

# IAR J-Link and IAR J-Trace

User Guide

JTAG Emulators for  
**ARM Cores**



J-Link/J-TraceARM-5

 IAR  
SYSTEMS

## **COPYRIGHT NOTICE**

© 2006-2011 IAR Systems AB.

No part of this document may be reproduced without the prior written consent of IAR Systems AB. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

## **DISCLAIMER**

The information in this document is subject to change without notice and does not represent a commitment on any part of IAR Systems. While the information contained herein is assumed to be accurate, IAR Systems assumes no responsibility for any errors or omissions.

In no event shall IAR Systems, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

## **TRADEMARKS**

IAR Systems, IAR Embedded Workbench, C-SPY, visualSTATE, The Code to Success, IAR KickStart Kit, IAR, and the logotype of IAR Systems are trademarks or registered trademarks owned by IAR Systems AB. J-Link and J-Trace are trademarks licensed to IAR Systems AB.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

Intel and Pentium are registered trademarks and XScale a trademark of Intel Corporation.

ARM and Thumb are registered trademarks of Advanced RISC Machines Ltd.

All other product names are trademarks or registered trademarks of their respective owners.

## **EDITION NOTICE**

Fifth edition: October 2011

Part number: J-Link/J-TraceARM-5

Internal reference: V4.36, IMAE

# Preface

Welcome to the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores.

---

## About this guide

This guide provides an overview over the major features of J-Link and J-Trace, gives you some background information about JTAG, ARM and Tracing in general and describes J-Link and J-Trace related software packages. Finally, the chapter *Support and FAQs*, page 135 helps to troubleshoot common problems.

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

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

## TYPOGRAPHIC CONVENTIONS

This manual uses the following typographic conventions:

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

Table 1: Typographic conventions

---

## Literature and references

To gain deeper understanding of technical details, see:

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 ( <a href="http://www.arm.com">www.arm.com</a> ).

Table 2: Literature and references



# Table of Contents

Preface .....	3
<b>About this guide</b> .....	3
Typographic conventions .....	3
<b>Literature and references</b> .....	3
Introduction .....	11
<b>Requirements</b> .....	11
<b>Supported OS</b> .....	11
<b>J-Link / J-Trace models</b> .....	11
Model comparison .....	12
J-Link ARM .....	12
J-Link Ultra .....	15
J-Link ARM Lite .....	16
J-Link Lite Cortex-M .....	17
J-Trace ARM .....	18
J-Trace for Cortex-M3 .....	19
<b>Common features of the J-Link product family</b> .....	20
<b>Supported CPU cores</b> .....	21
<b>Built-in intelligence for supported CPU-cores</b> .....	22
Intelligence in the J-Link firmware .....	22
Intelligence on the PC-side (DLL) .....	22
Firmware intelligence per model .....	23
Licensing .....	25
<b>Introduction</b> .....	25
<b>Software components requiring a license</b> .....	25
<b>License types</b> .....	25
Built-in license .....	26
Key-based license .....	26
Device-based license .....	27
<b>Legal use of SEGGER J-Link software</b> .....	29
<b>Products</b> .....	30
J-Link .....	30
J-Link Ultra .....	30
J-Trace .....	31
J-Trace for Cortex-M .....	31
<b>IAR J-Link Lite</b> .....	31
<b>J-Link OBs</b> .....	31
<b>Illegal Clones</b> .....	32
J-Link and J-Trace related software .....	33
<b>J-Link related software</b> .....	33
J-Link software and documentation package .....	33

<b>J-Link software and documentation package in detail</b> .....	33
J-Link Commander (Command line tool) .....	34
J-Link STR91x Commander (Command line tool) .....	34
J-Link STM32 Commander (Command line tool) .....	36
J-Mem Memory Viewer .....	36
J-Flash ARM (Program flash memory via JTAG) .....	36
<b>Using the J-LinkARM.dll</b> .....	37
What is the JLinkARM.dll? .....	37
Updating the DLL .....	37
Determining the version of JLinkARM.dll .....	38
Determining which DLL is used by a program .....	39
<b>Setup</b> .....	41
<b>Installing the J-Link ARM</b> .....	41
Setup procedure .....	41
<b>Setting up the USB interface</b> .....	41
Verifying correct driver installation .....	41
Uninstalling the J-Link USB driver .....	43
<b>J-Link USB identification</b> .....	44
Connecting to different J-Links connected to the same host PC via USB .....	45
<b>Working with J-Link and J-Trace</b> .....	47
<b>Connecting the target system</b> .....	47
Power-on sequence .....	47
Verifying target device connection .....	47
Problems .....	47
<b>Indicators</b> .....	47
Main indicator .....	47
Input indicator .....	48
Output indicator .....	49
<b>JTAG interface</b> .....	49
Multiple devices in the scan chain .....	50
Configuration dialog boxes .....	50
Determining values for scan chain configuration .....	52
JTAG Speed .....	53
<b>SWD interface</b> .....	53
SWD speed .....	53
SWO .....	53
<b>Multi-core debugging</b> .....	54
How multi-core debugging works .....	55
Using multi-core debugging in detail .....	56
Things you should be aware of .....	57
<b>Connecting multiple J-Links / J-Traces to your PC</b> .....	58
How does it work? .....	58
<b>J-Link control panel</b> .....	59
Tabs .....	59
<b>Reset strategies</b> .....	64
Strategies for ARM 7/9 devices .....	64
Strategies for Cortex-M devices .....	66
<b>Using DCC for memory access</b> .....	67
What is required? .....	68

Target DCC handler .....	68
Target DCC abort handler .....	68
<b>J-Link script files</b> .....	68
Actions that can be customized .....	68
Script file API functions .....	69
Global DLL variables .....	72
Global DLL constants .....	73
Script file language .....	74
Executing J-Link script files .....	75
<b>Command strings</b> .....	75
List of available commands .....	76
Using command strings .....	81
<b>Switching off CPU clock during debug</b> .....	82
<b>Cache handling</b> .....	82
Cache coherency .....	82
Cache clean area .....	83
Cache handling of ARM7 cores .....	83
Cache handling of ARM9 cores .....	83
Flash download .....	85
<b>Introduction</b> .....	85
<b>Licensing</b> .....	85
<b>Supported devices</b> .....	85
<b>Setup</b> .....	86
IAR Embedded Workbench .....	86
J-Link Commander .....	86
<b>Setup for CFI flash</b> .....	87
IAR Embedded Workbench .....	87
J-Link commander .....	89
<b>Using the DLL flash loaders in custom applications</b> .....	89
Flash breakpoints .....	91
<b>Introduction</b> .....	91
<b>Licensing</b> .....	91
24h flash breakpoint trial license .....	92
<b>Supported devices</b> .....	92
<b>Setup</b> .....	93
Setup .....	93
Device specifics .....	95
<b>Analog Devices</b> .....	95
ADuC7xxx .....	95
<b>ATMEL</b> .....	96
AT91SAM7 .....	97
AT91SAM9 .....	99
<b>DSPGroup</b> .....	99
<b>Ember</b> .....	99
<b>Energy Micro</b> .....	100
<b>Freescale</b> .....	101
Kinetic family .....	102
Unlocking .....	102
Tracing .....	102

<b>Fujitsu</b> .....	102
<b>Itron</b> .....	103
<b>Luminary Micro</b> .....	103
Unlocking LM3Sxxx devices .....	104
<b>NXP</b> .....	104
LPC ARM7-based devices .....	105
Reset (Cortex-M3 based devices) .....	106
<b>OKI</b> .....	106
<b>Renesas</b> .....	107
<b>Samsung</b> .....	107
S3FN60D .....	107
<b>ST Microelectronics</b> .....	107
STR91x .....	108
STM32F10x .....	109
<b>Texas Instruments</b> .....	110
<b>Toshiba</b> .....	110
<b>Target interfaces and adapters</b> .....	113
<b>20-pin JTAG/SWD connector</b> .....	113
Pinout for JTAG .....	113
Pinout for SWD .....	115
<b>38-pin Mictor JTAG and Trace connector</b> .....	116
Connecting the target board .....	117
Pinout .....	118
Assignment of trace information pins between ETM architecture versions .....	119
Trace signals .....	119
<b>19-pin JTAG/SWD and Trace connector</b> .....	120
Target power supply .....	121
<b>9-pin JTAG/SWD connector</b> .....	121
<b>Adapters</b> .....	122
<b>Background information</b> .....	123
<b>JTAG</b> .....	123
Test access port (TAP) .....	123
Data registers .....	123
Instruction register .....	123
The TAP controller .....	124
<b>Embedded Trace Macrocell (ETM)</b> .....	125
Trigger condition .....	125
Code tracing and data tracing .....	125
J-Trace integration example - IAR Embedded Workbench for ARM .....	126
<b>Embedded Trace Buffer (ETB)</b> .....	130
<b>Flash programming</b> .....	130
How does flash programming via J-Link / J-Trace work? .....	130
Data download to RAM .....	130
Data download via DCC .....	131
Available options for flash programming .....	131
<b>J-Link / J-Trace firmware</b> .....	131
Firmware update .....	131
Invalidating the firmware .....	131



Designing the target board for trace .....	133
<b>Overview of high-speed board design</b> .....	133
Avoiding stubs .....	133
Minimizing Signal Skew (Balancing PCB Track Lengths) .....	133
Minimizing Crosstalk .....	133
Using impedance matching and termination .....	133
<b>Terminating the trace signal</b> .....	133
Rules for series terminators .....	134
<b>Signal requirements</b> .....	134
Support and FAQs .....	135
<b>Measuring download speed</b> .....	135
Test environment .....	135
<b>Troubleshooting</b> .....	135
General procedure .....	135
Typical problem scenarios .....	136
<b>Signal analysis</b> .....	136
Start sequence .....	137
Troubleshooting .....	137
<b>Contacting support</b> .....	137
<b>Frequently Asked Questions</b> .....	138
Glossary .....	139
Literature and references .....	143
Index .....	145



# Introduction

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

---

## Requirements

### Host System

To use J-Link or J-Trace you need a host system running Windows XP or later. For a list of all operating systems which are supported by J-Link, please refer to *Supported OS* on page 11.

### 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 12.

---

## Supported OS

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

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

---

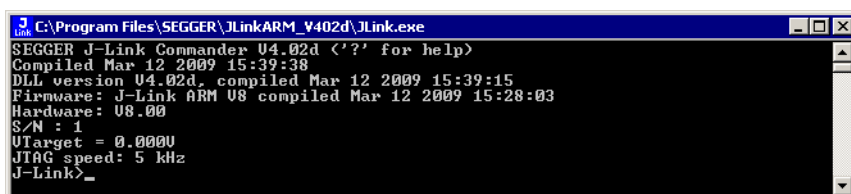
## J-Link / J-Trace models

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

- J-Link ARM
- J-Link Ultra
- J-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`, included in the `arm\bin` directory of your IAR Embedded Workbench installation. As part of the initial message, the hardware version is displayed.



```
C:\Program Files\SEGGER\JLinkARM_V402d\JLink.exe
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
Utarget = 0.0000
JTAG speed: 5 kHz
J-Link>_
```

## MODEL COMPARISON

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

### Hardware features

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

	J-Link	J-Trace for Cortex-M	J-Trace
Flash breakpoints <sup>2</sup>	yes(opt)	yes(opt)	yes(opt)
Flash download <sup>1</sup>	yes(opt)	yes(opt)	yes(opt)

<sup>1</sup> IAR Embedded Workbench comes with its own flashloaders for most targets, so in most cases this feature is not essential for debugging your applications in flash. For more information about how flash download via FlashDL works, refer to *Flash download* on page 85.

<sup>2</sup> 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, refer to *Flash breakpoints* on page 91.

## J-LINK ARM

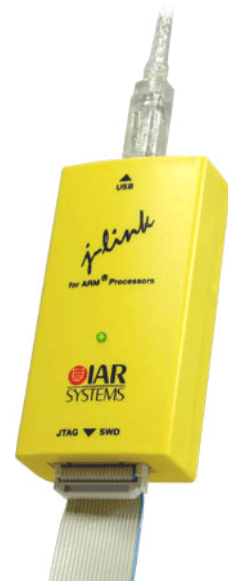
J-Link is a JTAG emulator designed for ARM cores. It connects via USB to a PC running Microsoft Windows XP or later. For a complete list of all operating systems which are supported, please refer to *Supported OS* on page 11. J-Link has a built-in 20-pin JTAG connector, which is compatible with the standard 20-pin connector defined by ARM.

### Additional features

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

\* = Supported since J-Link hardware version 6

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



## Specifications

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

General	
Supported OS	For a complete list of all operating systems which are supported, please refer to <i>Supported OS</i> on page 11.
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
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 ( $V_{OL}$ )	$V_{OL} \leq 10\%$ of $V_{IF}$
<b>For the whole target voltage range (<math>1.2V \leq V_{IF} \leq 5V</math>)</b>	
LOW level input voltage ( $V_{IL}$ )	$V_{IL} \leq 40\%$ of $V_{IF}$
HIGH level input voltage ( $V_{IH}$ )	$V_{IH} \geq 60\%$ of $V_{IF}$
<b>For <math>1.8V \leq V_{IF} \leq 3.6V</math></b>	
LOW level output voltage ( $V_{OL}$ ) with a load of 10 kOhm	$V_{OL} \leq 10\%$ of $V_{IF}$
HIGH level output voltage ( $V_{OH}$ ) with a load of 10 kOhm	$V_{OH} \geq 90\%$ of $V_{IF}$
<b>For <math>3.6 \leq V_{IF} \leq 5V</math></b>	
LOW level output voltage ( $V_{OL}$ ) with a load of 10 kOhm	$V_{OL} \leq 20\%$ of $V_{IF}$
HIGH level output voltage ( $V_{OH}$ ) with a load of 10 kOhm	$V_{OH} \geq 80\%$ of $V_{IF}$
JTAG/SWD Interface, Timing	
SWO sampling frequency	Max. 6 MHz
Data input rise time ( $T_{rdi}$ )	$T_{rdi} \leq 20ns$
Data input fall time ( $T_{fdi}$ )	$T_{fdi} \leq 20ns$
Data output rise time ( $T_{rdo}$ )	$T_{rdo} \leq 10ns$
Data output fall time ( $T_{fdo}$ )	$T_{fdo} \leq 10ns$
Clock rise time ( $T_{rc}$ )	$T_{rc} \leq 10ns$
Clock fall time ( $T_{fc}$ )	$T_{fc} \leq 10ns$

Table 1: J-Link ARM specifications

## Download speed

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

Hardware	ARM7 via JTAG	ARM9 via JTAG	Cortex-M3 via SWD
J-Link Rev. 6 — 8	720 Kbytes/s (12MHz JTAG)	550 Kbytes/s (12MHz JTAG)	180 Kbytes/s (12 MHz SWD)

Table 2: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 135.

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

## Hardware versions

### Versions 1-4

Obsolete.

### Version 5.0

Identical to version 4.0 with the following exception:

- Uses a 32-bit RISC CPU.
- Maximum download speed (using DCC) is over 700 Kbytes/second.
- JTAG speed: Maximum JTAG frequency is 12 MHz; possible JTAG speeds are: 48 MHz / n, where n is 4, 5, ..., resulting in speeds of:
  - 12.000 MHz (n = 4)
  - 9.600 MHz (n = 5)
  - 8.000 MHz (n = 6)
  - 6.857 MHz (n = 7)
  - 6.000 MHz (n = 8)
  - 5.333 MHz (n = 9)
  - 4.800 MHz (n = 10)
- Supports adaptive clocking.

### Version 5.2

Identical to version 5.0 with the following exception:

- Target interface: RESET is open drain

### Version 5.3

Identical to version 5.2 with the following exception:

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

### Version 5.4

Identical to version 5.3 with the following exception:

- Supports 5V target interfaces.

### Version 6.0

Identical to version 5.4 with the following exception:

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

### Version 7.0

Identical to version 6.0 with the following exception:

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

### Version 8.0

Identical to version 7.0 with the following exception:

- SWD support for non-3.3V targets.

## 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 XP 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.



### Additional features

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

### Specifications

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

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

General	
Supported OS	For a complete list of all operating systems which are supported, please refer to <i>Supported OS</i> on page 11.
Electromagnetic compatibility (EMC)	EN 55022, EN 55024
Operating temperature	+5°C ... +60°C
Storage temperature	-20°C ... +65 °C
Relative humidity (non-condensing)	Max. 90% rH
Mechanical	
Size (without cables)	100mm x 53mm x 27mm
Weight (without cables)	73g
Available interfaces	
USB interface	USB 2.0, Hi-Speed
Target interface	JTAG/SWD 20-pin
External (SPI) analog power measurement interface	4-pin (Pins 14, 16, 18 and 20 of the 20-pin JTAG/SWD interface)
JTAG/SWD Interface, Electrical	
Target interface voltage ( $V_{IF}$ )	1.8V ... 5V
Target supply voltage	4.5V ... 5V
Target supply current	Max. 300mA
Reset Type	Open drain. Can be pulled low or tristated.
Reset low level output voltage ( $V_{OL}$ )	$V_{OL} \leq 10\%$ of $V_{IF}$
<b>For the whole target voltage range (<math>1.8V \leq V_{IF} \leq 5V</math>)</b>	

Table 3: J-Link Ultra specifications

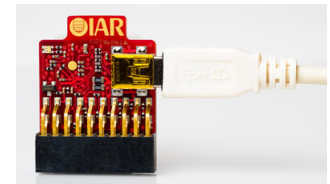
LOW level input voltage ( $V_{IL}$ )	$V_{IL} \leq 40\%$ of $V_{IF}$
HIGH level input voltage ( $V_{IH}$ )	$V_{IH} \geq 60\%$ of $V_{IF}$
<b>For 1.8V <math>\leq V_{IF} \leq 3.6</math>V</b>	
LOW level output voltage ( $V_{OL}$ ) with a load of 10 kOhm	$V_{OL} \leq 10\%$ of $V_{IF}$
HIGH level output voltage ( $V_{OH}$ ) with a load of 10 kOhm	$V_{OH} \geq 90\%$ of $V_{IF}$
<b>For 3.6 <math>\leq V_{IF} \leq 5</math>V</b>	
LOW level output voltage ( $V_{OL}$ ) with a load of 10 kOhm	$V_{OL} \leq 20\%$ of $V_{IF}$
HIGH level output voltage ( $V_{OH}$ ) with a load of 10 kOhm	$V_{OH} \geq 80\%$ of $V_{IF}$

<b>JTAG/SWD Interface, Timing</b>	
SWO sampling frequency	Max. 25 MHz
Data input rise time ( $T_{rdi}$ )	$T_{rdi} \leq 20$ ns
Data input fall time ( $T_{fdi}$ )	$T_{fdi} \leq 20$ ns
Data output rise time ( $T_{rdo}$ )	$T_{rdo} \leq 10$ ns
Data output fall time ( $T_{fdo}$ )	$T_{fdo} \leq 10$ ns
Clock rise time ( $T_{rc}$ )	$T_{rc} \leq 10$ ns
Clock fall time ( $T_{fc}$ )	$T_{fc} \leq 10$ ns
<b>Analog power measurement interface</b>	
Sampling frequency	50 kHz
Resolution	1 mA
<b>External (SPI) analog interface</b>	
SPI frequency	Max. 4 MHz
Samples/sec	Max. 50000
Resolution	Max. 16-bit

Table 3: J-Link Ultra specifications

## J-LINK ARM LITE

J-Link ARM Lite is a fully functional version of J-Link ARM debug probe, which is available with an IAR KickStart Kit only.



### Additional features

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

### Specifications

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

<b>General</b>	
Supported OS	For a complete list of all operating systems which are supported, please refer to <i>Supported OS</i> on page 11.
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

Table 4: J-Link ARM Lite specifications



Weight (without cables)	6g
<b>Mechanical</b>	
USB interface	USB 2.0, full speed
Target interface	JTAG 20-pin
<b>JTAG/SWD Interface, Electrical</b>	
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 $V_{IF}$
<b>JTAG/SWD Interface, Timing</b>	
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 4: J-Link ARM Lite specifications

## J-LINK LITE CORTEX-M

J-Link Lite Cortex-M is a specific version of J-Link Lite which is designed to be used with Cortex-M devices.

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



## Specifications

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

<b>General</b>	
Supported OS	For a complete list of all operating systems which are supported, please refer to <i>Supported OS</i> on page 11.
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
<b>Mechanical</b>	
USB interface	USB 2.0, full speed
Target interface	19-pin 0.05" Samtec FTSH connector 9-pin 0.05" Samtec FTSH connector

Table 5: J-Link Lite Cortex-M specifications

JTAG/SWD Interface, Electrical	
Power supply	USB powered Max. 50mA + Target Supply current.
Target interface voltage ( $V_{IF}$ )	3.3V
Target supply voltage	4.5V ... 5V
Target supply current	Max. 300mA
LOW level input voltage ( $V_{IL}$ )	Max. 40% of $V_{IF}$
HIGH level input voltage ( $V_{IH}$ )	Min. 60% of $V_{IF}$
JTAG/SWD Interface, Timing	
Data input rise time ( $T_{rdi}$ )	Max. 20ns
Data input fall time ( $T_{fdi}$ )	Max. 20ns
Data output rise time ( $T_{rdo}$ )	Max. 10ns
Data output fall time ( $T_{fdo}$ )	Max. 10ns
Clock rise time ( $T_{rc}$ )	Max. 10ns
Clock fall time ( $T_{fc}$ )	Max. 10ns

Table 5: J-Link Lite Cortex-M specifications

## 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 XP, Windows 2003, Windows Vista or Windows 7. 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.

