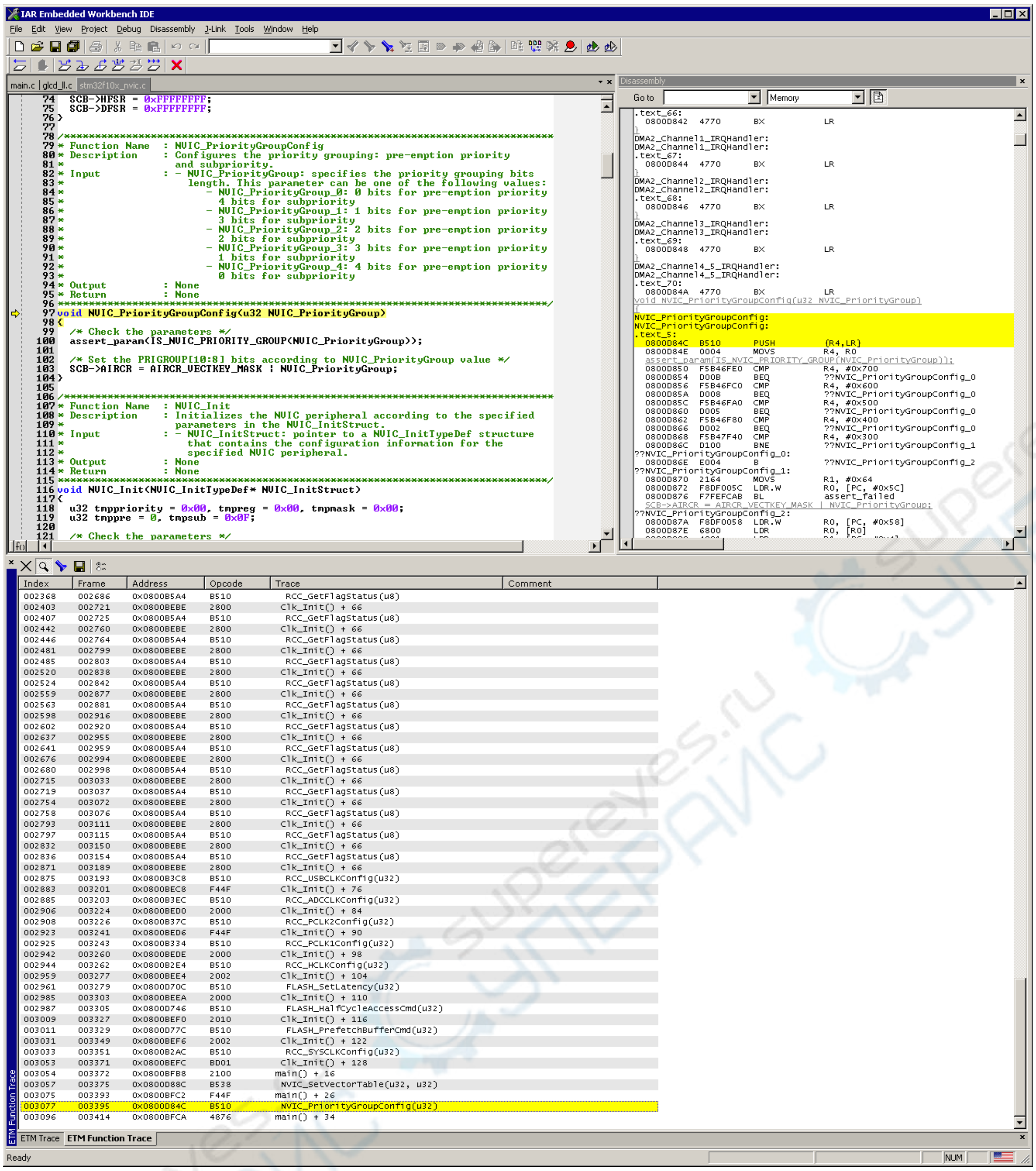


IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly J-Link Tools Window Help
main.c glcd_ll.c stm32f10x_nvic.c
Disassembly
Go to
Memory
ETM Trace
ETM Function Trace
Ready
NUM

```
#ifdef DEBUG
  debug();
#endif

  ENTR_CRT_SECTION();
  // Init clock system
  Clk_Init();

  // NVIC init
#ifndef  EMB_FLASH
  /* Set the Vector Table base location at 0x20000000 */
  NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
#else  /* VECT_TAB_FLASH  */
  /* Set the Vector Table base location at 0x08000000 */
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
#endif
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

  // SysTick end of count event each 0.1s with input clock equal to 9MHz (HCLK/8, defaul
  SysTick_SetReload(900000);
  // Enable SysTick interrupt
  SysTick_ITConfig(ENABLE);
  SysTick_CounterCmd(SysTick_Counter_Enable);

  // Buttons port init
  // GPIO enable clock and release Reset
  RCC_APB2PeriphResetCmd(  RCC_APB2Periph_GPIOA
                         | RCC_APB2Periph_GPIOG, DISABLE);
  RCC_APB2PeriphClockCmd(  RCC_APB2Periph_GPIOA

                         | RCC_APB2Periph_GPIOG, ENABLE);

  GPIO_InitStructure.GPIO_Pin =  B1_MASK;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(B1_PORT, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin =  B2_MASK;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(B2_PORT, &GPIO_InitStructure);

  EXT_CRT_SECTION();

  // AN_TR port and ADC init
  // Enable ADC1 and GPIOC clock
  RCC_APB2PeriphResetCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, DISABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, ENABLE);
```

```
??DrawTable_0:
  0800BFA4  DDC0      BLE        0x800BF28
  0800BFA6  0800      LSRS       R0, R0, #0x0
void main(void)
{
main:
main:
.text_14:
  0800BFA8  B510      PUSH       {R4,LR}
  0800BFAA  B088      SUB        SP, SP, #0x20
  debug();
  0800BFAC  F001F8A8  BL         debug
  ENTR_CRT_SECTION();
  0800BFB0  F002F87E  BL         EntrCritSection
  Clk_Init();
  0800BFB4  F7FFFF62  BL         Clk_Init
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
  0800BFB8  2100      MOVS       R1, #0x0
  0800BFBA  F05F6000  MOVS       R0, #0x8000000