### 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.



### Specifications for J-Trace

General	
Supported OS	For a complete list of all operating systems which are supported, please refer to <i>Supported OS</i> on page 11.
Electromagnetic Compatibility (EMC)	EN 55022, EN 55024
Operating Temperature	+5°C ... +40°C
Storage Temperature	-20°C ... +65 °C
Relative Humidity (non-condensing)	<90% rH
Size (without cables)	123mm x 68mm x 30mm
Weight (without cables)	120g
Mechanical	
USB Interface	USB 2.0, full speed
Target Interface	JTAG 20-pin (14-pin adapter available) JTAG+Trace: Mictor, 38-pin
JTAG/SWD Interface, Electrical	
Power Supply	USB powered < 300mA
Supported Target interface voltage	3.0 - 3.6 V (5V adapter available)

Table 6: J-Trace specifications

## 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 7: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 135.

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

## 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).

## J-TRACE FOR CORTEX-M3

J-Trace for Cortex-M is a JTAG/SWD emulator designed for Cortex-M3 cores which includes trace (ETM) support. J-Trace for Cortex-M3 can also be used as a J-Link and it also supports ARM7/9 cores. Tracing on ARM7/9 targets is not supported.



### Additional features

- Has all the J-Link functionality
- Supports tracing on Cortex-M3 targets

### Specifications

The following table gives an overview about the specifications (general, mechanical, electrical) for J-Trace for Cortex-M3. All values are valid for the latest hardware version of J-Trace for Cortex-M3.

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 $V_{IF}$

Table 8: J-Trace for Cortex-M3 specifications

HIGH level input voltage ( $V_{IH}$ )	Min. 60% of $V_{IF}$
<b>JTAG/SWD Interface, Timing</b>	
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
<b>Trace Interface, Electrical</b>	
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 $V_{IF}$
<b>Trace Interface, Timing</b>	
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 8: J-Trace for Cortex-M3 specifications

## 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 9: Download speed differences between hardware revisions

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

## 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

## 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
- 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

---

## 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

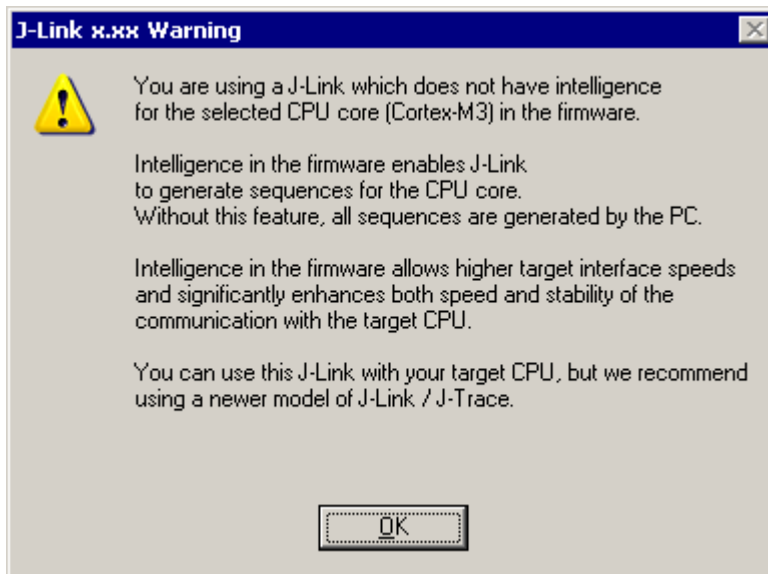
---

## Built-in intelligence for supported CPU-cores

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

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

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



### 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.

### 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!

## 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.

## 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.

### 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	ARM	Cortex-A/R	Cortex-M		Renesas
		7/9	II	JTAG	JTAG	SWD	RX600
		JTAG	JTAG	JTAG	JTAG	SWD	JTAG
J-Link	8	✓	✓	✓	✓	✓	✓
J-Link Ultra	1	✓	✓	✓	✓	✓	✓
J-Link Lite	8	✓	✓	✓	✓	✓	✓
J-Link Lite Cortex-M	8	✗	✗	✗	✓	✓	✗
J-Trace ARM	1	✓	✗	✗	✗	✗	✗
J-Trace for Cortex-M	3	✗	✗	✗	✓	✓	✗

Table 10: Built-in intelligence of current J-Links

### 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	ARM	Cortex-A/R	Cortex-M		Renesas
		7/9	II	JTAG	JTAG	SWD	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-Trace for Cortex-M	1	✗	✗	✓	✓	✓	✗

Table 11: Built-in intelligence of older J-Link models





# Licensing

This chapter describes the different license types of J-Link related software and the legal use of the J-Link software.

---

## Introduction

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

---

## Software components requiring a license

There are different software components which need an additional license:

- Flash breakpoints (FlashBP)

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

---

## License types

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

### Built-in License

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

### Key-based license

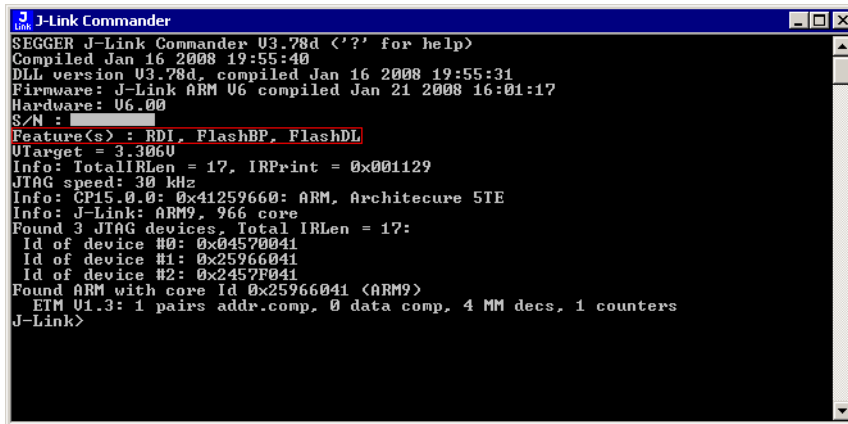
This type of license is used if you already have a J-Link, but want to enhance its functionality by using flash breakpoints. In addition to that, the key-based license is used for trial licenses. To enable this type of license you need to obtain a license key. This license key has to be added to the J-Link license management. How to enter a license key is described in detail in *Licensing* on page 91. Every license can be used on different PCs, but only with the J-Link the license is for. This means that if you want to use flash breakpoints with other J-Links, every J-Link needs a license.

### Device-based license

The device-based license comes with the J-Link software and is available for some devices. For a complete list of devices which have built-in licenses, please refer to *Device list* on page 28. The device-based license has to be activated via the debugger. How to activate a device-based license is described in detail in the section *Activating a device-based license* on page 27.

## 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.



```
SEGGGER J-Link Commander U3.78d <'?' for help>
Compiled Jan 16 2008 19:55:40
DLL version U3.78d, compiled Jan 16 2008 19:55:31
Firmware: J-Link ARM U6 compiled Jan 21 2008 16:01:17
Hardware: U6.00
S/N:
Feature(s) : RDI, FlashBP, FlashDL
Utarget = 3.306U
Info: TotalIRLen = 17, IRPrint = 0x001129
JTAG speed: 30 kHz
Info: CP15.0.0: 0x41259660: ARM, Architecture 5TE
Info: J-Link: ARM9, 966 core
Found 3 JTAG devices, Total IRLen = 17:
Id of device #0: 0x04570041
Id of device #1: 0x25966041
Id of device #2: 0x2457F041
Found ARM with core Id 0x25966041 <ARM9>
  ETM U1.3: 1 pairs addr.comp, 0 data comp, 4 MM decs, 1 counters
J-Link>
```

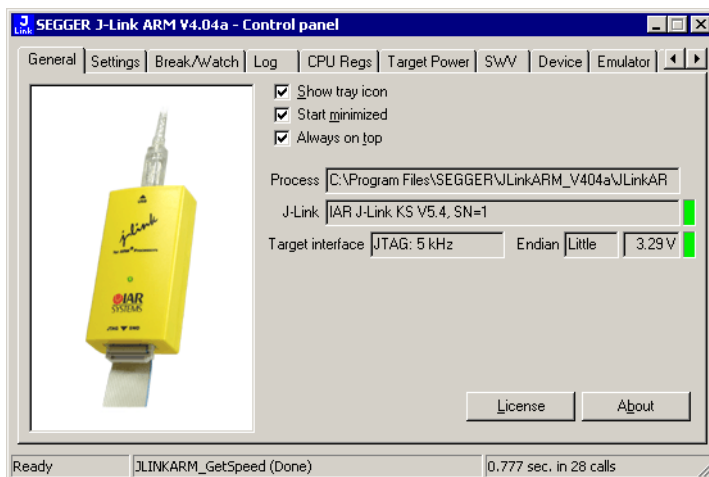
## KEY-BASED LICENSE

When using a key-based license, a license key is required in order to enable the J-Link flash breakpoint feature. License keys can be added via the license manager. How to enter a license via the license manager is described in *Licensing* on page 91. 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.

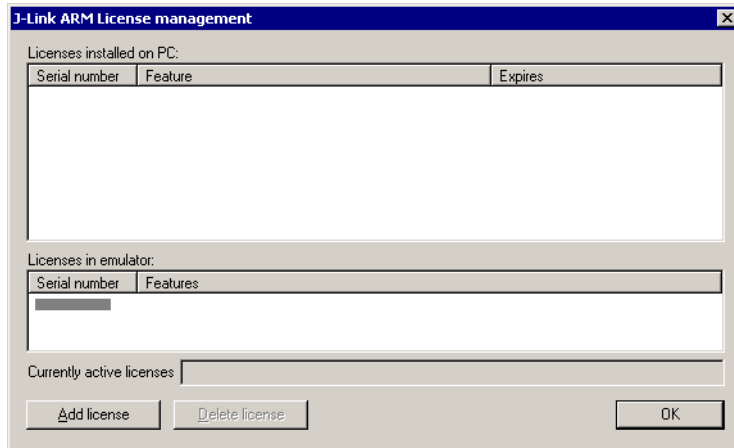
### Entering a key-based license

The easiest way to enter a license is the following:

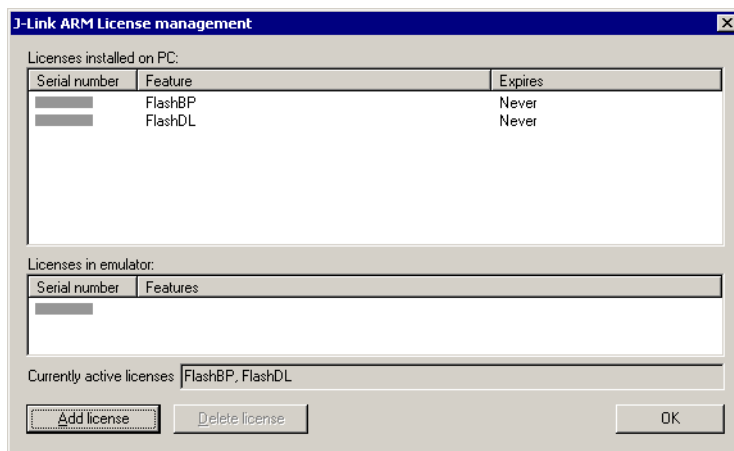
Open the J-Link control panel window, go to the **General** tab and choose **License**.



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



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

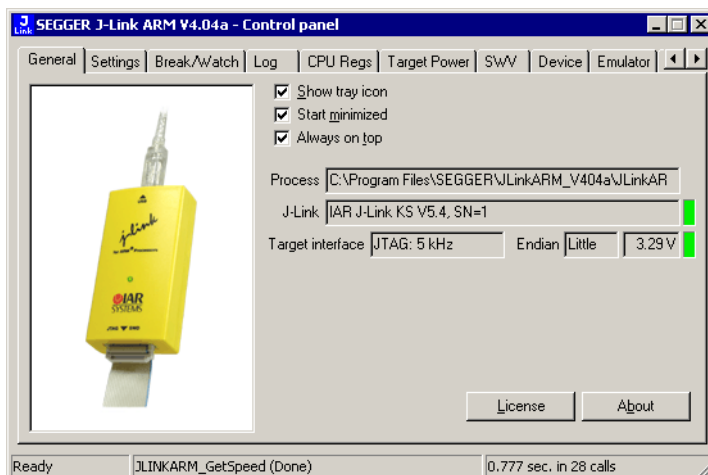


## DEVICE-BASED LICENSE

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

### Activating a device-based license

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



## Device list

The following list contains all devices which are supported by the device-based license

Manufacturer	Name	Licenses
NXP	LPC2101	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2102	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2103	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2104	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2105	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2106,	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2109	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2114	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2119	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2124	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2129	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2131	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2132	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2134	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2136	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2138	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2141	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2142	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2144	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2146	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2148	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2194	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2212	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2214	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2292	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2294	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2364	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2366	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2368	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2378	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2468	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2478	J-Link ARM FlashDL, J-Link ARM FlashBP

Table 1: Device list

Manufacturer	Name	Licenses
NXP	LPC2101	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2102	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2103	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2104	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2105	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2106	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2109	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2114	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2119	J-Link ARM FlashDL, J-Link ARM FlashBP

Table 2: Device list

Manufacturer	Name	Licenses
NXP	LPC2124	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2129	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2131	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2132	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2134	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2136	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2138	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2141	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2142	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2144	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2146	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2148	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2194	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2212	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2214	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2292	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2294	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2364	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2366	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2368	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2378	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2468	J-Link ARM FlashDL, J-Link ARM FlashBP
NXP	LPC2478	J-Link ARM FlashDL, J-Link ARM FlashBP

Table 2: Device list

## 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

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

---

## Products

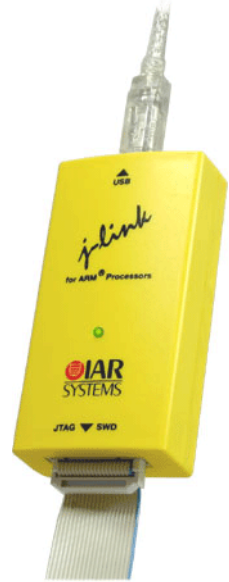
The following products are original products for which the use of the J-Link software is allowed:

### J-LINK

J-Link is a JTAG emulator designed for ARM cores. It connects via USB to a PC running Microsoft 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 and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 28.



### 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 XP, Windows 2003, Windows Vista or Windows 7.

J-Link Ultra has a built-in 20-pin JTAG/SWD connector.



#### Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 28.

## 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 XP, Windows 2003, Windows Vista or Windows 7. J-Trace has a built-in 20-pin JTAG connector and a built in 38-pin JTAG+Trace connector, which is compatible with the standard 20-pin connector and 38-pin connector defined by ARM.

### Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 28.



## J-TRACE FOR CORTEX-M

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

### Licenses

Comes with built-in licenses for flash download and flash breakpoints for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 28.




---

## IAR J-Link Lite

IAR J-Link Lite is an OEM version of J-Link.

### Limitations

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.




---

## J-Link OBs

- J-Link OBs (J-Link On Board) are single chip versions of J-Link which are used on various evaluation boards.

---

## Illegal Clones

Clones are copies of original products which use the copyrighted original firmware without a license. It is strictly prohibited to use original J-Link software with illegal clones. 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 original J-Link software, please get in touch with us.

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



# J-Link and J-Trace related software

This chapter describes J-Link / J-Trace related software.

---

## J-Link related software

### J-LINK SOFTWARE AND DOCUMENTATION PACKAGE

J-Link is shipped with a bundle of applications. Some of the applications require an additional license.

Software	Description
JLinkARM.dll	DLL for using J-Link / J-Trace with third-party programs.
JLink.exe	Free command-line tool with basic functionality for target analysis.
JLinkSTR91x	Free command-line tool to configure the ST STR91x cores. For more information please refer to <i>J-Link STR91x Commander (Command line tool)</i> on page 34
JLinkSTM32	Free command-line tool for STM32 devices. Can be used to disable the hardware watchdog and to unsecure STM32 devices (override read-protection).
J-Mem memory viewer	Free target memory viewer. Shows the memory content of a running target and allows editing as well.
J-Flash	Stand-alone flash programming application. Requires an additional license. For more information about J-Flash please refer to <i>J-Flash ARM User's Guide (UM08003)</i> .

---

Table 1: J-Link / J-Trace related software

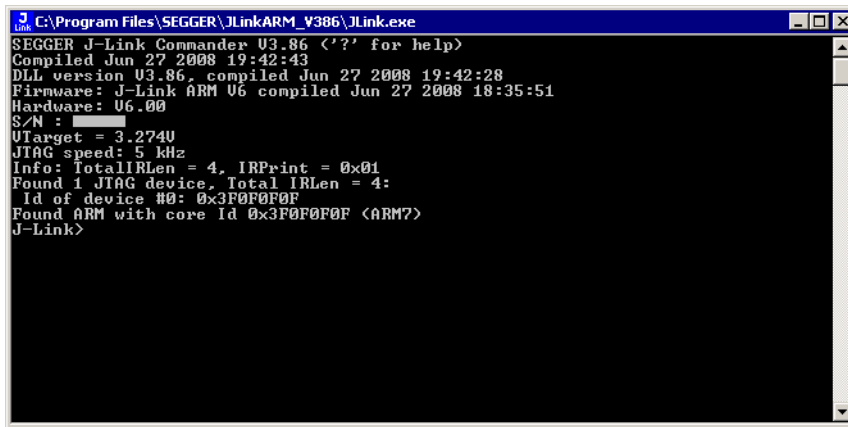
---

## J-Link software and documentation package in detail

The J-Link / J-Trace software documentation is supplied with IAR Embedded Workbench.

## J-LINK COMMANDER (COMMAND LINE TOOL)

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



```
C:\Program Files\SEGGER\JLinkARM_V386\JLink.exe
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 : 
UTarget = 3.274U
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>
```

### Using command script files

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

#### Example

```
JLink.exe C:\script.jlink
```

Contents of `script.jlink`:

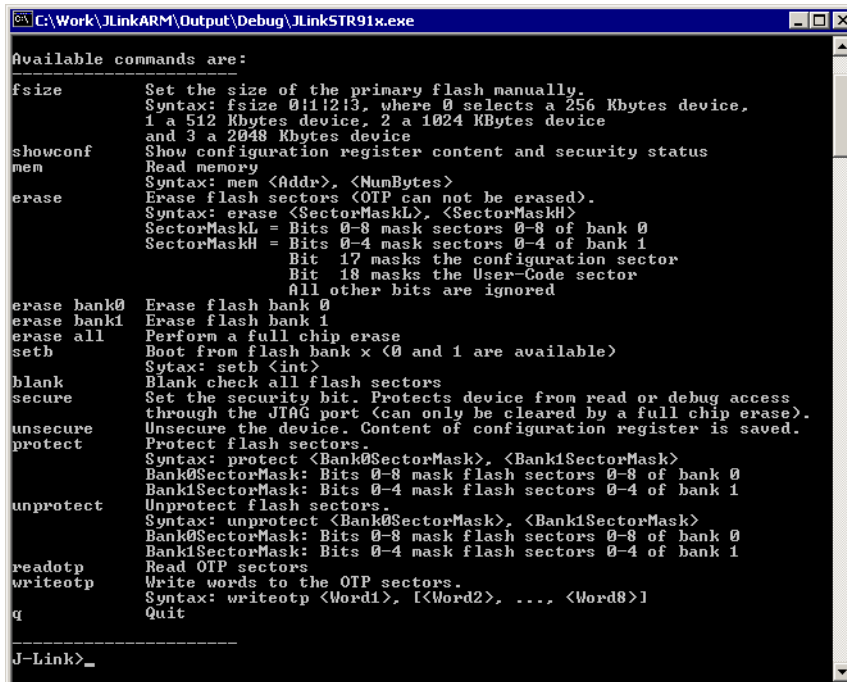
```
r
h
exec device = STM32F103ZE
loadbin C:\firmware.bin,0x08000000
```

## J-LINK STR91X COMMANDER (COMMAND LINE TOOL)

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

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

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



```

C:\Work\JLinkARM\Output\Debug\JLinkSTR91x.exe
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
seth         Boot from flash bank x (<0 and 1 are available>)
               Syntax: seth <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.

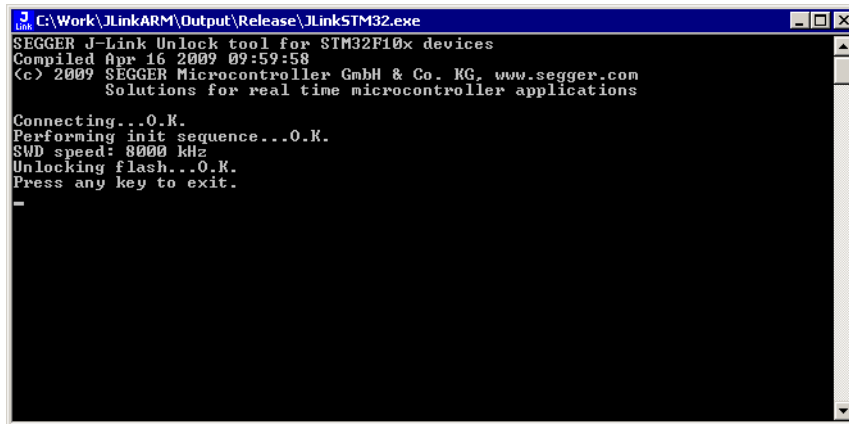
"While enabling the Turbo Mode, a dedicated test mode signal is set and controls the GPIOs in output. The IOs are maintained in this state until a next JTAG instruction is send." (ST Microelectronics)

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

## J-LINK STM32 COMMANDER (COMMAND LINE TOOL)

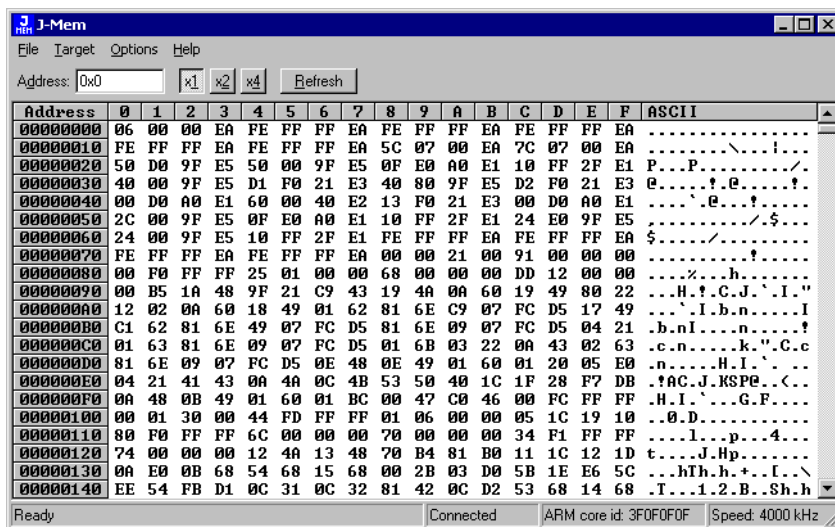
J-Link STM32 Commander (JLinkSTM32.exe) is a free command line tool which can be used to disable the hardware watchdog of STM32 devices which can be activated by programming the option bytes. Moreover the J-Link STM32 Commander unsecures a read-protected STM32 device by re-programming the option bytes.

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



## J-MEM MEMORY VIEWER

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

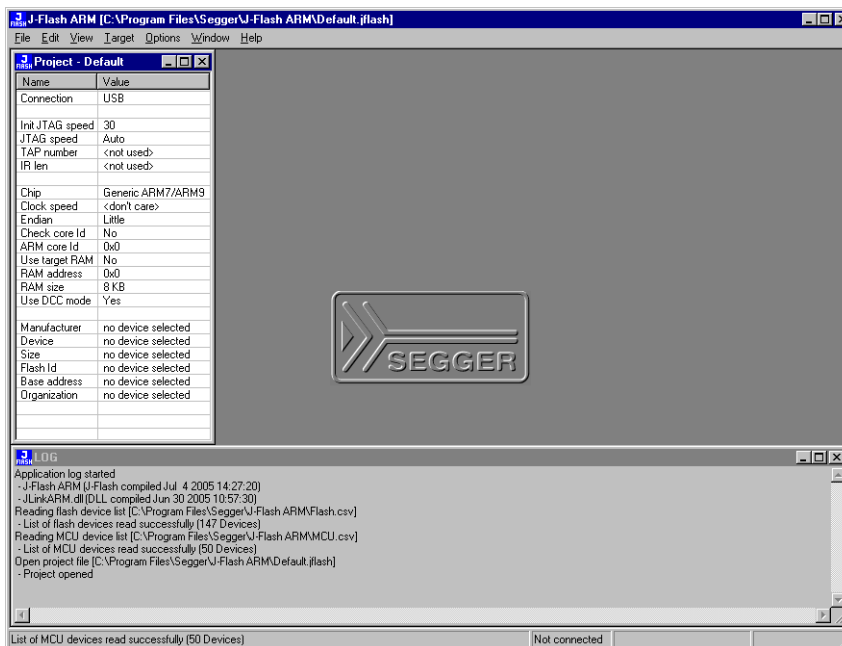


## J-FLASH ARM (PROGRAM FLASH MEMORY VIA JTAG)

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

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

J-Flash requires a additional license. Even without a license key you can still use J-Flash ARM to open project files, read from connected devices, blank check target memory, verify data files and so on. However, to actually program devices via J-Flash ARM and J-Link / J-Trace you are required to obtain a license key.



## Features

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

## Using the J-LinkARM.dll

### WHAT IS THE JLINKARM.DLL?

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

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

### UPDATING THE DLL

The IAR C-SPY<sup>®</sup> debugger is shipped with the `JLinkARM.dll` already installed. Anyhow it may make sense to replace the included DLL with the latest one available, to take advantage of improvements in the newer version.

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

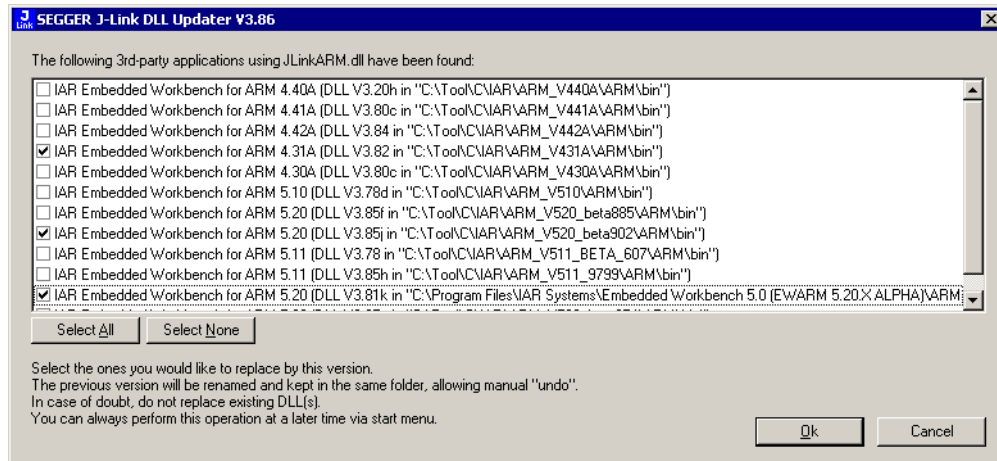
It's recommended to use the J-Link DLL updater to update the `JLinkARM.dll` in the IAR Embedded Workbench. The IAR Embedded Workbench IDE is a high-performance integrated development environment with an editor, compiler, linker, debugger. The compiler generates very efficient code and is widely used.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

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

After updating the DLL, it is recommended to verify that the new DLL is loaded as described in *Determining which DLL is used by a program* on page 39.

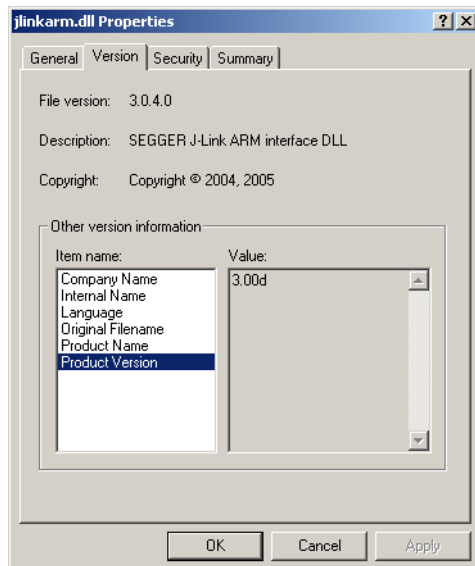
## J-Link DLL updater

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.



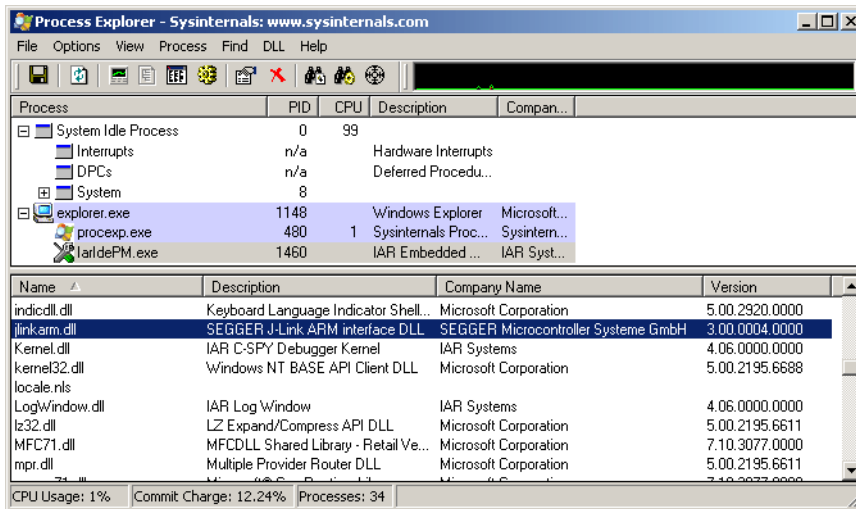
## DETERMINING THE VERSION OF JLINKARM.DLL

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



## DETERMINING WHICH DLL IS USED BY A PROGRAM

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



Process Explorer is - at the time of writing - a free utility which can be downloaded from [www.sysinternals.com](http://www.sysinternals.com).





# 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.

---

## Installing the J-Link ARM

Refer to chapter *J-Link and J-Trace related software* on page 33 for an overview about the J-Link software and documentation pack.

### SETUP PROCEDURE

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

**Note:** Check for J-Link related downloads on our website: <http://www.iar.com/jlinkarm>

- 1 Install IAR Embedded Workbench.
- 2 Connect your computer and the J-Link debug probe using the USB cable. The green LED on the front panel of the J-Link debug probe will blink for a few moments while Windows searches for a USB driver.
- 3 When you do this for the first time, Windows will start the Install wizard. Choose **Install from a specific location**.
- 4 When asked to locate the USB drivers, click the browse button and navigate to the directory `Program Files\IAR Systems\Embedded Workbench 6.n\Kickstart\arm\drivers\JLink`.  
Note that Windows XP might display a warning that the driver is not certified by Microsoft. Ignore this warning and click **Continue**.
- 5 Click **Finish**. The green LED on the J-Link debug probe stops blinking. The installation is now ready.

**Note:**Note: In Windows 7, the Installation Wizard is not started automatically and you will get a message that the driver has not been installed properly. Navigate to `IAR Systems\Embedded Workbench 6.n\arm\drivers\Jlink` and start the **InstDrivers** application. This will install the driver and the green light should shine steadily.

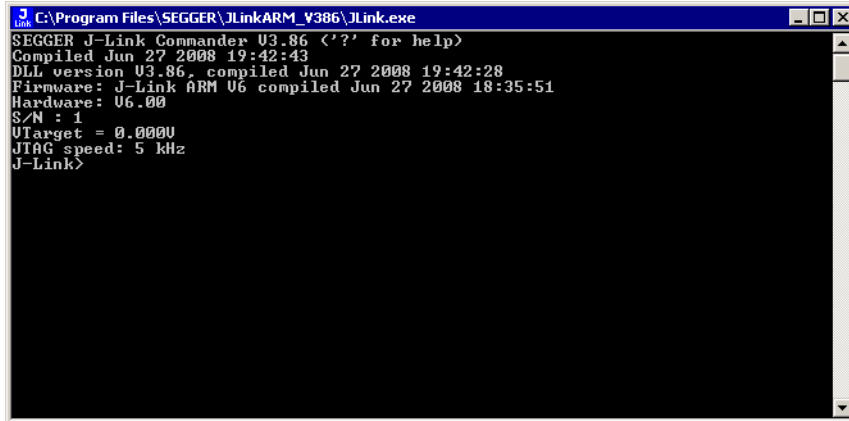
---

## Setting up the USB interface

### VERIFYING CORRECT DRIVER INSTALLATION

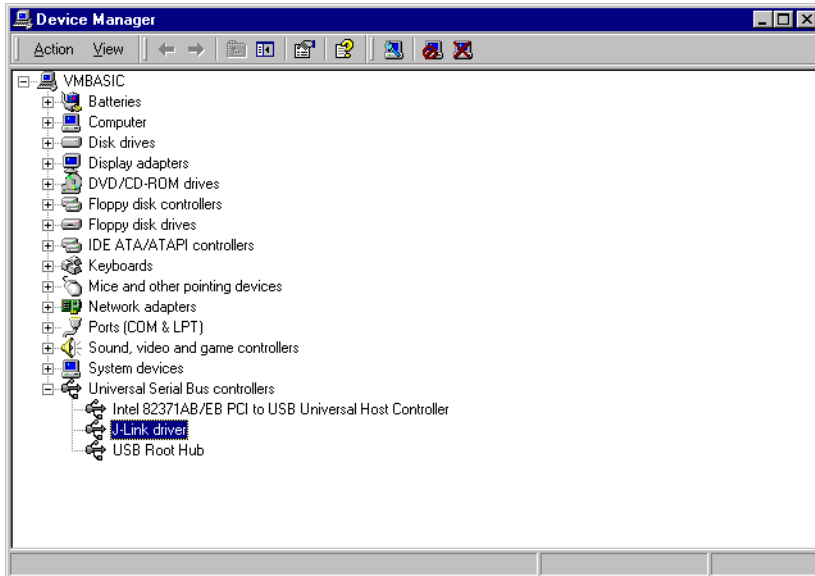
To verify the correct installation of the driver, disconnect and reconnect J-Link / J-Trace to the USB port. During the enumeration process which takes about 2 seconds, the LED on J-Link / J-Trace is flashing. After successful enumeration, the LED stays on permanently.

Start the provided sample application `JLink.exe`, available in the `arm\bin` directory of your installation, 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.

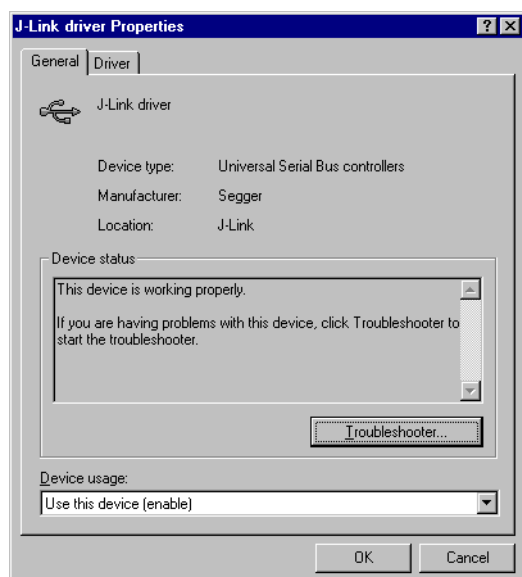


```
C:\Program Files\SEGGER\JLinkARM_V386\JLink.exe
SEGGER J-Link Commander V3.86 ('?' for help)
Compiled Jun 27 2008 19:42:43
DLL version V3.86, compiled Jun 27 2008 19:42:28
Firmware: J-Link ARM V6 compiled Jun 27 2008 18:35:51
Hardware: U6.00
S/N : 1
VTarget = 0.0000
JTAG speed: 5 kHz
J-Link>
```

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**.



If you experience problems, refer to the chapter *Support and FAQs* on page 135 for help. You can select the **Driver** tab for detailed information about driver provider, version, date and digital signer.



## 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.

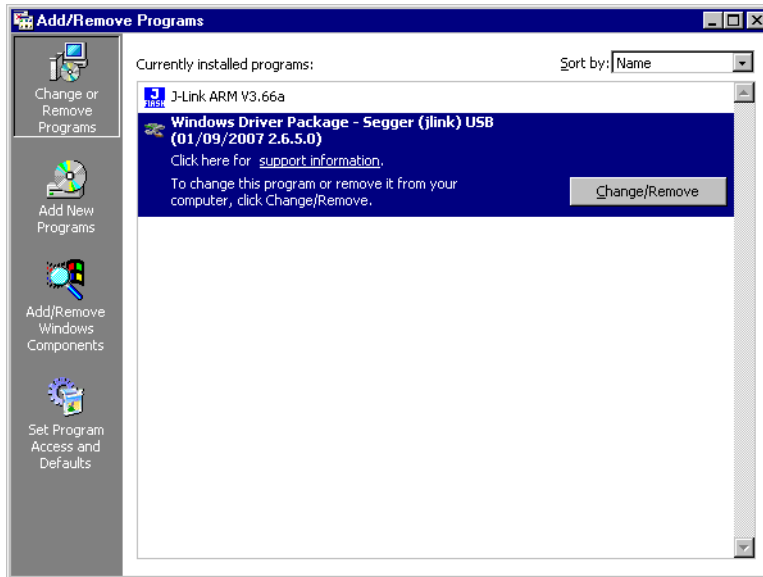
This might be the case when:

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

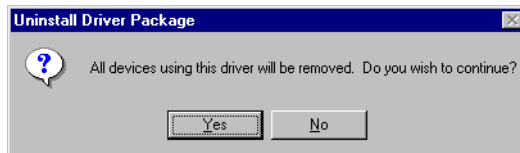
To have a clean system and help Windows to reinstall the J-Link driver, follow this procedure:

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

- 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.



- Confirm the uninstallation process.




---

## 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.

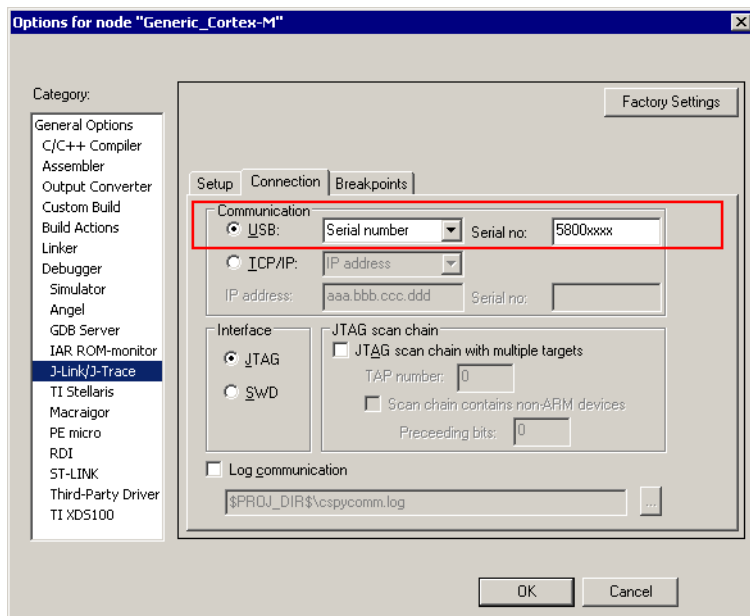
Identification	PID	Serial number
Serial number (default)	0x0101	Serial number is real serial number of the J-Link or user assigned.
USB address 0 (Deprecated)	0x0101	123456
USB address 1 (Deprecated)	0x0102	123456
USB address 2 (Deprecated)	0x0103	123456
USB address 3 (Deprecated)	0x0104	123456

*Table 1: J-Link enumeration in different identification modes*

## 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. IAR Embedded Workbench provides an extra field to type-in the serial number of the J-Link to connect to:

The following screenshot shows the connection dialog of IAR Embedded Workbench for ARM:





# Working with J-Link and J-Trace

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

---

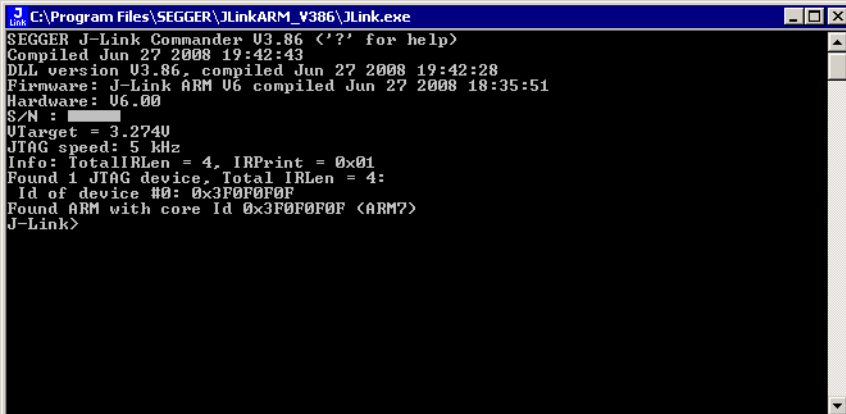
## Connecting the target system

### 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.

### 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.