  0800BFBE  F001FC65  BL         NVIC_SetVectorTable
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
  0800BFC2  F44F7040  MOV        R0, #0x300
  0800BFC6  F001FC41  BL         NVIC_PriorityGroupConfig
  SysTick_SetReload(900000);
  0800BFCA  4876      LDR        R0, [PC, #0x1D8]
  0800BFCC  F001FAE2  BL         SysTick_SetReload
  SysTick_ITConfig(ENABLE);
  0800BFD0  2001      MOVS       R0, #0x1
  0800BFD2  F001FB1A  BL         SysTick_ITConfig
  SysTick_CounterCmd(SysTick_Counter_Enable);
  0800BFD6  2001      MOVS       R0, #0x1
  0800BFD8  F001FAEB  BL         SysTick_CounterCmd
  RCC_APB2PeriphResetCmd(  RCC_APB2Periph_GPIOA
                         | RCC_APB2Periph_GPIOG, DISABLE);
  0800BFDC  2100      MOVS       R1, #0x0
  0800BFDE  F44F7082  MOV        R0, #0x104
  0800BFE2  F7FFFA80  BL         RCC_APB2PeriphResetCmd
  RCC_APB2PeriphClockCmd(  RCC_APB2Periph_GPIOA

                         | RCC_APB2Periph_GPIOG, ENABLE);
  0800BFE6  2101      MOVS       R1, #0x1
  0800BFE8  F44F7082  MOV        R0, #0x104
  0800BFEC  F7FFFA20  BL         RCC_APB2PeriphClockCmd
  GPIO_InitStructure.GPIO_Pin =  B1_MASK;
  0800BFF0  F44F7080  MOV        R0, #0x100
  0800BFF4  F8AD0000  STRH       R0, [SP]
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  0800BFF8  2004      MOVS       R0, #0x4
  0800BFFA  F88D0003  STRB       R0, [SP, #0x3]
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

Index	Frame	Address	Opcode	Trace	Comment
002368	002686	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002403	002721	0x0800BEBE	2800	Clk_Init() + 66	
002407	002725	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002442	002760	0x0800BEBE	2800	Clk_Init() + 66	
002446	002764	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002481	002799	0x0800BEBE	2800	Clk_Init() + 66	
002485	002803	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002520	002838	0x0800BEBE	2800	Clk_Init() + 66	
002524	002842	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002559	002877	0x0800BEBE	2800	Clk_Init() + 66	
002563	002881	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002598	002916	0x0800BEBE	2800	Clk_Init() + 66	
002602	002920	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002637	002955	0x0800BEBE	2800	Clk_Init() + 66	