```
C:\Program Files\SEGGER\JLinkARM_V386\JLink.exe
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 : 
VTarget = 3.2740
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 <ARM?>
J-Link>
```

### PROBLEMS

If you experience problems with any of the steps described above, read the chapter *Support and FAQs* on page 135 for troubleshooting tips. If you still do not find appropriate help there and your J-Link / J-Trace is an original product, you can contact 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 135 as well.

---

## 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 Ultra come with additional input/output Indicators. In the following, the meaning of these indicators will be explained.

### 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.

## 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 1: J-Link single color main indicator

## 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 2: J-Link single color LED main color indicator

## INPUT INDICATOR

Some newer J-Links such as the J-Link 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.



## 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 3: J-Link bi-color input indicator

## OUTPUT INDICATOR

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

### 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).

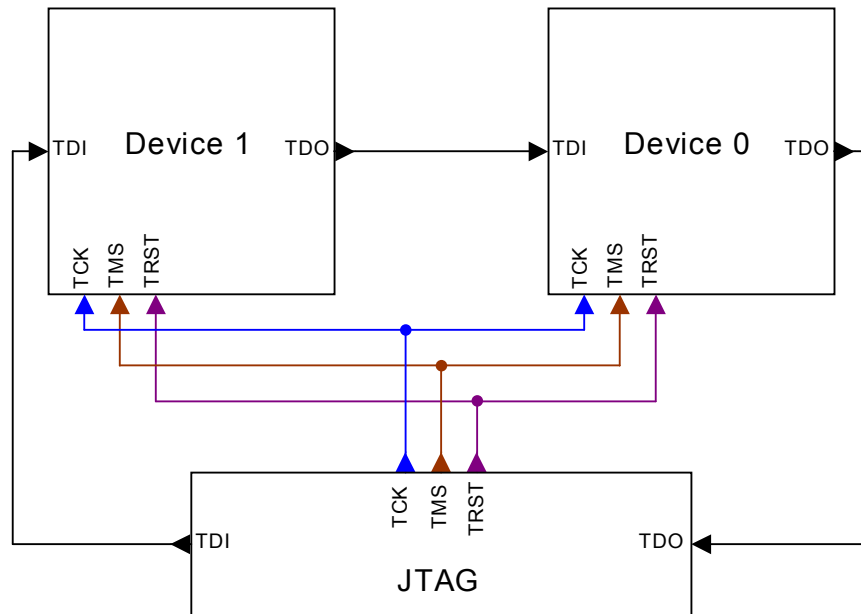
Table 4: J-Link bi-color output indicator

## JTAG interface

By default, only one ARM device is assumed to be in the JTAG scan chain. If you have multiple devices in the scan chain, you must properly configure it. To do so, you have to specify the exact position of the ARM device that should be addressed. Configuration of the scan is done by the target application. A target application can be a debugger such as the IAR C-SPY® debugger, which offers a dialog box for this purpose.

## MULTIPLE DEVICES IN THE SCAN CHAIN

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



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

### Configuration

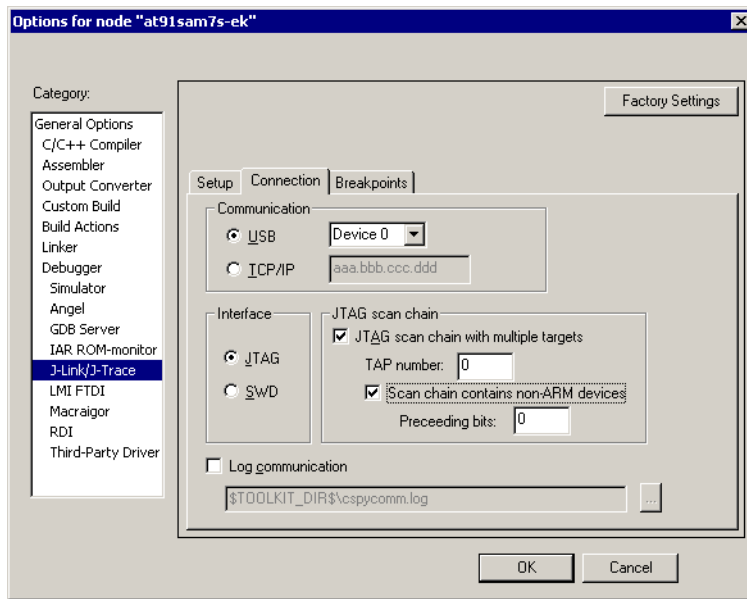
The configuration of the scan chain depends on the application used. Read *JTAG interface* on page 49 for further instructions and configuration examples.

## CONFIGURATION DIALOG BOXES

As explained before, it is responsibility of the application to allow the user to configure the scan chain. This is done in a dialog box.

## IAR J-Link configuration dialog box

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



# 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:

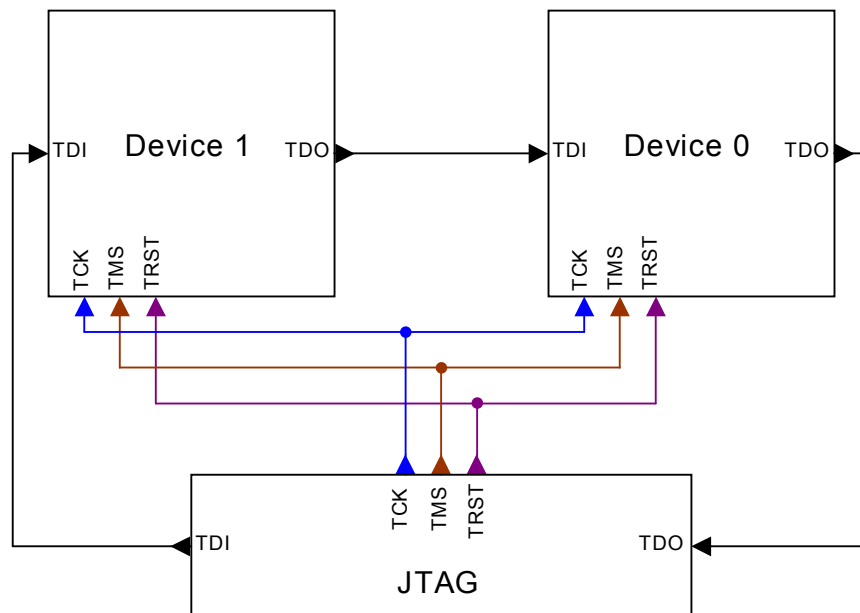
- The position of the target device in the scan chain
- The total number of bits in the instruction registers of the devices before the target device (IR len).

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
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: Example scan chain configurations

The target device is marked in blue.

## JTAG SPEED

There are basically three types of speed settings:

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

These are explained below.

### Fixed JTAG speed

The target is clocked at a fixed clock speed. The maximum JTAG speed the target can handle depends on the target itself. In general ARM cores without JTAG synchronization logic (such as ARM7-TDMI) can handle JTAG speeds up to the CPU speed, ARM cores with JTAG synchronization logic (such as ARM7-TDMI-S, ARM946E-S, ARM966EJ-S) can handle JTAG speeds up to 1/6 of the CPU speed.

JTAG speeds of more than 10 MHz are not recommended.

### Automatic JTAG speed

Selects 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.

### 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.

---

## SWD interface

The J-Link support ARM's 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.

### 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 11.

### 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.

### Max. SWO speeds

The supported SWO speeds depend on the connected emulator. They can be retrieved

from the emulator. Currently, the following are supported:

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

Table 6: J-Link supported SWO input speeds

## 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 53 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.

### Example 1

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

Possible SWO output speeds are:

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

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

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

Permitted combinations are:

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

Table 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 8: Permitted SWO speed combinations

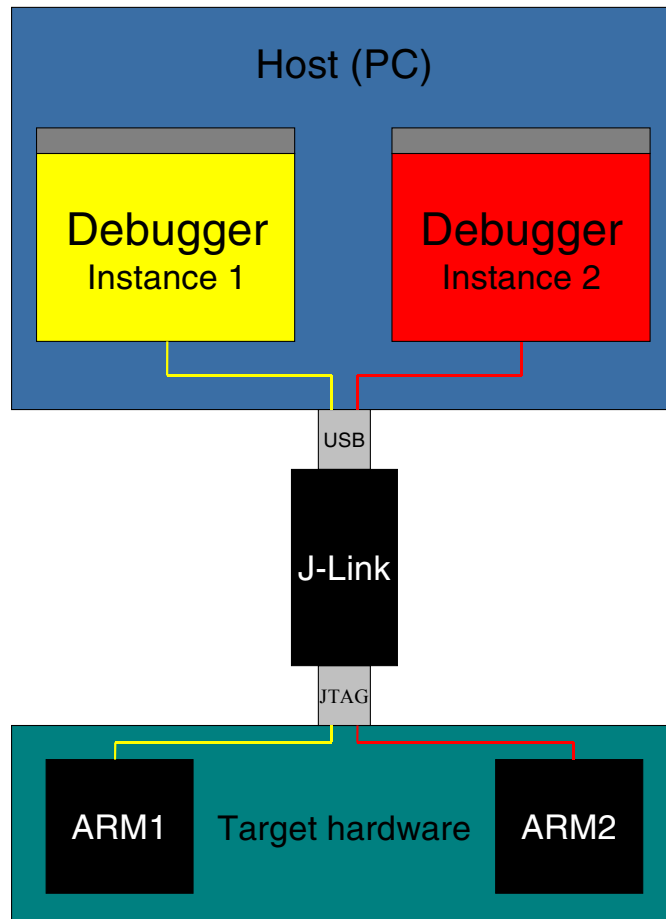
## Multi-core debugging

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.

## HOW MULTI-CORE DEBUGGING WORKS

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

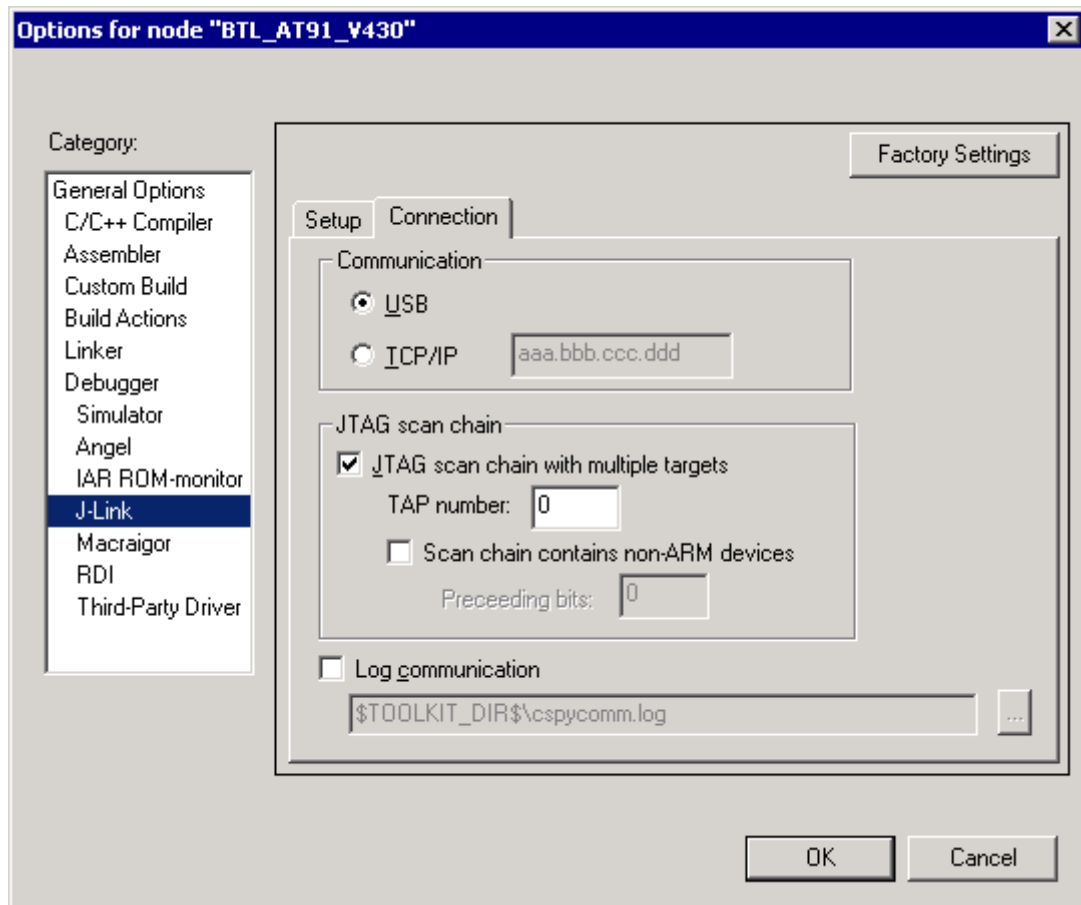
The following figure shows a host, debugging two ARM cores with two instances of the same debugger.



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

## USING MULTI-CORE DEBUGGING IN DETAIL

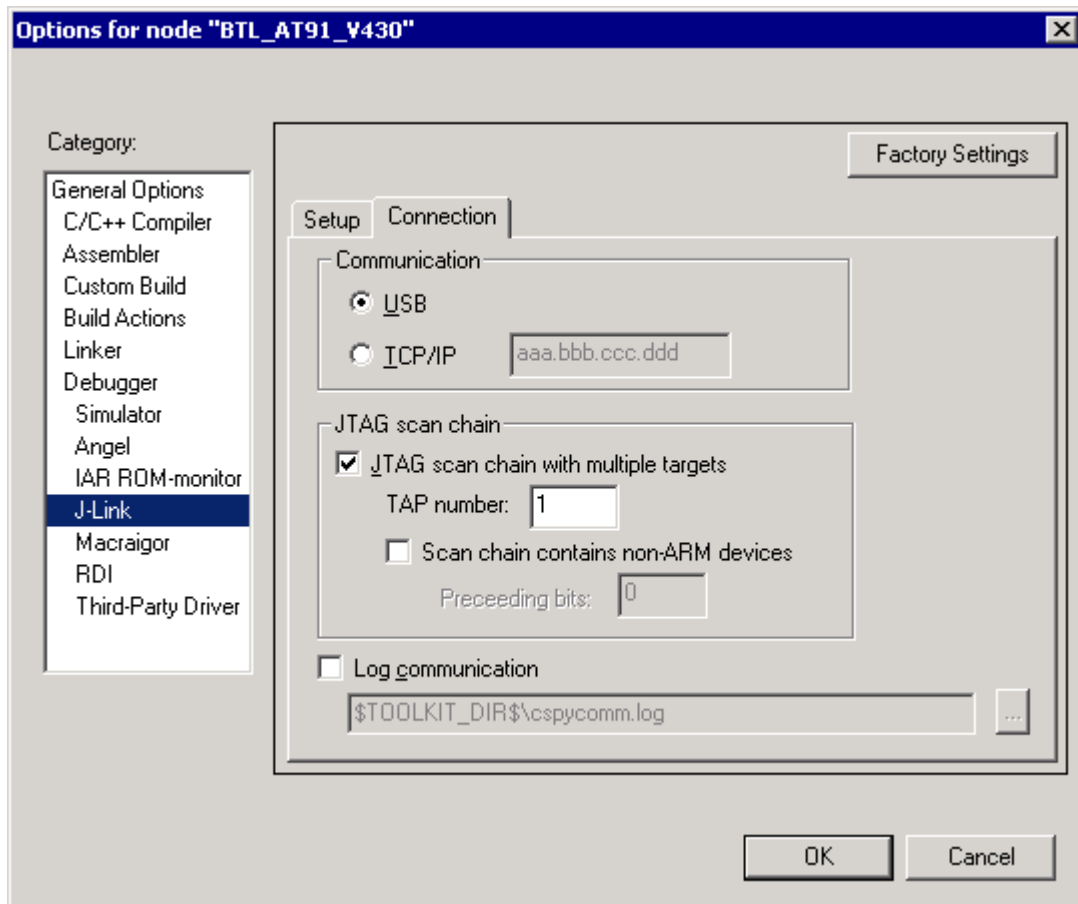
- 1 Connect your target to J-Link / J-Trace.
- 2 Start IAR Embedded Workbench for ARM.
- 3 Choose Project | Options and configure your scan chain. The picture below shows the configuration for the first ARM core on your target.



- 4 Start debugging the first core.
- 5 Start 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.



- 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 9: Multicore debugging

Cores to debug are marked in blue.

## 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.

### 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

## 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.

---

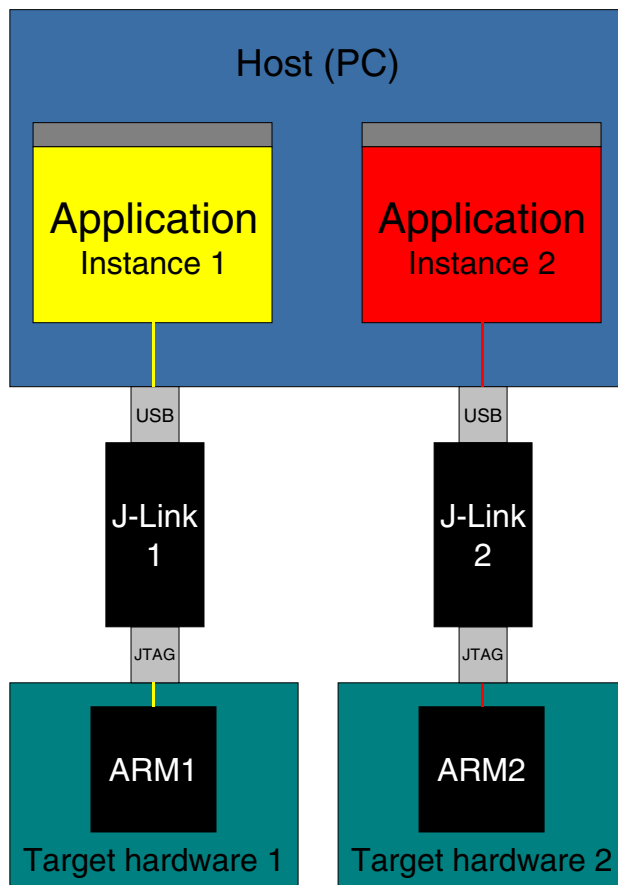
## Connecting multiple J-Links / J-Traces to your PC

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.

### HOW DOES IT WORK?

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

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



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

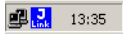
For these J-Links, we recommend to re-configure them to use the new enumeration method (report real serial number).

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

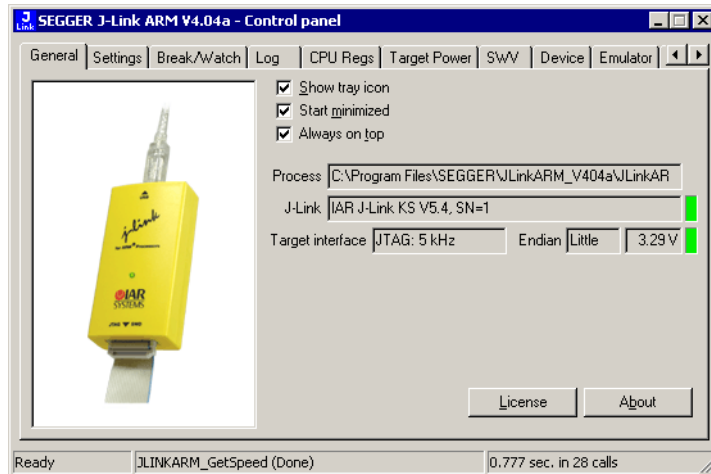
Re-configuring J-Link to use the new method does not have any bad side-effects on the current debug environment. Usually the user does not see any difference as long as only one emulator is connected.

## J-Link control panel

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



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



## 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

### 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:

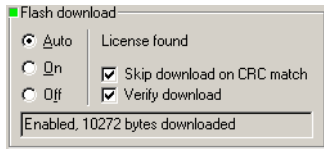
- **Process:** Shows the path of the file which loaded the DLL.
- **J-Link:** Shows OEM of the connected J-Link, the hardware version and the Serial number. If no J-Link is connected it shows "not connected" and the color indicator is red.
- **Target interface:** Shows the selected target interface (JTAG/SWD) and the current JTAG speed. The target current is also shown. (Only visible if J-Link is connected)
- **Endian:** Shows the target endianness (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.

## 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 be changed by the user himself. All settings which are modified during the debug session have to be saved by pressing **Save settings**, otherwise they are lost when the debug session is closed.

### Section: Flash download

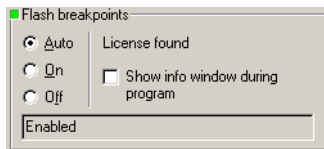
In this section, settings for the use of the J-Link ARM FlashDL feature and related settings can be configured. When a license for J-Link ARM FlashDL is found, the color indicator is green and "License found" appears right to the J-Link ARM FlashDL usage settings.



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

### Section: Flash breakpoints:

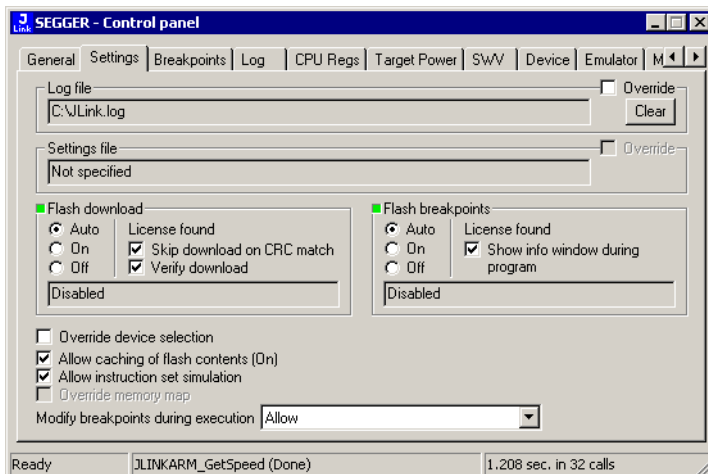
In this section, settings for the use of the FlashBP feature and related settings can be configured. When a license for FlashBP is found, the color indicator is green and "License found" appears right to the FlashBP usage settings.



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

## Flash download and flash breakpoints independent settings

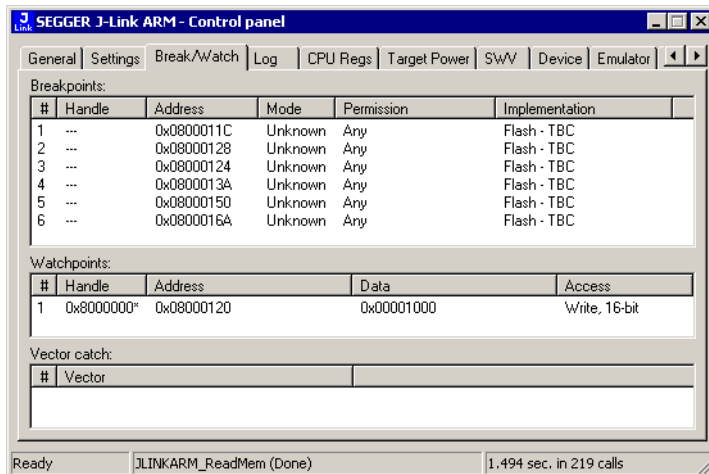
These settings do not belong to the J-Link flash download and flash breakpoints settings section. They can be configured without any license needed.



- **Log file:** Shows the path where the J-Link log file is placed. It is possible to override the selection manually by enabling the Override checkbox. If the Override checkbox is enabled a button appears which let the user choose the new location of the log file.
- **Settings file:** Shows the path where the configuration file is placed. This configuration file contains all the settings which can be configured in the **Settings** tab.
- **Override device selection:** If this checkbox is enabled, a dropdown list appears, which allows the user to set a device manually. This especially makes sense when J-Link can not identify the device name given by the debugger or if a particular device is not yet known to the debugger, but to the J-Link software.
- **Allow caching of flash contents:** If this checkbox is enabled, the flash contents are cached by J-Link to avoid reading data twice. This speeds up the transfer between debugger and target.
- **Allow instruction set simulation:** If this checkbox is enabled, ARM instructions will be simulated as far as possible. This speeds up single stepping, especially when `FlashBPs` are used.
- **Save settings:** When this button is pushed, the current settings in the **Settings** tab will be saved in a configuration file. This file is created by J-Link and will be created for each project and each project configuration (e.g. `Debug_RAM`, `Debug_Flash`). If no settings file is given, this button is not visible.
- **Modify breakpoints during execution:** This dropdown box allows the user to change the behavior of the DLL when setting breakpoints if the CPU is running. The following options are available:
  - Allow:** Allows settings breakpoints while the CPU is running. If the CPU needs to be halted in order to set the breakpoint, the DLL halts the CPU, sets the breakpoints and restarts the CPU.
  - Allow if CPU does not need to be halted:** Allows setting breakpoints while the CPU is running, if it does not need to be halted in order to set the breakpoint. If the CPU has to be halted the breakpoint is not set.
  - Ask user if CPU needs to be halted:** If the user tries to set a breakpoint while the CPU is running and the CPU needs to be halted in order to set the breakpoint, the user is asked if the breakpoint should be set. If the breakpoint can be set without halting the CPU, the breakpoint is set without explicitly confirmation by the user.
  - Do not allow:** It is not allowed to set breakpoints while the CPU is running.

## Break/Watch

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



### 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

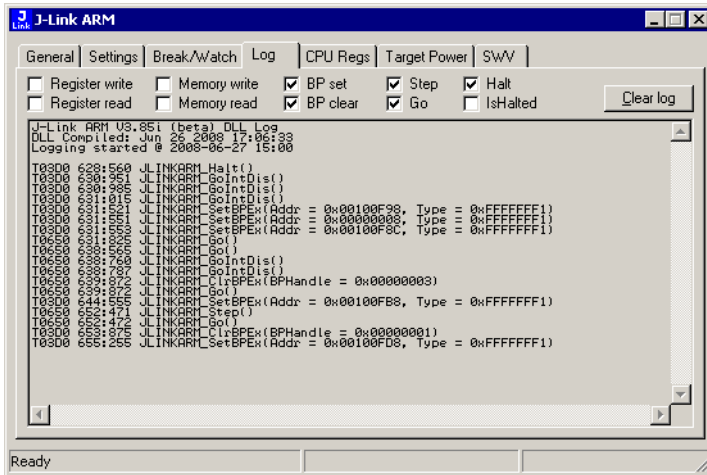
In this section, all data breakpoints which are listed in the DLL internal breakpoint list are shown.

- **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.

### Log

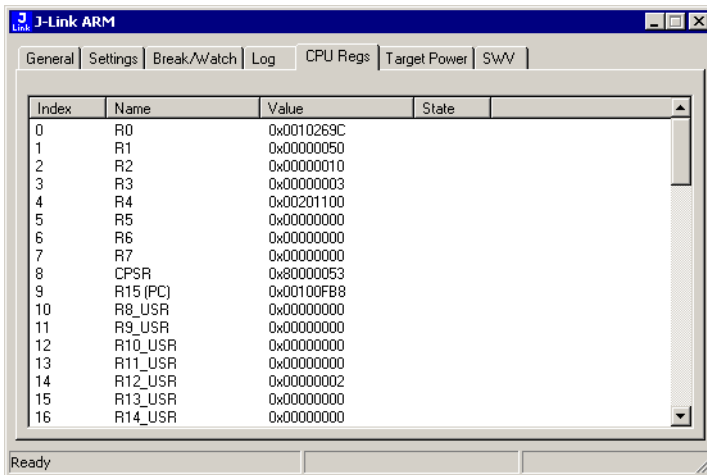
In this section the log output of the DLL is shown. The user can determine which function calls should be shown in the log window.

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



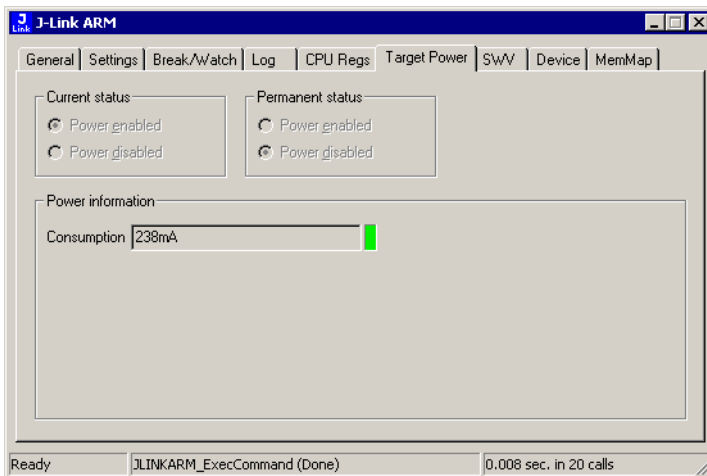
## CPU Regs

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



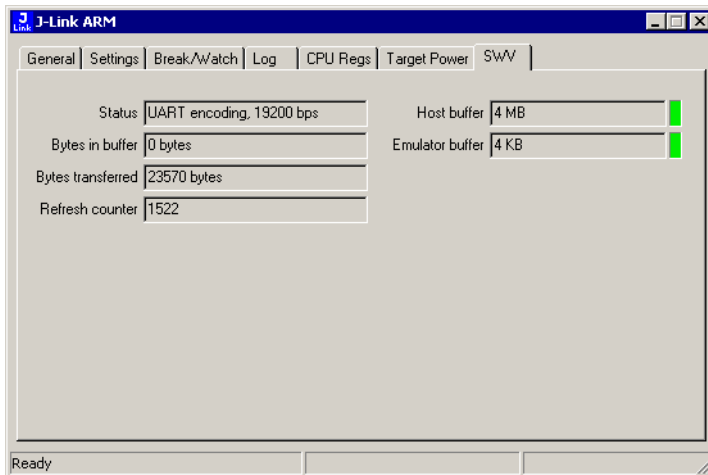
## Target Power

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



## SWV

In this section SWV information are shown.



- **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.

---

## Reset strategies

J-Link / J-Trace supports different reset strategies. This is necessary because there is no single way of resetting and halting an ARM core before it starts to execute instructions. For example reset strategies which use the reset pin can not succeed on targets where the reset pin of the CPU is not connected to the reset pin of the JTAG connector. Reset strategy 0 is always the recommended one because it has been adapted to work on every target even if the reset pin (Pin 15) is not connected.

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

The instructions which are executed can cause various problems. Some cores can be completely "confused", which means they can not be switched into debug mode (CPU can not be halted). In other cases, the CPU may already have initialized some hardware components, causing unexpected interrupts or worse, the hardware may have been initialized with illegal values. In some of these cases, such as illegal PLL settings, the CPU may be operated beyond specification, possibly locking the CPU.

## STRATEGIES FOR ARM 7/9 DEVICES

### 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.

### **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.

### **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.

### **Type 3: No reset**

No reset is performed. Nothing happens.

### **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

### **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.

### **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.

## Type 7: Reserved

Reserved reset type.

## 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.

## 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.

# 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.

## Type 0: Normal

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

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

This type of RESET can fail:

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

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

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

## 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\_CORERESSET bit before reset, which causes the CPU to halt before execution of the first instruction.

## 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\_CORERESSET bit before reset, which causes the CPU to halt before execution of the first instruction.

## 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\_CORERESSET bit can not be set in order to guarantee that the core is halted immediately after reset.

**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/CPU's 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.

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

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

**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.

**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\_CORERESSET bit is used to halt the CPU before it executes a single instruction.

**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.

**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\_CORERESSET in order to halt the CPU after reset, before it executes a single instruction. Then the watchdog of the S3FN60D device is disabled.

---

## 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  $\mu$ s per call.

The DCC handler, as well as the optional DCC abort handler, is part of the J-Link software package and can be found in the Program Files\IAR Systems\Embedded Workbench 6.n\arm\src\debugger\DCC directory of the package.

## WHAT IS REQUIRED?

- An application program on the host (typically a debugger) that uses DCC, in this case C-SPY
- 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`.

## 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.

## 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

---

## J-Link script files

In some situations it is 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. Normally, the J-Link script file is automatically set up by the debugger (if needed), based on the selected device in IAR Embedded Workbench. 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.

## ACTIONS THAT CAN BE CUSTOMIZED

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

### ResetTarget()

#### Description

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

#### Prototype

```
void ResetTarget(void);
```

## InitEMU()

### Description

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 72.

### Prototype

```
void InitEMU(void);
```

## InitTarget()

### Description

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);
```

## 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.

## MessageBox()

### Description

Outputs a string in a message box.

### Prototype

```
__api__ int MessageBox(const char * sMsg);
```

## 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);
```

## Report()

### Description

Outputs a constant character string on stdio.

### Prototype

```
__api__ int Report(const char * sMsg);
```

## 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);
```

## 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 72.

### Prototype

```
__api__ int JTAG_SetDeviceId(int DeviceIndex, unsigned int Id);
```

## 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);
```

## 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 72.

### Prototype

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

## 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 72.

### Prototype

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

## 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 72.

### Prototype

```
__api__ int JTAG_WriteDR(unsigned __int64 tdi, int NumBits);
```

## JTAG\_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 72.

### Prototype

```
__api__ int JTAG_StoreDR(unsigned __int64 tdi, int NumBits);
```

### JTAG\_Write()

#### Description

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

### Prototype

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

### JTAG\_Store()

#### Description

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

### Prototype

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

### JTAG\_GetU32()

#### Description

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

### Prototype

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

### JTAG\_WriteClocks()

#### Description

Writes a given number of clocks.

### Prototype

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

### JTAG\_StoreClocks()

#### Description

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

### Prototype

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

### 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 72.

**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);
```

## SYS\_Sleep()

### Description

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

### Prototype

```
__api__ int SYS_Sleep(int Delaysms);
```

## 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 specific 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	Used to set the target CPU core. If auto-detection of the device is not possible, you have to set this variable to tell J-Link what CPU core is connected to it. This variable can only be set to a known Global J-Link DLL constant. For a list of all valid values, please refer to <i>Global DLL constants</i> on page 73. <b>Example</b> CPU = ARM926EJS;	Write-only
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. <b>Example</b> JTAG_IRPre = 6;	Read/Write
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. <b>Example</b> JTAG_DRPre = 2;	Read/Write
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. <b>Example</b> JTAG_IRPost = 6;	Read/Write
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. <b>Example</b> JTAG_DRPost = 0;	Read/Write
JTAG_IRLen	IR-Len (in bits) of the device we want to communicate with. <b>Example</b> JTAG_IRLen = 4;	Read/Write
JTAG_TotalIRLen	Computed automatically, based on the values of <i>JTAG_IRPre</i> , <i>JTAG_DRPre</i> , <i>JTAG_IRPost</i> and <i>JTAG_DRPost</i> . <b>Example</b> v = JTAG_TotalIRLen;	Read-only
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. <b>Allowed values</b> 0 Auto-detection is enabled. 1 Auto-detection is disabled.	Write-only
JTAG_Speed	Sets the JTAG interface speed. Speed is given in kHz. <b>Example</b> JTAG_Speed = 2000; // 2MHz JTAG speed	Write-only

Table 10: Global DLL variables



Variable	Description	R/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. <b>Example</b> <pre>JTAG_ResetPin = 0; SYS_Sleep(5); // Give pin some time to get low JTAG_ResetPin = 1;</pre>	Write-only
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. <b>Example</b> <pre>JTAG_TRSTPin = 0; SYS_Sleep(5); // Give pin some time to get low JTAG_TRSTPin = 1;</pre>	Write-only
JTAG_TCKPin	Pulls TCK pin LOW / HIGH. Value assigned to reset pin reflects the state. 0 = LOW, 1 = HIGH. <b>Example</b> <pre>JTAG_TCKPin = 0;</pre>	Read/Write
JTAG_TDIPin	Pulls TDI pin LOW / HIGH. Value assigned to reset pin reflects the state. 0 = LOW, 1 = HIGH. <b>Example</b> <pre>JTAG_TDIPin = 0;</pre>	Read/Write
JTAG_TMSPin	Pulls TMS pin LOW / HIGH. Value assigned to reset pin reflects the state. 0 = LOW, 1 = HIGH. <b>Example</b> <pre>JTAG_TMSPin = 0;</pre>	Read/Write
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 <code>InitEmu()</code> does not have any effect. <b>Example</b> <pre>void InitEmu(void) {     EMU_ETB_IsPresent = 1; }</pre>	Write-only

Table 10: Global DLL variables

## 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.

### 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\_R4

## SCRIPT FILE LANGUAGE

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

### 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: =, \*=, /=, +=, -=, <<=, >>=, &=, ^=, |=

### Supported type specifiers

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

- void
- char
- int (32-bit)
- \_\_int64

### Supported type qualifiers

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

- const

- signed
- unsigned

### Supported declarators

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

- Array declarators

### Supported selection statements

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

- if-statements
- if-else-statements

### Supported iteration statements

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

- while
- do-while

### Jump statements

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

- return

### 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`.

## EXECUTING J-LINK SCRIPT FILES

### In J-Link commander

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

### 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.

---

## 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.

## LIST OF AVAILABLE COMMANDS

The table below lists and describes the available command strings.

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

Table 11: Available command line options

### 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 85.

#### Example

```
device = AT91SAM7S256
```

### DisableFlashBPs

This command disables the FlashBP feature.

#### Syntax

```
DisableFlashBPs
```

### DisableFlashDL

This command disables the J-Link ARM FlashDL feature.

#### Syntax

```
DisableFlashDL
```

### EnableFlashBPs

This command enables the FlashBP feature.

## Syntax

```
EnableFlashBPs
```

## EnableFlashDL

This command enables the J-Link ARM FlashDL feature.

## Syntax

```
EnableFlashDL
```

## 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
```

## 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  $\geq 256$  byte.

### Typical applications

Refer to chapter *Fast GPIO bug* on page 105 for an example.

### Syntax

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

### Example

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

## 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 105 for an example.

### Syntax

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

### Example

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

## 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
```

## 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
```

## 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
```

## 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
```

## SetResetType

This command selects the reset strategy 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 64. 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
```

## 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
```

## 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.
```

## 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
```

## SupplyPower

This command activates power supply over pin 19 of the JTAG connector. The J-Link has 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
```

## SupplyPowerDefault

This command activates power supply over pin 19 of the JTAG connector permanently. The J-Link has 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

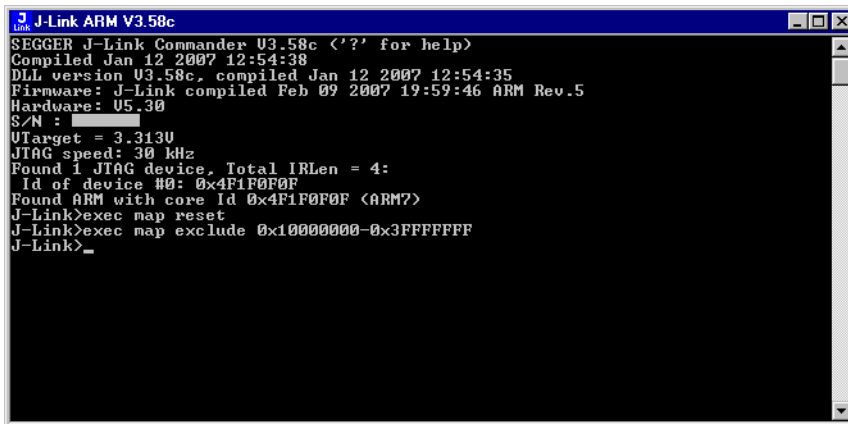
```
SupplyPowerDefault = 1
```



## USING COMMAND STRINGS

### 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.



```

J-Link ARM V3.58c
SEGGER J-Link Commander U3.58c <'?' for help>
Compiled Jan 12 2007 12:54:38
DLL version U3.58c, compiled Jan 12 2007 12:54:35
Firmware: J-Link compiled Feb 09 2007 19:59:46 ARM Rev.5
Hardware: U5.30
S/N : 
Utarget = 3.313U
JTAG speed: 30 kHz
Found 1 JTAG device, Total IRLen = 4:
  Id of device #0: 0x4F1F0F0F
Found ARM with core Id 0x4F1F0F0F <ARM?>
J-Link>exec map reset
J-Link>exec map exclude 0x10000000-0x3FFFFFFF
J-Link>_
  