002641	002959	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002676	002994	0x0800BEBE	2800	Clk_Init() + 66	
002680	002998	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002715	003033	0x0800BEBE	2800	Clk_Init() + 66	
002719	003037	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002754	003072	0x0800BEBE	2800	Clk_Init() + 66	
002758	003076	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002793	003111	0x0800BEBE	2800	Clk_Init() + 66	
002797	003115	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002832	003150	0x0800BEBE	2800	Clk_Init() + 66	
002836	003154	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002871	003189	0x0800BEBE	2800	Clk_Init() + 66	
002875	003193	0x0800B3C8	B510	RCC_USBCLKConfig(u32)	
002883	003201	0x0800BEC8	F44F	Clk_Init() + 76	
002885	003203	0x0800B3EC	B510	RCC_ADCCLKConfig(u32)	
002906	003224	0x0800BED0	2000	Clk_Init() + 84	
002908	003226	0x0800B37C	B510	RCC_PCLK2Config(u32)	
002923	003241	0x0800BED6	F44F	Clk_Init() + 90	
002925	003243	0x0800B334	B510	RCC_PCLK1Config(u32)	
002942	003260	0x0800BEDE	2000	Clk_Init() + 98	
002944	003262	0x0800B2E4	B510	RCC_HCLKConfig(u32)	
002959	003277	0x0800BEE4	2002	Clk_Init() + 104	
002961	003279	0x0800D70C	B510	FLASH_SetLatency(u32)	
002985	003303	0x0800BEEA	2000	Clk_Init() + 110	
002987	003305	0x0800D746	B510	FLASH_HalfCycleAccessCmd(u32)	
003009	003327	0x0800BEF0	2010	Clk_Init() + 116	
003011	003329	0x0800D77C	B510	FLASH_PrefetchBufferCmd(u32)	
003031	003349	0x0800BEF6	2002	Clk_Init() + 122	
003033	003351	0x0800B2AC	B510	RCC_SYSCLKConfig(u32)	
003053	003371	0x0800BEFC	BD01	Clk_Init() + 128	
003054	003372	0x0800BFB8	2100	main() + 16	
003057	003375	0x0800D88C	B538	NVIC_SetVectorTable(u32, u32)	
003075	003393	0x0800BFC2	F44F	main() + 26	
003077	003395	0x0800D84C	B510	NVIC_PriorityGroupConfig(u32)	
003096	003414	0x0800BFCA	4876	main() + 34	


IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly J-Link Tools Window Help
main.c | glcd_ll.c | stm32f10x_nvic.c
Disassembly
Go to
Memory
Index Frame Address Opcode Trace Comment
ETM Function Trace
ETM Trace
ETM Function Trace
Ready
NUM

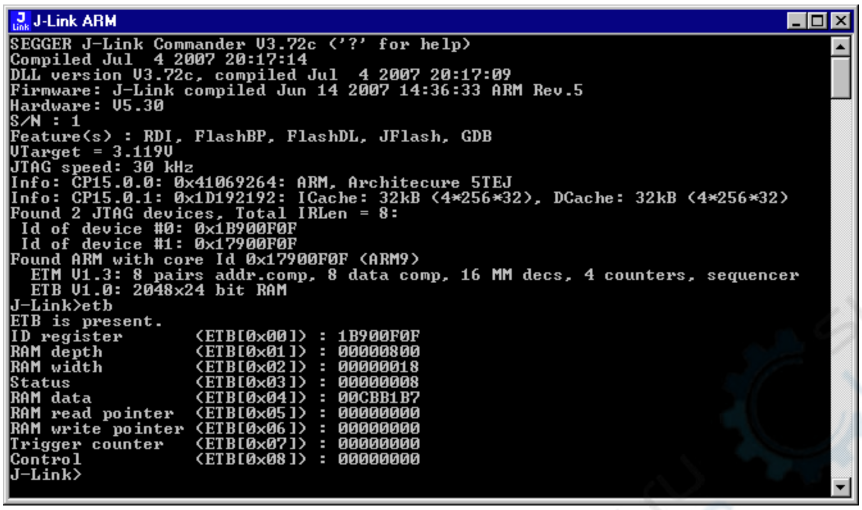
```
  SCB->HFSR = 0xFFFFFFFF;
  SCB->DFSR = 0xFFFFFFFF;
}