```

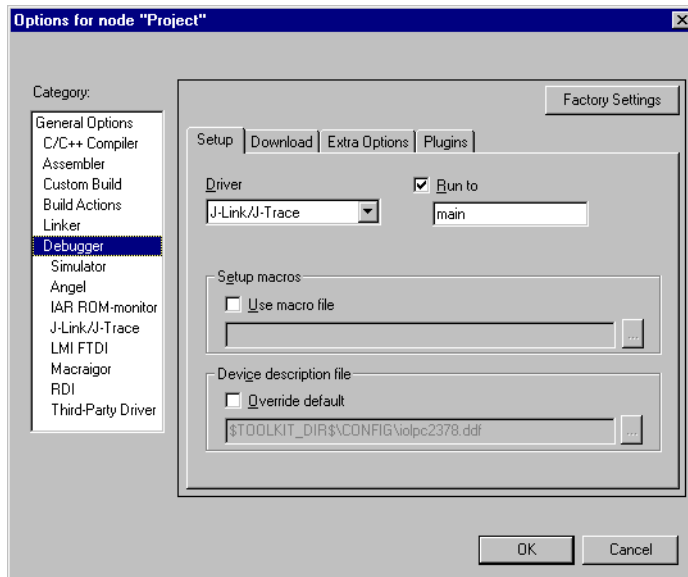
### Example

```

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

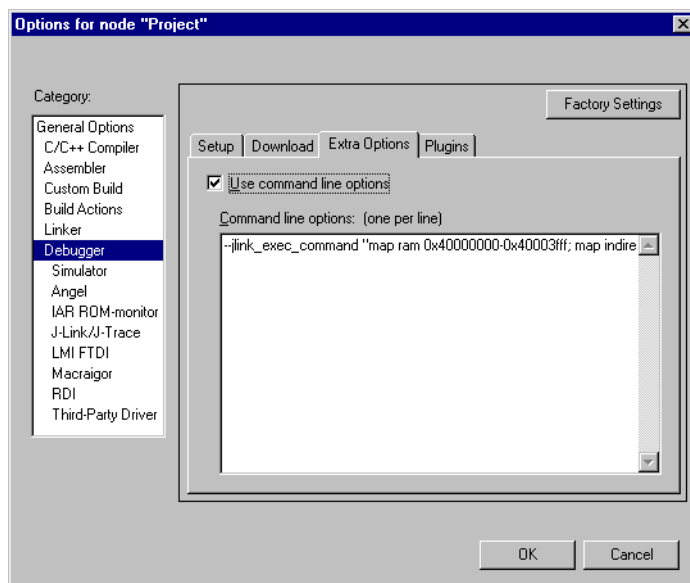
### IAR Embedded Workbench

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**.



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.



---

## 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).

---

## Cache handling

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

### 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.

## 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.

## CACHE HANDLING OF ARM7 CORES

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

## 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.



# Flash download

This chapter describes how the flash download feature of the DLL can be used.

---

## 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.

---

## Licensing

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

---

## 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 Segger's website:

*[http://www.segger.com/jlink\\_supported\\_devices.html](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.

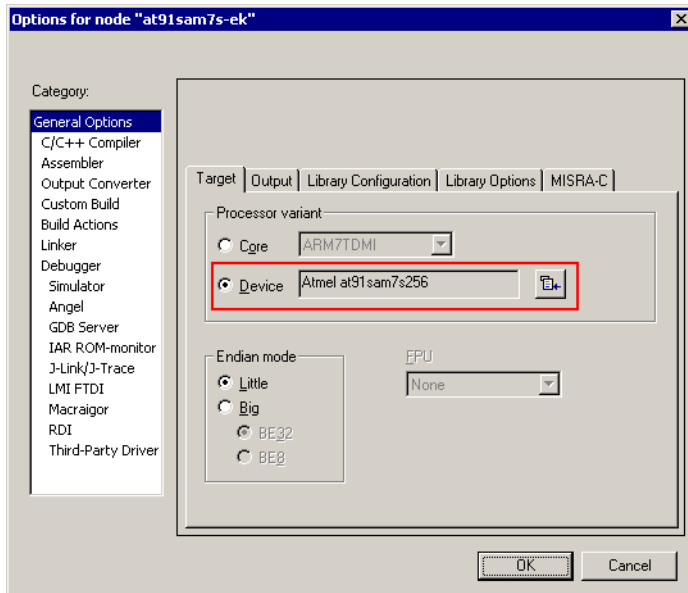
Furthermore, flash download is also available for all CFI-compliant external NOR-flash devices.

# Setup

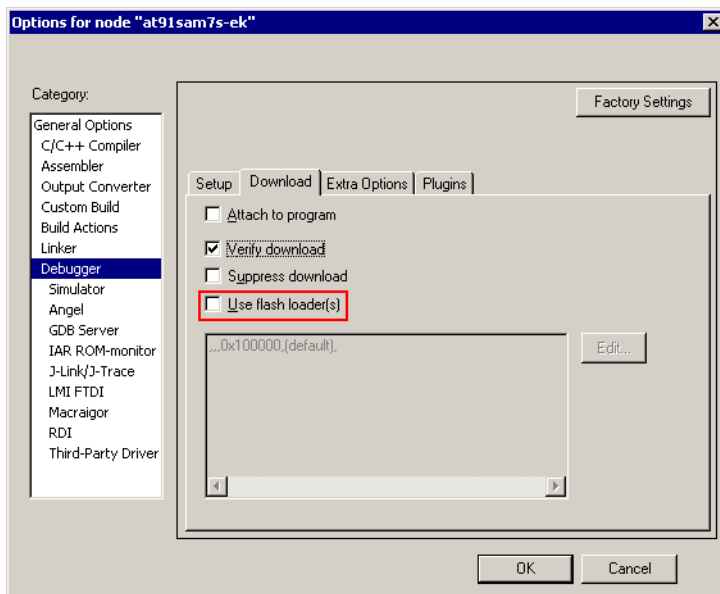
## IAR EMBEDDED WORKBENCH

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**.



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.

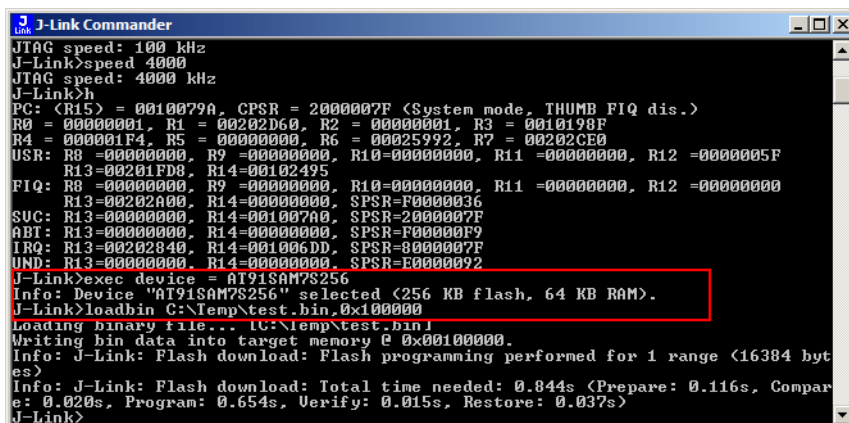


## J-LINK COMMANDER

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

```
exec device = <DeviceName>
```

<DeviceName> is the name of the device for which download into internal flash memory shall be enabled. For a list of supported devices, please refer to *Supported devices* on page 85.



```

J-Link Commander
JTAG speed: 100 kHz
J-Link>speed 4000
JTAG speed: 4000 kHz
J-Link>h
PC: (R15) = 0010079A, CPSR = 2000007F (System mode, THUMB FIQ dis.)
R0 = 00000001, R1 = 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
SUC: R13 = 00000000, R14 = 001007A0, SPSR = 2000007F
ABT: R13 = 00000000, R14 = 00000000, SPSR = F00000F9
IRQ: R13 = 00202040, R14 = 001006DD, SPSR = 0000007F
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 @x100000
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 bytes)
Info: J-Link: Flash download: Total time needed: 0.044s (Prepare: 0.116s, Compare: 0.020s, Program: 0.654s, Verify: 0.015s, Restore: 0.037s)
J-Link>
  
```

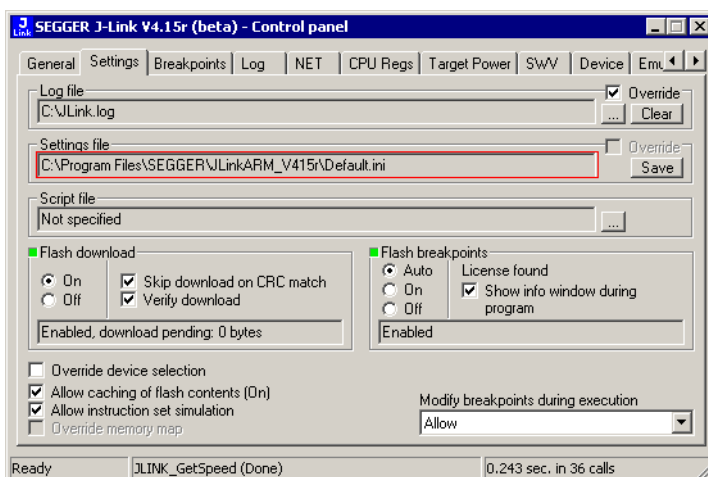
## Setup for CFI flash

The setup for download into CFI-compliant memory is different from the one for internal flash.

## IAR EMBEDDED WORKBENCH

Using the J-Link flash download feature with IAR Embedded Workbench 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.

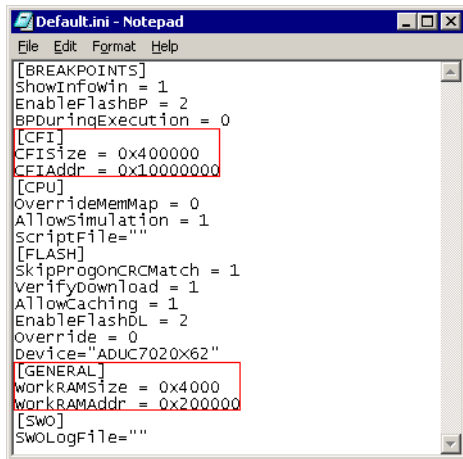


Close the debug session and open the settings file with a text editor. Add the following lines to the file:

```

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

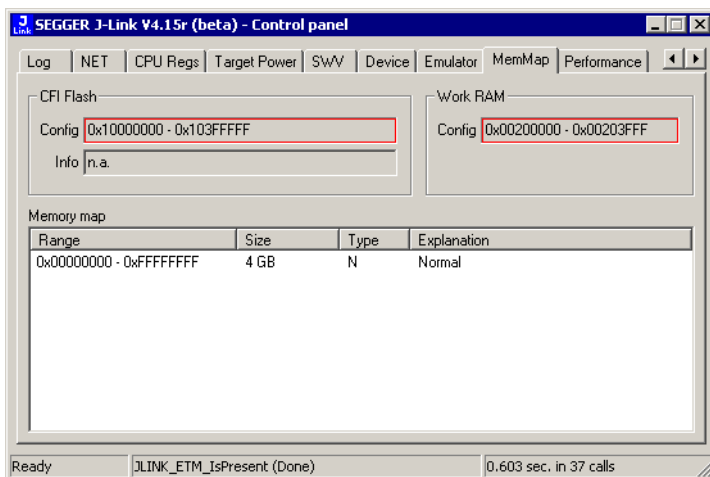
After this the file should look similar to the sample in the following screenshot.



```
Default.ini - Notepad
File Edit Format Help
[BREAKPOINTS]
ShowInFowin = 1
EnableFlashBP = 2
BPDuringExecution = 0
[CFI]
CFISize = 0x400000
CFIAddr = 0x10000000
[CPU]
OverrideMemMap = 0
AllowSimulation = 1
ScriptFile=""
[FLASH]
SkipPrigonCRCMatch = 1
VerifyDownload = 1
AllowCaching = 1
EnableFlashBL = 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.



## J-LINK COMMANDER

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

```
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, 0BBBBBBBB // GPIOD high config, A16-A18
w4 0x40011800, 0BBBBBBBB // GPIOE low config, A19-A23
w4 0x40011804, 0BBBBBBBB // GPIOE high config, D5-D12
w4 0x40011C00, 0x44BBBBBB // GPIOF low config, A0-A5
w4 0x40011C04, 0BBBBB4444 // 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
```

## 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).



# Flash breakpoints

This chapter describes how the flash breakpoints feature of the DLL can be used.

---

## 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 an 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 than 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.

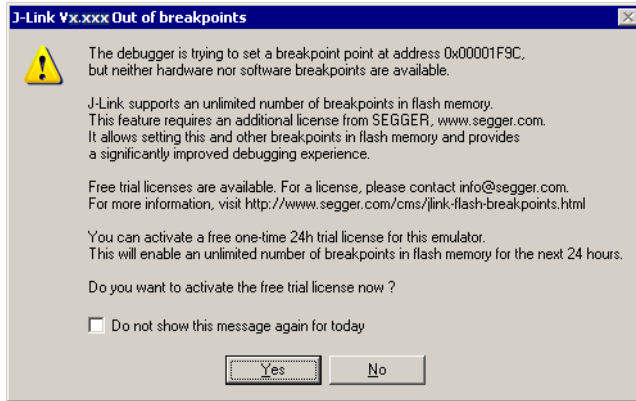
---

## 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 25.

## 24H FLASH BREAKPOINT TRIAL LICENSE

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



---

## Supported devices

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

[http://www.segger.com/jlink\\_supported\\_devices.html](http://www.segger.com/jlink_supported_devices.html)

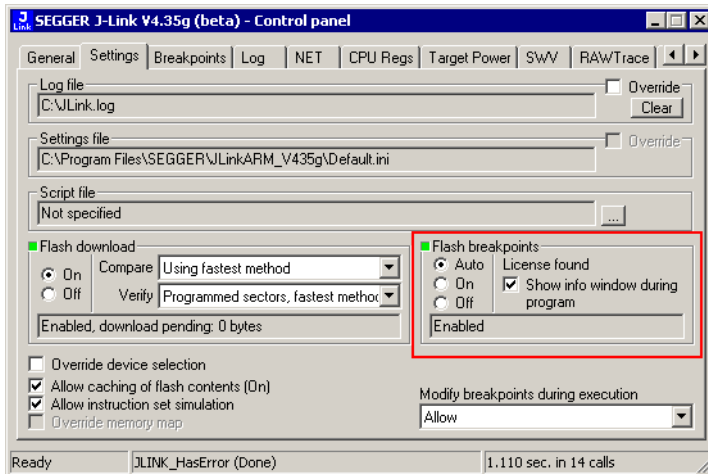
In general, J-Link can be used with any ARM7/9/11, Cortex-M0/M1/M3/M4 and Cortex-A5/A8/R4 core even if it does not provide internal flash.

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

# Setup

## SETUP

In IAR Embedded Workbench, flash breakpoints work if 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 IAR Embedded Workbench for flash download, please refer to *Setup* on page 86. If flash breakpoints are available can be verified using the J-Link control panel:





# Device specifics

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

---

## 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

## ADUC7XXX

### 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

## 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

## AT91SAM7

### Reset strategy

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

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

### **This information is applicable to the following devices:**

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

### **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)

### **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 IAR Workbench is given.

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

## Example

```

/*****
*
*     _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(0x002f0100,0xFFFFF60,"Memory"); // Set 1 wait state for
    __sleep(20000);                 // flash (2 cycles)
    __emulatorSpeed(12000000);      // Use full JTAG speed
}

/*****
*
*     execUserReset()
*/
execUserReset() {
    __message "execUserReset()";
    _Init();
}

/*****
*
*     execUserPreload()
*/
execUserPreload() {
    __message "execUserPreload()";
    _Init();
}

```

## AT91SAM9

### JTAG settings

We recommend using adaptive clocking.

**This information is applicable to the following devices:**

- AT91RM9200
- AT91SAM9260
- AT91SAM9261
- AT91SAM9262
- AT91SAM9263

## DSPGroup

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

- DA56KLF

Currently, there are no specifics for these devices.

## Ember

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

- EM351

- EM357

Currently, there are no specifics for these devices.

---

## Energy Micro

J-Link has been tested with the following Energy Micro 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

Currently, there are no specifics for these devices.

---

## Freescle

J-Link has been tested with the following Freescle 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

## KINETIS FAMILY

### 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.

### 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.

---

## 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.

---

## Itron

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

- TRIFECTA

Currently, there are no specifics for these devices.

---

## 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

## UNLOCKING LM3SXXX DEVICES

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

---

## 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

## LPC ARM7-BASED DEVICES

### Fast GPIO bug

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

### Solution / Workaround

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

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

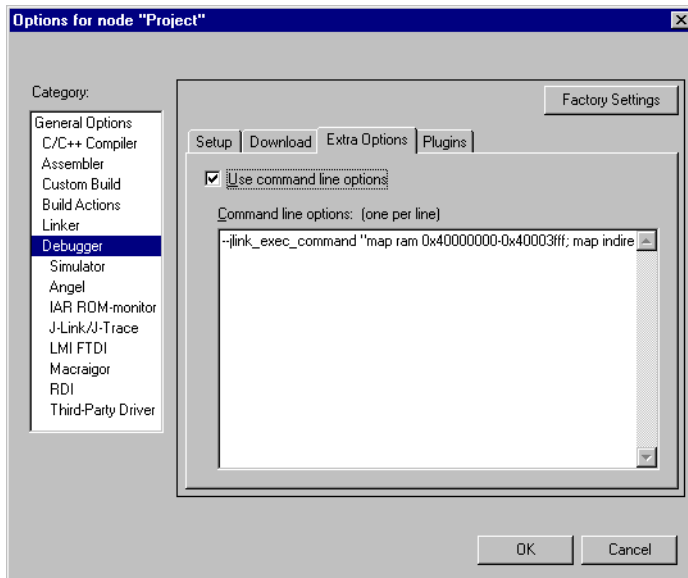
### This information is applicable to the following devices:

- LPC2101
- LPC2102
- LPC2103

- LPC213x/01
- LPC214x (all devices)
- LPC23xx (all devices)
- LPC24xx (all devices)

## Example

J-Link command line options can be used 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** tab and enter in the textfield `--jlink_exec_command "map ram 0x40000000-0x40003fff; map indirectread 0x3fffc000-0x3fffcfff; map exclude 0x3fffd000-0x3ffffff";` as shown in the screenshot below.



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 75.

## 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.

---

## 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.

---

## 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.

---

## Samsung

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

- S3FN60D

### 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.

---

## 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

## **STR91X**

### **JTAG settings**

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

## Unlocking

The devices have 3 TAP controllers built-in. When starting `J-Link.exe`, it reports 3 JTAG devices. A special tool, J-Link STR9 Commander (`JLinkSTR91x.exe`) is available to directly access the flash controller of the device. This tool can be used to erase the flash of the controller even if a program is in flash which causes the ARM core to stall. For more information about the J-Link STR9 Commander, please refer to *J-Link STR91x Commander (Command line tool)* on page 34.

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.

## Switching the boot bank

The bootbank of the STR91x devices can be switched by using the J-Link STR9 Commander which is part of the J-Link software and documentation package. For more information about the J-Link STR9 Commander, please refer to *J-Link STR91x Commander (Command line tool)* on page 34.

## STM32F10X

These device are Cortex-M3 based.

All devices of this family are supported by J-Link.

## 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 computed 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 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 2: Option bytes programming example

For a detailed description of each option byte, please refer to *ST programming manual PM0042, section "Option byte description"*.

## Securing/unsecuring the device

The user area internal flash of the STM32 devices can be protected (secured) against read by untrusted code. The J-Flash software allows securing a STM32F10x device. For more information about J-Flash, please refer to *UM08003, J-Flash User Guide*. In order to unsecure a read-protected STM32F10x device, SEGGER offers two software components:

- J-Flash
- J-Link STM32 Commander (command line utility)

For more information about J-Flash, please refer to *UM08003, J-Flash User Guide*. For more information about the J-Link STM32 Commander, please refer to *J-Link STM32 Commander (Command line tool)* on page 36.

**Note:**Unsecuring a secured device will cause a mass-erase of the internal flash memory.

## Hardware watchdog

The hardware watchdog of a STM32F10x device can be enabled by programming the option bytes. If the hardware watchdog is enabled the device is reset periodically if the watchdog timer is not refreshed and reaches 0. If the hardware watchdog is enabled by an application which is located in flash and which does not refresh the watchdog timer, the device can not be debugged anymore.

## Disabling the hardware watchdog

In order to disable the hardware watchdog the option bytes have to be re-programmed. SEGGER offers a free command line tool included in your IAR Embedded Workbench installation which reprograms the option bytes in order to disable the hardware watchdog. For more information about the STM32 commander, please refer to *J-Link STM32 Commander (Command line tool)* on page 36.

## Software watchdog

If you enable the software watchdog (independent watchdog / window watchdog) in your target application and still want to debug it, you should make sure that the watchdog does not keep running while the CPU is in debug mode (e.g. halted by J-Link). This can be configured via the DBGMCU\_CR register of the STM32F10x devices. To configure the watchdog timers to stop while the CPU is in debug mode, bits 8 and 9 of the DBGMCU\_CR have to be set:

```
*((volatile int *) (0x40040520)) = (1 << 8) | (1 << 9);
```

---

## Texas Instruments

J-Link has been tested with the following Texas Instruments devices:

- TMS470R1A64
- TMS470R1A128
- TMS470R1A256
- TMS470R1A288
- TMS470R1A384
- TMS470R1B512
- TMS470R1B768
- TMS470R1B1M
- TMS470R1VF288
- TMS470R1VF688
- TMS470R1VF689

Currently, there are no specifics for these devices.

---

## Toshiba

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

- TTPM321F10FG
- TTPM322F10FG
- TTPM323F10FG
- TTPM324F10FG
- TTPM330FDFG
- TTPM330FWFG
- TTPM330FYFG
- TTPM332FWUG
- TTPM333FDFG
- TTPM333FWFG
- TTPM333FYFG
- TTPM341FDXBG
- TTPM341FYXBG
- TTPM360F20FG
- TTPM361F10FG
- TTPM362F10FG
- TTPM363F10FG
- TTPM364F10FG
- TTPM366FDFG
- TTPM366FWFG
- TTPM366FYFG
- TTPM370FYDFG
- TTPM370FYFG
- TTPM372FWUG
- TTPM373FWDUG
- TTPM374FWUG
- TTPM380FWDFG
- TTPM380FWFG
- TTPM380FYDFG
- TTPM380FYFG
- TTPM382FSFG
- TTPM382FWFG
- TTPM395FWXBG

Currently, there are no specifics for these devices.





# Target interfaces and adapters

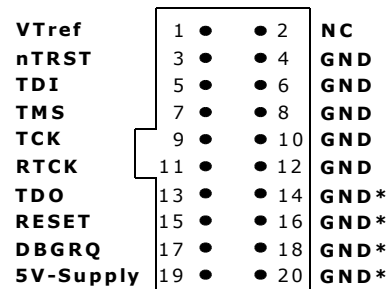
This chapter gives an overview about J-Link / J-Trace specific hardware details, such as the pinouts and available adapters.

## 20-pin JTAG/SWD connector

### PINOUT FOR JTAG

J-Link and J-Trace have a JTAG connector compatible to ARM's Multi-ICE. The JTAG connector is a 20 way Insulation Displacement Connector (IDC) keyed box header (2.54mm male) that mates with IDC sockets mounted on a ribbon cable.

\*On later J-Link products like the J-Link Ultra, these pins are reserved for firmware extension purposes. They can be left open or connected to GND in normal debug environment. They are not essential for JTAG/SWD in general.



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.
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	DBGQR	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 DBGQR 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 <i>Target power supply</i> on page 115.

Table 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.

## Target board design

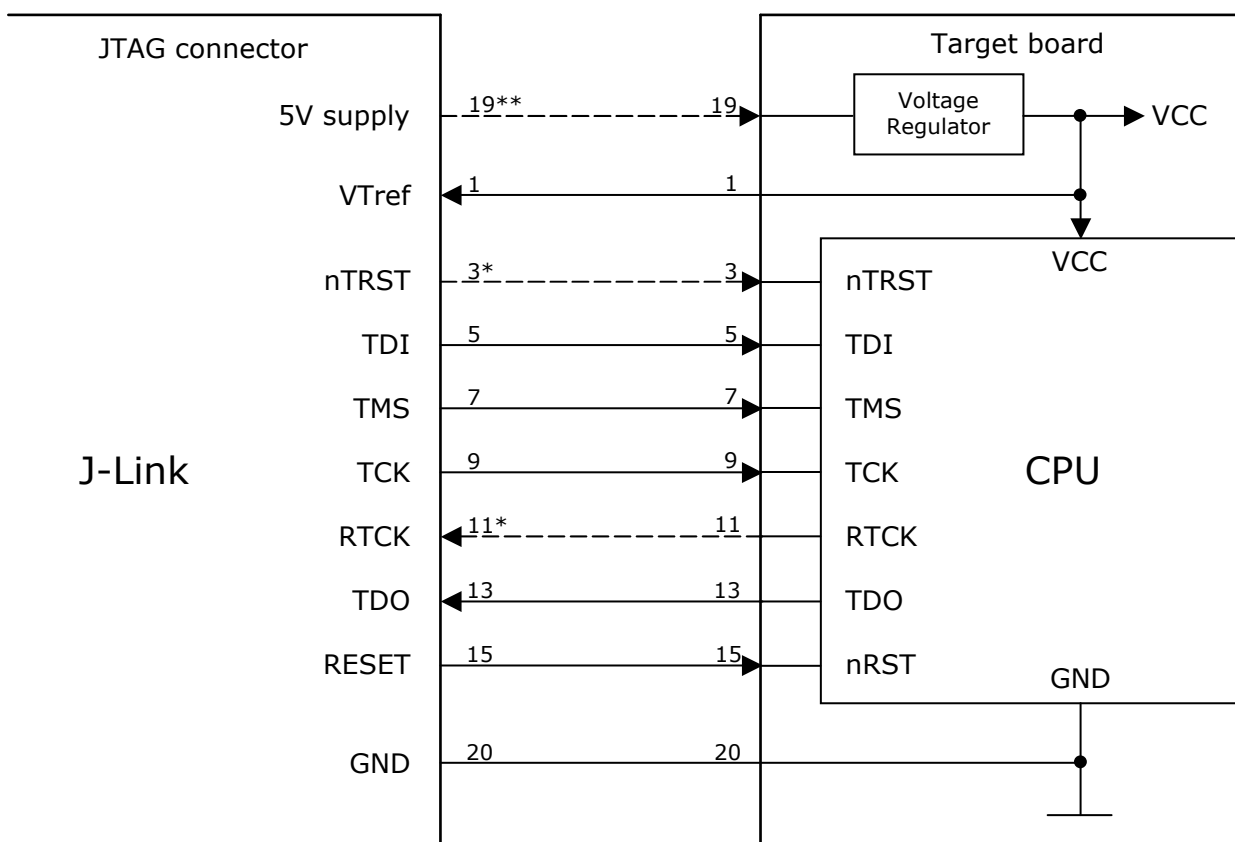
We strongly advise following the recommendations given by the chip manufacturer. These recommendations are normally in line with the recommendations given in the table *Pinout for JTAG* on page 113. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

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.

## Pull-up/pull-down resistors

Unless otherwise specified by developer's manual, pull-ups/pull-downs are recommended to be between 2.2 kOhms and 47 kOhms.

## 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 2: Command List

## PINOUT FOR SWD

The J-Link and J-Trace JTAG connector is also compatible to ARM's Serial Wire Debug (SWD).

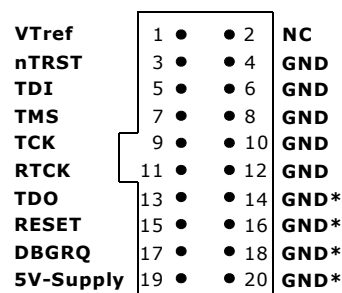
\*On later J-Link products like the J-Link Ultra, these pins are reserved for firmware extension purposes. They can be left open or connected to GND in normal debug environment. They are not essential for JTAG/SWD in general.

The following table lists the J-Link / J-Trace SWD pinout.

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	Output	Serial Wire Output trace port. (Optional, not required for SWD communication.)
15	RESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
17	Not used	NC	This pin is not connected in J-Link.
19	5V-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 <i>Target power supply</i> on page 116.

Table 3: J-Link / J-Trace SWD pinout

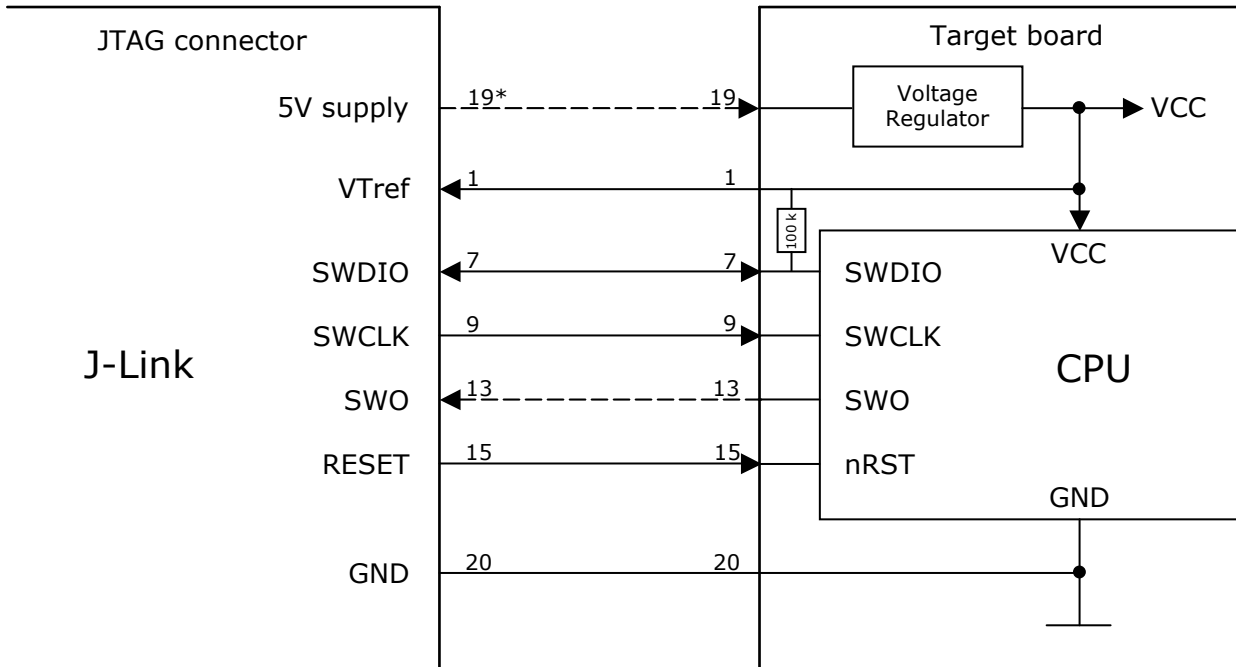
Pins 4, 6, 8, 10, 12, 14, 16, 18, 20 are GND pins connected to GND in J-Link. They should also be connected to GND in the target system.



## 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 115. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

### Typical target connection for SWD



\* Optional to supply the target board from J-Link.

## 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.

## 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
<code>power on</code>	Switch target power on
<code>power off</code>	Switch target power off
<code>power on perm</code>	Set target power supply default to "on"
<code>power off perm</code>	Set target power supply default to "off"

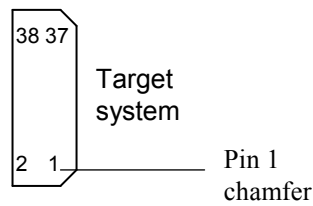
Table 4: Command List

## 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

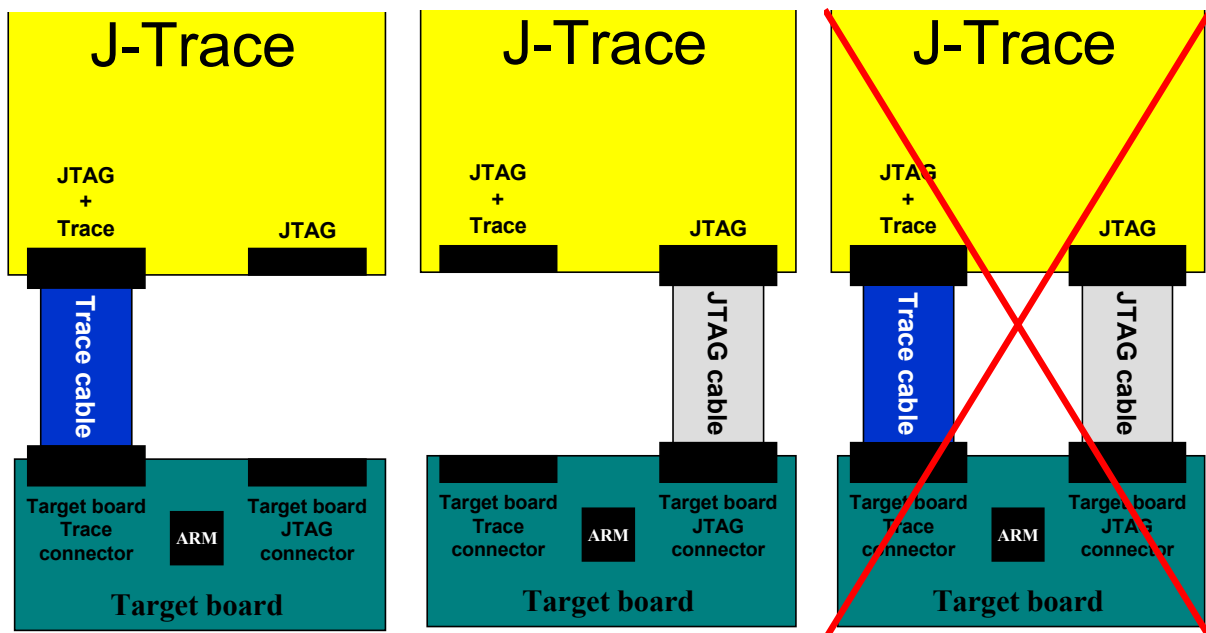
TRACEPKT[n:0] at each rising

edge of each TRACECLK or on each alternate rising or falling edge.

## 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.**



## 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	DBGREQ	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 <i>Assignment of trace information pins between ETM architecture versions</i> on page 119.
17	TMS	Test mode select from run control to the JTAG port.
18	Trace signal 11	Trace signal. For more information, please refer to <i>Assignment of trace information pins between ETM architecture versions</i> on page 119.
19	TDI	Test data input from run control to the JTAG port.
20	Trace signal 10	Trace signal. For more information, please refer to <i>Assignment of trace information pins between ETM architecture versions</i> on page 119.
21	nTRST	Active-low JTAG reset
22	Trace signal 9	Trace signals. For more information, please refer to <i>Assignment of trace information pins between ETM architecture versions</i> on page 119.
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 5: JTAG+Trace connector pinout

## 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 6: Assignment of trace information pins between ETM architecture versions

## TRACE SIGNALS

Data transfer is synchronized by TRACECLK.

### 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.

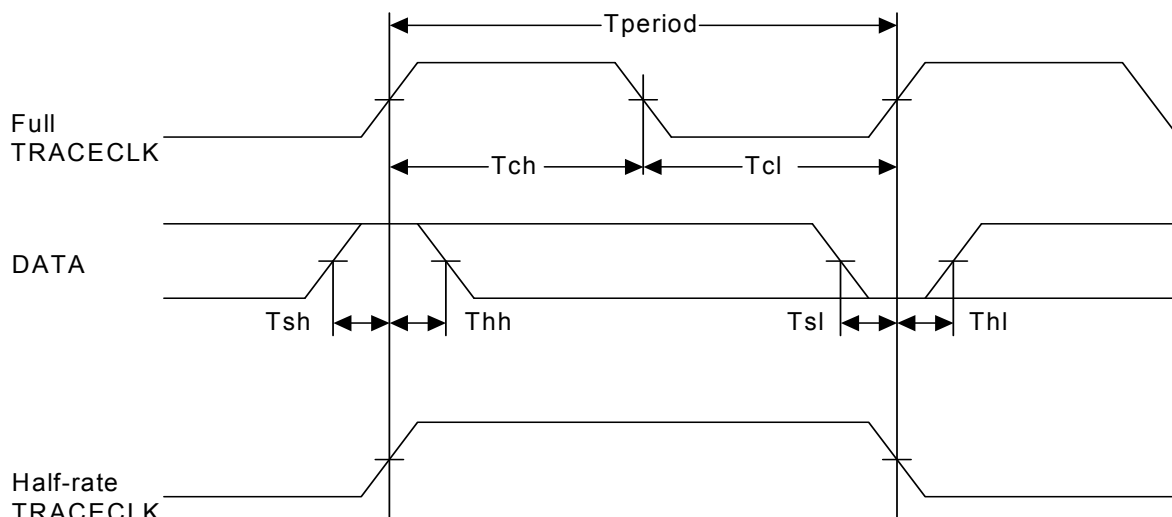
### Clock frequency

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
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 7: Clock frequency

The diagram below shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACECLK.



**Note:**J-Trace supports half-rate clocking mode. Data is output on each edge of the TRACECLK signal and TRACECLK (max) <= 100MHz. For half-rate clocking, the setup and hold times at the JTAG+Trace connector must be observed.

## 19-pin JTAG/SWD and Trace connector

J-Trace provides a JTAG/SWD+Trace connector. This connector is a 19-pin connector. It connects to the target via an 1-1 cable.

VTref	1 ●● 2	SWDIO/TMS
GND	3 ●● 4	SWCLK/TCK
GND	5 ●● 6	SWO/TDO
---	7 ●● 8	TDI
NC	9 ●● 10	nRESET
5V-Supply	11 ●● 12	TRACECLK
5V-Supply	13 ●● 14	TRACEDATA[0]
GND	15 ●● 16	TRACEDATA[1]
GND	17 ●● 18	TRACEDATA[2]
GND	19 ●● 20	TRACEDATA[3]

The following table lists the J-Link / J-Trace SWD pinout.

PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	SWDIO/TMS	I/O / output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU.
4	SWCLK/TCK	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.
6	SWO/TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU. When using SWD, this pin is used as Serial Wire Output trace port. (Optional, not required for SWD communication)
---	---	---	This pin (normally pin 7) is not existent on the 19-pin JTAG/SWD and Trace connector.
8	TDI	Output	JTAG data input of target CPU.- It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU. For CPUs which do not provide TDI (SWD-only devices), this pin is not used. J-Link will ignore the signal on this pin when using SWD.

Table 8: 19-pin JTAG/SWD and Trace pinout



PIN	SIGNAL	TYPE	Description
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 <i>Target power supply</i> on page 121.
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 <i>Target power supply</i> on page 121.
14	TRACEDATA[0]	Input	Input Trace data pin 0.
16	TRACEDATA[1]	Input	Input Trace data pin 0.
18	TRACEDATA[2]	Input	Input Trace data pin 0.
20	TRACEDATA[3]	Input	Input Trace data pin 0.

Table 8: 19-pin JTAG/SWD and Trace pinout

Pins 3, 5, 15, 17, 19 are GND pins connected to GND in J-Trace CM3. They should also be connected to GND in the target system.

## 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.

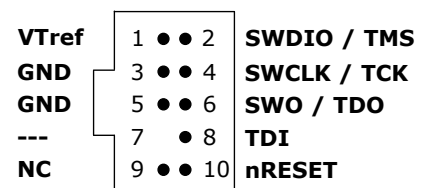
Power can be controlled via the J-Link commander. The following commands are available to control power:

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 9: Command List

## 9-pin JTAG/SWD connector

Some target boards only provide a 9-pin JTAG/SWD connector for Cortex-M. For these devices a 20-pin -> 9-pin Cortex-M adapter is available.



The following table lists the output of the 9-pin Cortex-M connector.

PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	SWDIO/TMS	I/O / output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU. When using SWD, this pin is used as Serial Wire Output trace port. (Optional, not required for SWD communication)
4	SWCLK/TCK	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.

Table 10: 9-pin JTAG/SWD pinout

PIN	SIGNAL	TYPE	Description
6	SWO/TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU.
---	---	---	This pin (normally pin 7) is not existent on the 19-pin JTAG/SWD and Trace connector.
8	TDI	Output	JTAG data input of target CPU.- It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU. For CPUs which do not provide TDI (SWD-only devices), this pin is not used. J-Link will ignore the signal on this pin when using SWD.
9	NC	NC	Not connected inside J-Link. Leave open on target hardware.

*Table 10: 9-pin JTAG/SWD pinout*

Pins 3 and 5 are GND pins connected to GND on the Cortex-M adapter. They should also be connected to GND in the target system.

---

## 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> or <http://www.iar.com/probes>.

# 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)*.

---

## 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.

### 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 1: Test access port

### 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.

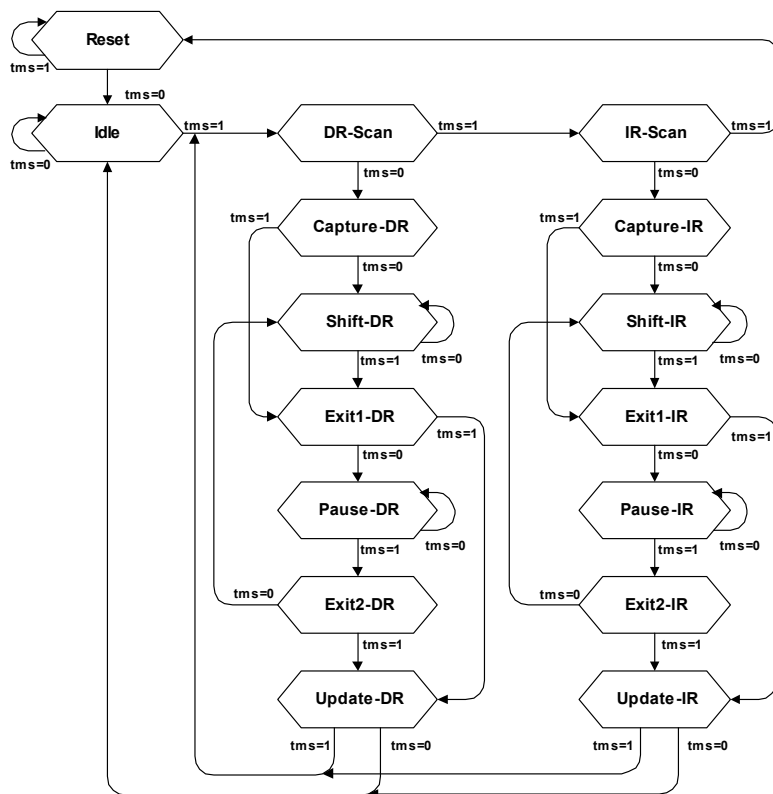
### INSTRUCTION REGISTER

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

# THE TAP CONTROLLER

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



## 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.

---

## 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.

## 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.

## 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.

## 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.

## Code coverage - Disassembly tracing

The screenshot displays the IAR Embedded Workbench IDE interface, showing code coverage and disassembly tracing for the file `stm32f10x_mvc.c`.

**Code Coverage (Left Panel):**

```

193 #ifdef DEBUG
194     debug();
195 #endif
196
197 #ifndef EMB_SPI_SECTION;
198     // Init clock system
199     CLK_Init();
200
201 // NVIC init
202 #ifndef EMB_FLASH
203     // Set the Vector Table base location at 0x20000000
204     NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
205 #else // VECT_TAB_FLASH
206     // Set the Vector Table base location at 0x08000000
207     NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
208 #endif
209     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
210
211 // SysTick end of count event each 0.1s with input clock equal to 9MHz (HCLK/8, default)
212     SysTick_SetReload(900000);
213     // Enable SysTick interrupt
214     SysTick_ITConfig(ENABLE);
215     SysTick_CounterCmd(SysTick_Counter_Enable);
216
217 // Buttons port init
218 // GPIO enable clock and release Reset
219     RCC_APB2PeriphResetCmd(RCC_APB2Periph_GPIOA
220                             | RCC_APB2Periph_GPIOC, DISABLE);
221     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA
222                             | RCC_APB2Periph_GPIOC, ENABLE);
223
224     GPIO_InitStructure.GPIO_Pin = B1_MASK;
225     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
226     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
227     GPIO_Init(B1_PORT, &GPIO_InitStructure);
228
229     GPIO_InitStructure.GPIO_Pin = B2_MASK;
230     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
231     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
232     GPIO_Init(B2_PORT, &GPIO_InitStructure);
233
234     EXT_CR1_SECTION();
235
236 // AN_IR port and ADC init
237     // Enable ADC1 and GPIO clock
238     RCC_APB2PeriphResetCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, DISABLE);
239     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, ENABLE);
  
```

**Disassembly (Right Panel):**