/*******************************************************************************
* Function Name  : NVIC_PriorityGroupConfig
* Description    : Configures the priority grouping: pre-emption priority
*                  and subpriority.
* Input          : - NVIC_PriorityGroup: specifies the priority grouping bits
*                    length. This parameter can be one of the following values:
*                       - NVIC_PriorityGroup_0: 0 bits for pre-emption priority
*                         4 bits for subpriority
*                       - NVIC_PriorityGroup_1: 1 bits for pre-emption priority
*                         3 bits for subpriority
*                       - NVIC_PriorityGroup_2: 2 bits for pre-emption priority
*                         2 bits for subpriority
*                       - NVIC_PriorityGroup_3: 3 bits for pre-emption priority
*                         1 bits for subpriority
*                       - NVIC_PriorityGroup_4: 4 bits for pre-emption priority
*                         0 bits for subpriority
* Output         : None
* Return         : None
*******************************************************************************/
void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup)
{
  /* Check the parameters */
  assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));

  /* Set the PRIGROUP[10:8] bits according to NVIC_PriorityGroup value */
  SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
}

/*******************************************************************************
* Function Name  : NVIC_Init
* Description    : Initializes the NVIC peripheral according to the specified
*                  parameters in the NVIC_InitStruct.
* Input          : - NVIC_InitStruct: pointer to a NVIC_InitTypeDef structure
*                    that contains the configuration information for the
*                    specified NVIC peripheral.
* Output         : None
* Return         : None
*******************************************************************************/
void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)
{
  u32 tmppriority = 0x00, tmpreg = 0x00, tmpmask = 0x00;
  u32 tmppre = 0, tmpsub = 0x0F;

  /* Check the parameters */
```

```
.text_66:
  0800D842  4770      BX          LR
}
DMA2_Channel1_IRQHandler:
DMA2_Channel1_IRQHandler:
.text_67:
  0800D844  4770      BX          LR
}
DMA2_Channel2_IRQHandler:
DMA2_Channel2_IRQHandler:
.text_68:
  0800D846  4770      BX          LR
}
DMA2_Channel3_IRQHandler:
DMA2_Channel3_IRQHandler:
.text_69:
  0800D848  4770      BX          LR
}
DMA2_Channel4_5_IRQHandler:
DMA2_Channel4_5_IRQHandler:
.text_70:
  0800D84A  4770      BX          LR
void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup)
{
NVIC_PriorityGroupConfig:
NVIC_PriorityGroupConfig:
.text_5:
  0800D84C  B510      PUSH        {R4,LR}
  0800D84E  0004      MOVS        R4, R0
  assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));
  0800D850  F5B46FE0  CMP         R4, #0x700
  0800D854  D00B      BEQ         ??NVIC_PriorityGroupConfig_0
  0800D856  F5B46FC0  CMP         R4, #0x600
  0800D85A  D008      BEQ         ??NVIC_PriorityGroupConfig_0
  0800D85C  F5B46FA0  CMP         R4, #0x500
  0800D860  D005      BEQ         ??NVIC_PriorityGroupConfig_0
  0800D862  F5B46F80  CMP         R4, #0x400
  0800D866  D002      BEQ         ??NVIC_PriorityGroupConfig_0
  0800D868  F5B47F40  CMP         R4, #0x300
  0800D86C  D100      BNE         ??NVIC_PriorityGroupConfig_1
??NVIC_PriorityGroupConfig_0:
  0800D86E  E004      B           ??NVIC_PriorityGroupConfig_2
??NVIC_PriorityGroupConfig_1:
  0800D870  2164      MOVS        R1, #0x64
  0800D872  F8DF005C  LDR.W       R0, [PC, #0x5C]
  0800D876  F7FEFCAB  BL          assert_failed
  SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
??NVIC_PriorityGroupConfig_2:
  0800D87A  F8DF0058  LDR.W       R0, [PC, #0x58]
  0800D87E  6800      LDR         R0, [R0]
```

```
Index   Frame   Address     Opcode  Trace                                 Comment
002368  002686  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002403  002721  0x0800BEBE  2800     Clk_Init() + 66
002407  002725  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002442  002760  0x0800BEBE  2800     Clk_Init() + 66
002446  002764  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002481  002799  0x0800BEBE  2800     Clk_Init() + 66
002485  002803  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002520  002838  0x0800BEBE  2800     Clk_Init() + 66
002524  002842  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002559  002877  0x0800BEBE  2800     Clk_Init() + 66
002563  002881  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002598  002916  0x0800BEBE  2800     Clk_Init() + 66
002602  002920  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002637  002955  0x0800BEBE  2800     Clk_Init() + 66
002641  002959  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002676  002994  0x0800BEBE  2800     Clk_Init() + 66
002680  002998  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002715  003033  0x0800BEBE  2800     Clk_Init() + 66
002719  003037  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002754  003072  0x0800BEBE  2800     Clk_Init() + 66
002758  003076  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002793  003111  0x0800BEBE  2800     Clk_Init() + 66
002797  003115  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002832  003150  0x0800BEBE  2800     Clk_Init() + 66
002836  003154  0x0800B5A4  B510      RCC_GetFlagStatus(u8)
002871  003189  0x0800BEBE  2800     Clk_Init() + 66
002875  003193  0x0800B3C8  B510      RCC_USBCLKConfig(u32)
002883  003201  0x0800BEC8  F44F     Clk_Init() + 76
002885  003203  0x0800B3EC  B510      RCC_ADCCLKConfig(u32)
002906  003224  0x0800BED0  2000     Clk_Init() + 84
002908  003226  0x0800B37C  B510      RCC_PCLK2Config(u32)
002923  003241  0x0800BED6  F44F     Clk_Init() + 90
002925  003243  0x0800B334  B510      RCC_PCLK1Config(u32)
002942  003260  0x0800BEDE  2000     Clk_Init() + 98
002944  003262  0x0800B2E4  B510      RCC_HCLKConfig(u32)
002959  003277  0x0800BEE4  2002     Clk_Init() + 104
002961  003279  0x0800D70C  B510      FLASH_SetLatency(u32)
002985  003303  0x0800BEEA  2000     Clk_Init() + 110
002987  003305  0x0800D746  B510      FLASH_HalfCycleAccessCmd(u32)
003009  003327  0x0800BEF0  2010     Clk_Init() + 116
003011  003329  0x0800D77C  B510      FLASH_PrefetchBufferCmd(u32)
003031  003349  0x0800BEF6  2002     Clk_Init() + 122
003033  003351  0x0800B2AC  B510      RCC_SYSCLKConfig(u32)
003053  003371  0x0800BEFC  BD01     Clk_Init() + 128
003054  003372  0x0800BFB8  2100    main() + 16
003057  003375  0x0800D88C  B538     NVIC_SetVectorTable(u32, u32)
003075  003393  0x0800BFC2  F44F    main() + 26
003077  003395  0x0800D84C  B510     NVIC_PriorityGroupConfig(u32)
003096  003414  0x0800BFCA  4876    main() + 34
```

12.3 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.



```
SEGGER J-Link Commander V3.72c ('?' 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
VTarget = 3.119V
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: 0x1B900F0F
 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: 2048x24 bit RAM
J-Link>etb
ETB is present.
ID register       (ETB[0x00]) : 1B900F0F
RAM depth         (ETB[0x01]) : 00000800
RAM width         (ETB[0x02]) : 00000018
Status            (ETB[0x03]) : 00000008
RAM data          (ETB[0x04]) : 00CBB1B7
RAM read pointer  (ETB[0x05]) : 00000000
RAM write pointer (ETB[0x06]) : 00000000
Trigger counter   (ETB[0x07]) : 00000000
Control           (ETB[0x08]) : 00000000
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.

12.4 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.

12.4.1 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.

12.4.2 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.

12.4.3 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.

12.4.4 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.

12.4.4.1 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.

12.4.4.2 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.

12.4.4.3 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.

12.4.4.4 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 knowledge of the flash programming algorithm used as well as of the target system.