```

003096 003414 0x0800FBCA 4876 LDR R0, [PC, #0x108]
  
```

**ETM Trace (Bottom Panel):**

Index	Frame	Address	Opcode	Trace	Comment
003064	003382	0x0800D89E	E004	B	?NVIC_SetVectorTable_2
003065	003383	0x0800D8AA	4807	LDR R0, [PC, #0x1C]	
003066	003384	0x0800D8AC	4285	CMR R5, R0	
003067	003385	0x0800D8AE	D304	BCC ?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	B031	POP {R0,R4,R5,PC}	
003075	003393	0x0800FBC2	F44F	MOV R0, #0x300	NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
003076	003394	0x0800FBC6	F001	BL NVIC_PriorityGroupConfig	
003077	003395	0x0800D84C	B510	PUSH {R4,LR}	NVIC_PriorityGroupConfig;
003078	003396	0x0800D84E	0004	MOV R4, R0	
003079	003397	0x0800D850	F5B4	CMR R4, #0x700	assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));
003080	003398	0x0800D854	F5B4	CMR R4, #0x600	Not executed
003081	003399	0x0800D856	F5B4	CMR R4, #0x500	Not executed
003082	003400	0x0800D85A	F5B4	CMR R4, #0x400	Not executed
003083	003401	0x0800D85C	F5B4	CMR R4, #0x300	Not executed
003084	003402	0x0800D860	F5B4	CMR R4, #0x200	Not executed
003085	003403	0x0800D862	F5B4	CMR R4, #0x100	Not executed
003086	003404	0x0800D866	F5B4	CMR R4, #0x000	Not executed
003087	003405	0x0800D868	F5B4	CMR R4, #0x300	Not executed
003088	003406	0x0800D86C	F5B4	CMR R4, #0x200	Not executed
003089	003407	0x0800D86E	E004	B	?NVIC_PriorityGroupConfig_0
003090	003408	0x0800D87A	F80F	LDR.W R0, [PC, #0x58]	
003091	003409	0x0800D87E	6900	LDR R0, [R0]	
003092	003410	0x0800D880	4901	LDR R1, [PC, #0x4]	
003093	003411	0x0800D882	4721	ORRS R1, R1, R4	
003094	003412	0x0800D884	60C1	STR R1, [R0, #0xC]	
003095	003413	0x0800D886	B010	POP {R4,PC}	
003096	003414	0x0800FBCA	4876	LDR R0, [PC, #0x108]	SysTick_SetReload(900000);

# Code coverage - Source code tracing

The screenshot displays the IAR Embedded Workbench IDE interface. The main window shows the source code for 'stm32f10x\_mvc.c' with line numbers 193 to 248. The disassembly window on the right shows assembly instructions from 0800FA4 to 0800FCA. The ETM Function Trace window at the bottom shows a list of instructions with columns for Index, Frame, Address, Opcode, Trace, and Comment. The trace shows a sequence of instructions including RCC\_GetFlagStatus, CLK\_Init, and NVIC\_SetPriorityGroupConfig.

Index	Frame	Address	Opcode	Trace	Comment
001268	002686	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002403	002721	0x0800B5E8	2800	CLK_Init() + 66	
002407	002725	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002442	002760	0x0800B5E8	2800	CLK_Init() + 66	
002446	002764	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002481	002799	0x0800B5E8	2800	CLK_Init() + 66	
002485	002803	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002520	002838	0x0800B5E8	2800	CLK_Init() + 66	
002524	002842	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002559	002877	0x0800B5E8	2800	CLK_Init() + 66	
002563	002881	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002598	002916	0x0800B5E8	2800	CLK_Init() + 66	
002602	002920	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002637	002955	0x0800B5E8	2800	CLK_Init() + 66	
002641	002959	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002676	002994	0x0800B5E8	2800	CLK_Init() + 66	
002680	002998	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002715	003033	0x0800B5E8	2800	CLK_Init() + 66	
002719	003037	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002754	003072	0x0800B5E8	2800	CLK_Init() + 66	
002758	003076	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002793	003111	0x0800B5E8	2800	CLK_Init() + 66	
002797	003115	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002832	003150	0x0800B5E8	2800	CLK_Init() + 66	
002836	003154	0x0800B5A4	B510	RCC_GetFlagStatus(u8)	
002871	003189	0x0800B5E8	2800	CLK_Init() + 66	
002875	003193	0x0800B3C8	B510	RCC_USBCLKConfig(u32)	
002883	003201	0x0800B5E8	F44F	CLK_Init() + 76	
002885	003203	0x0800B3E8	B510	RCC_ADCLKConfig(u32)	
002906	003224	0x0800B5E8	2000	CLK_Init() + 84	
002908	003226	0x0800B37C	B510	RCC_PCLK2Config(u32)	
002923	003241	0x0800B5E8	F44F	CLK_Init() + 90	
002925	003243	0x0800B374	B510	RCC_PCLK1Config(u32)	
002942	003260	0x0800B5E8	2000	CLK_Init() + 98	
002944	003262	0x0800B2E4	B510	RCC_HCLKConfig(u32)	
002959	003277	0x0800B5E8	2002	CLK_Init() + 104	
002961	003279	0x0800D00C	B510	FLASH_SetLatency(u32)	
002985	003303	0x0800B5E8	2000	CLK_Init() + 110	
002987	003305	0x0800D746	B510	FLASH_HalfCycleAccessCmd(u32)	
003009	003327	0x0800B5E8	2010	CLK_Init() + 116	
003011	003329	0x0800D77C	B510	FLASH_PrefetchBufferCmd(u32)	
003031	003349	0x0800B5E8	2002	CLK_Init() + 122	
003033	003351	0x0800B2AC	B510	RCC_SysCLKConfig(u32)	
003053	003371	0x0800B5E8	B001	CLK_Init() + 128	
003054	003372	0x0800B098	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	0x0800BFC4	4876	main() + 34	

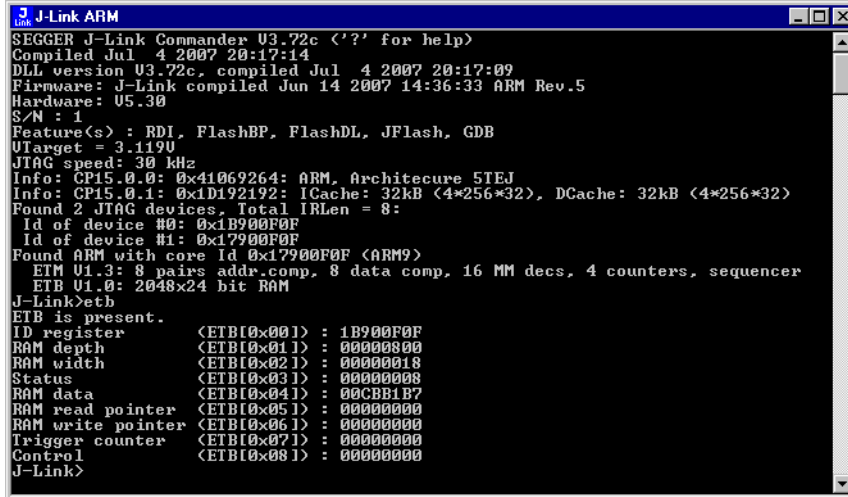




---

## Embedded Trace Buffer (ETB)

The ETB is a small, circular on-chip memory area where trace information is stored during capture. It contains the data which is normally exported immediately after it has been captured from the ETM. The buffer can be read out through the JTAG port of the device once capture has been completed. No additional special trace port is required, so that the ETB can be read via J-Link. The trace functionality via J-Link is limited by the size of the ETB. While capturing runs, the trace information in the buffer will be overwritten every time the buffer size has been reached.



```
J-Link ARM
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
UTarget = 3.119U
JTAG speed: 30 kHz
Info: CP15.0.0: 0x41069264: ARM, Architecture 5TEJ
Info: CP15.0.1: 0xD192192: 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[0x001]) : 1B900F0F
RAM depth       (ETB[0x011]) : 00000000
RAM width       (ETB[0x021]) : 00000018
Status          (ETB[0x031]) : 00000008
RAM data        (ETB[0x041]) : 00CBB1B7
RAM read pointer (ETB[0x051]) : 00000000
RAM write pointer (ETB[0x061]) : 00000000
Trigger counter  (ETB[0x071]) : 00000000
Control         (ETB[0x081]) : 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.

---

## 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.

### 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 endianness 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.

### 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.

## 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.

## 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.

### Flash loader of compiler / debugger vendor such as IAR

IAR Embedded Workbench comes with its 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.

---

## 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.

### 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.

```

C:\> JLink.exe
SEGGER J-Link Commander V2.68.01. '?' for help.
Compiled 14:02:49 on Oct 25 2005.
Updating firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Replacing firmware: J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5
... Firmware update successful. CRC=5EF3
Waiting for new firmware to boot
DLL version V2.70a, compiled Oct 25 2005 14:02:40
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Hardware: U5.00
S/N :
Utarget = 0.0000
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.

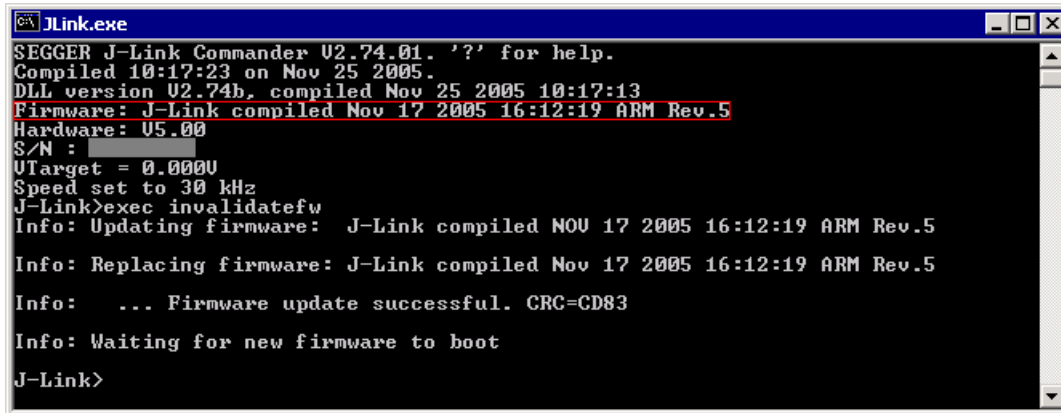
### INVALIDATING THE FIRMWARE

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 downgrade J-Link / J-Trace, you need to invalidate the current J-Link / J-Trace firmware, using the command `exec InvalidateFW`.

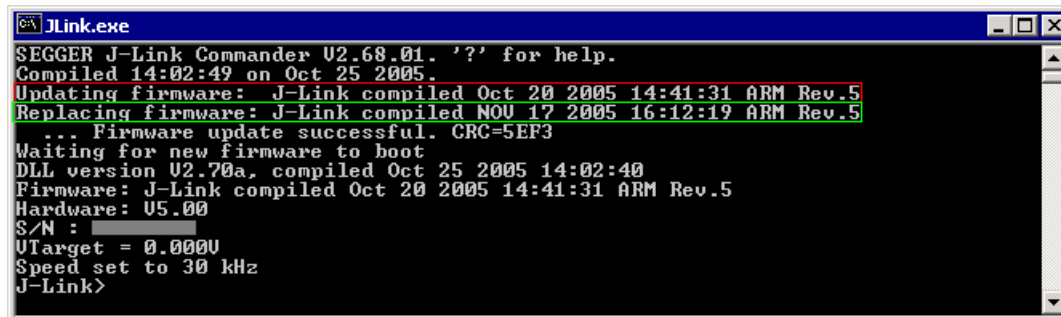


```
JLink.exe
SEGGER J-Link Commander V2.74.01. '?' for help.
Compiled 10:17:23 on Nov 25 2005.
DLL version V2.74b, compiled Nov 25 2005 10:17:13
Firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5
Hardware: U5.00
S/N : 
UTarget = 0.0000
Speed set to 30 kHz
J-Link>exec invalidatefw
Info: Updating 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.



```
JLink.exe
SEGGER J-Link Commander V2.68.01. '?' for help.
Compiled 14:02:49 on Oct 25 2005.
Updating firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Replacing firmware: J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5
... Firmware update successful. CRC=5EF3
Waiting for new firmware to boot
DLL version V2.70a, compiled Oct 25 2005 14:02:40
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Hardware: U5.00
S/N : 
UTarget = 0.0000
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.

# Designing the target board for trace

This chapter describes the hardware requirements which have to be met by the target board.

---

## 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.

### 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.

### 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.

### 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.

### 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 133](#) for further reference.

---

## 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.

## 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.

---

## 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 1: Signal requirements

# 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.

---

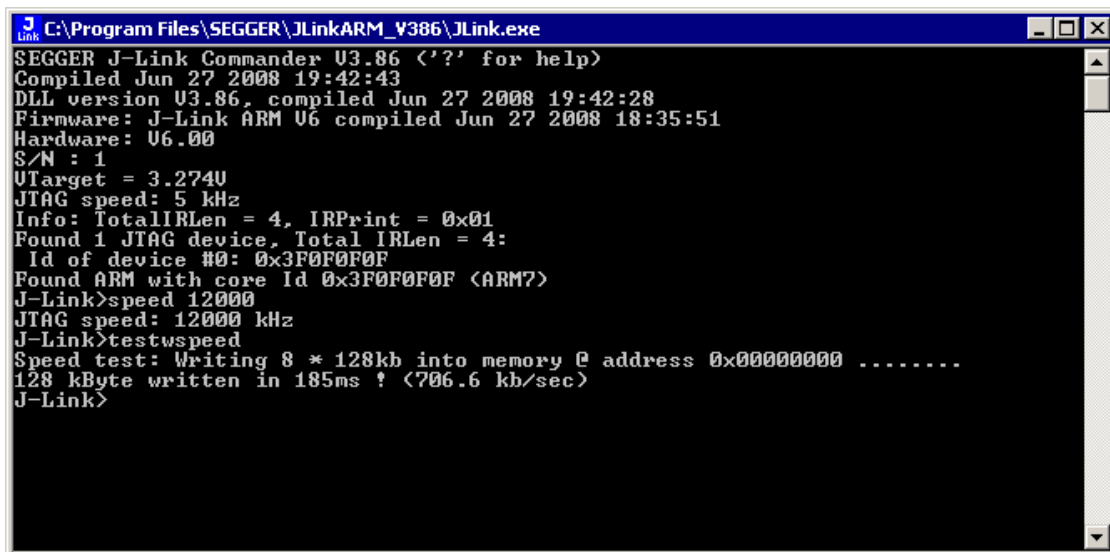
## Measuring download speed

### 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.



```
JLink C:\Program Files\SEGGER\JLinkARM_V386\JLink.exe
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
UTarget = 3.274U
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>
```

---

## Troubleshooting

### 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 `JLink.exe` is unable to read the target processor's core ID you should analyze the communication between your target and J-Link / J-Trace with a logic analyzer or oscilloscope. Follow the instructions in section .

## 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 41.

### 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 135.

---

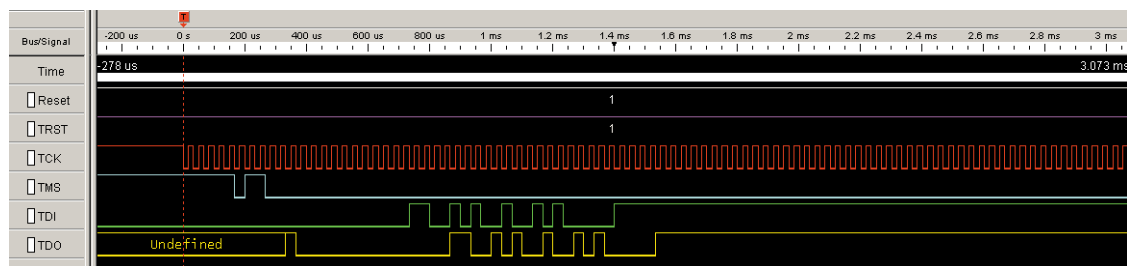
## Signal analysis

The following screenshots show the data flow of the startup and ID communication between J-Link / J-Trace and the target device.



## START SEQUENCE

This is the signal sequence output by J-Link / J-Trace at start of `JLink.exe`. It should be used as reference when tracing potential J-Link / J-Trace related hardware problems.



The sequence consists of the following sections:

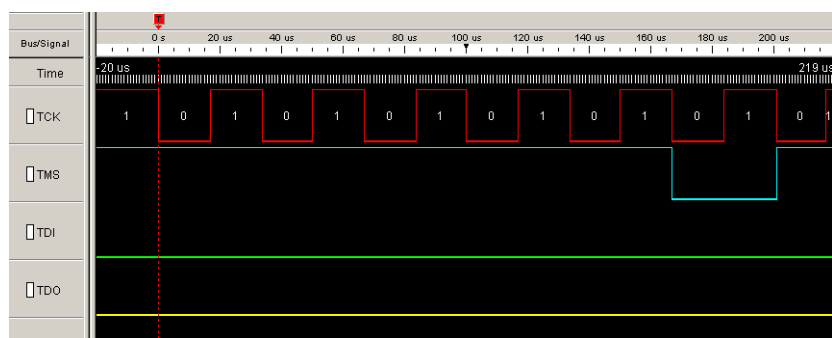
- 5 clocks: TDI low, TMS high. Brings TAP controller into RESET state
- 1 clock: TDI low, TMS low: Brings TAP controller into IDLE state
- 2 clocks: TDI low, TMS high: Brings TAP controller into IR-SCAN state
- 2 clocks: TDI low, TMS low: Brings TAP controller into SHIFT-IR state
- 32 clocks: TMS low, TDI: 0x05253000 (lsb first): J-Link Signature as IR data
- 240 clocks: TMS low, last clock high, TDI high: Bypass command
- 1 clock: TDI low, TMS high: Brings TAP controller into UPDATE-IR state.

J-Link / J-Trace checks the output of the device (output on TDO) for the signature to measure the IR length. For ARM7 / ARM9 chips, the IR length is 4, which means TDO shifts out the data shifted in on TDI with 4 clock cycles delay. If you compare the screenshot with your own measurements, the signals of TCK, TMS, TDI, and TDO should be identical.

Note that the TDO signal is undefined for the first 10 clocks, since the output is usually tristated and the signal level depends on external components connected to TDO, such as pull-up or pull-down.

### Zoom-in

The next screenshot shows the first 6 clock cycles of the screenshot above. For the first 5 clock cycles, TMS is high (Resulting in a TAP reset). TMS changes to low with the falling edge of TCK. At this time the TDI signal is low. Your signals should be identical. Signal rise and fall times should be shorter than 100ns.



## TROUBLESHOOTING

If your measurements of TCK, TMS and TDI (the signals output by J-Link / J-Trace) differ from the results shown, disconnect your target hardware and test the output of TCK, TMS and TDI without a connection to a target, just supplying voltage to J-Link's/J-Trace's JTAG connector: VCC at pin 1; GND at pin 4.

## Contacting support

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section *General procedure* on page 135. 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@iar.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.).

---

## 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 21.

### 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.

# 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 programmer-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.



# 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 ( <a href="http://www.arm.com">www.arm.com</a> ).
[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 ( <a href="http://www.arm.com">www.arm.com</a> ).

Table 1: Literature and References





# Index

- A**
- Adaptive clocking ..... 139
  - Application Program Interface ..... 139
- B**
- Big-endian ..... 139
- C**
- Cache cleaning ..... 139
  - Coprocessor ..... 139
  - copyright notice ..... 2
- D**
- Dirty data ..... 139
  - disclaimer ..... 2
  - Dynamic Linked Library (DLL) ..... 139
- E**
- Embedded Trace Buffer (ETB) ..... 130, 139
  - Embedded Trace Macrocell (ETM) ..... 125, 139
  - EmbeddedICE ..... 139
- H**
- Halfword ..... 139
  - Host ..... 139
- I**
- ICache ..... 139
  - ICE Extension Unit ..... 139
  - ID ..... 139
  - IEEE 1149.1 ..... 140
  - Image ..... 140
  - Instruction Register ..... 140
  - In-Circuit Emulator ..... 140
  - IR ..... 140
- J**
- Joint Test Action Group (JTAG) ..... 140
  - JTAG ..... 123
    - TAP controller ..... 124
  - J-Flash ARM ..... 36
  - J-Link
    - Adapters ..... 122
    - Supported chips ..... 85, 91
  - J-Link Commander ..... 34
  - J-Link STR9 Commander ..... 34
  - J-Mem Memory Viewer ..... 36
- L**
- Little-endian ..... 140
- M**
- Memory coherency ..... 140
  - Memory management unit (MMU) ..... 140
  - Memory Protection Unit (MPU) ..... 140
  - Multi-ICE ..... 140
- N**
- nTRST ..... 113, 140
- O**
- Open collector ..... 140
- P**
- Processor Core ..... 140
  - Program Status Register (PSR) ..... 140
- R**
- registered trademarks ..... 2
  - Remapping ..... 141
  - Remote Debug Interface (RDI) ..... 141
  - RESET ..... 140
  - RTCK ..... 141
  - RTOS ..... 141
- S**
- Scan Chain ..... 141
  - Semihosting ..... 141
  - SetDbgPowerDownOnClose ..... 79
  - SetSysPowerDownOnIdle ..... 80
  - Support ..... 135, 139
  - Supported flash devices ..... 85–87, 92

SWI ..... 141

## T

Tabs ..... 59

TAP Controller ..... 141

Target ..... 141

TCK ..... 113, 141

TDI ..... 113, 141

TDO ..... 113, 141

Test Access Port (TAP) ..... 141

trademarks ..... 2

Transistor-transistor logic (TTL) ..... 141

## W

Watchpoint ..... 141

Word ..... 141