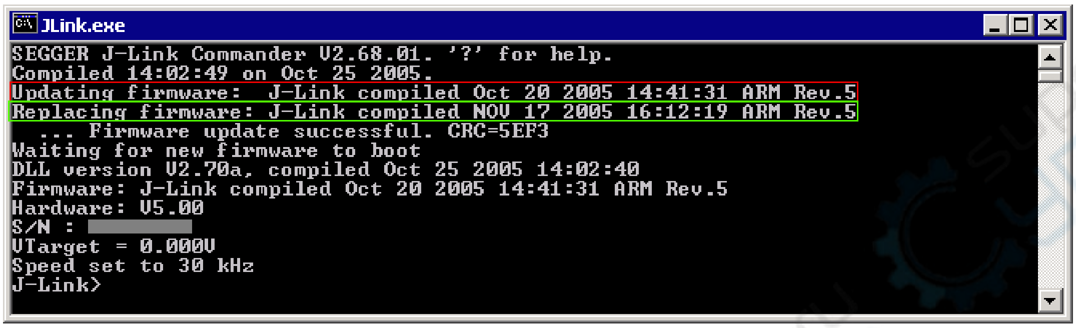
12.5 J-Link / J-Trace firmware

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.

12.5.1 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.



```
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
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
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.

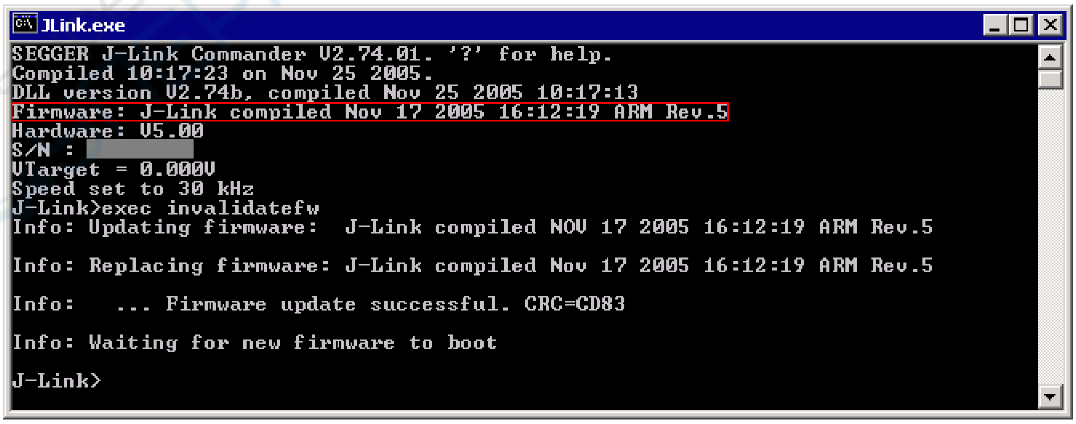
12.5.2 Invalidating the firmware

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.

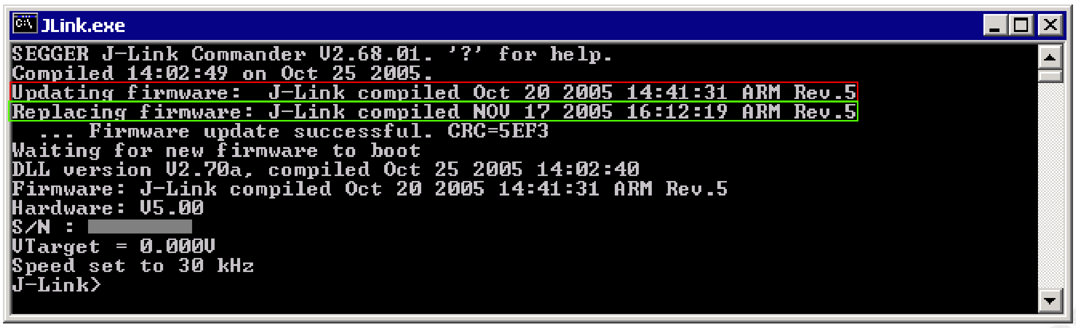
To downdate J-Link / J-Trace, you need to invalidate the current J-Link / J-Trace firmware, using the command `exec InvalidateFW`.



```
SEGGER J-Link Commander V2.74.01. '?' for help.
Compiled 10:17:23 on Nov 25 2005.
DLL version V2.74b, compiled Nov 25 2005 10:17:13
Firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>exec invalidatefw
Info: Updating firmware:  J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5

Info: Replacing firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5

Info:   ... Firmware update successful. CRC=CD83

Info: Waiting for new firmware to boot

J-Link>
```

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.



```
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
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
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 13

Designing the target board for trace

This chapter describes the hardware requirements which have to be met by the target board.

13.1 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.

13.1.1 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.

13.1.2 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.

13.1.3 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.

13.1.4 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 331 for further reference.

13.2 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.

13.2.1 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.

13.3 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 13.1: Signal requirements

Chapter 14

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.

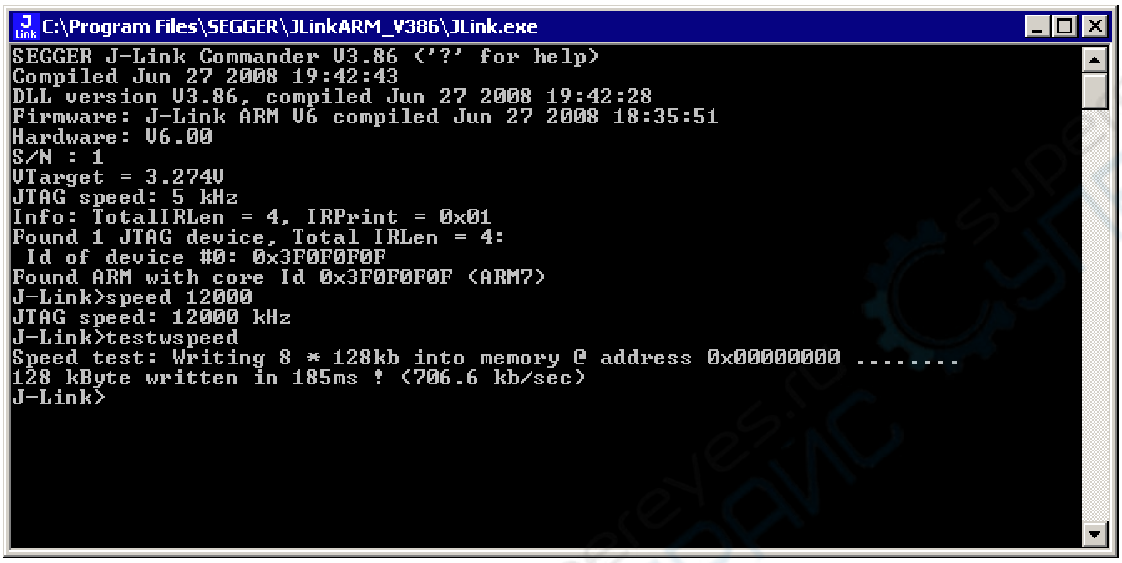
14.1 Measuring download speed

14.1.1 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 `JLink.exe` after the measurement has been performed.



```
SEGGER J-Link Commander V3.86 ('?' for help)
Compiled Jun 27 2008 19:42:43
DLL version V3.86, compiled Jun 27 2008 19:42:28
Firmware: J-Link ARM V6 compiled Jun 27 2008 18:35:51
Hardware: V6.00
S/N : 1
VTarget = 3.274V
JTAG speed: 5 kHz
Info: TotalIRLen = 4, IRPrint = 0x01
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x3F0F0F0F
Found ARM with core Id 0x3F0F0F0F (ARM7)
J-Link>speed 12000
JTAG speed: 12000 kHz
J-Link>testwspeed
Speed test: Writing 8 * 128kb into memory @ address 0x00000000 ........
128 kByte written in 185ms ! (706.6 kb/sec)
J-Link>
```

14.2 Troubleshooting

14.2.1 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 337.

14.2.2 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 121.

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 335.

14.3 Contacting support

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section *General procedure* on page 335. 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.

14.4 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 41.

Converting data files

Q: I want to download my application into flash memory using J-Link Commander but my application is a *.hex data file and J-Link Commander supports *.bin files only. How do I download it?

A: Please use the J-Flash (which is part of the J-Link software and documentation package) software to convert your *.hex/*.mot/... file to a *.bin file. For data file conversion, no J-Flash license is necessary.

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 120 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 built-in.

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 15

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 floating-point math calculations, signal processing, or memory management.

Dirty data

When referring to a processor data cache, data that has been written to the cache but has not been written to main memory 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.

TCK

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

TDI

The electronic signal input to a TAP controller from the data source (upstream). Usually, this is seen connecting the 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 16

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 (*www.arm.com*).

Table 16.1: Literature and References

Index