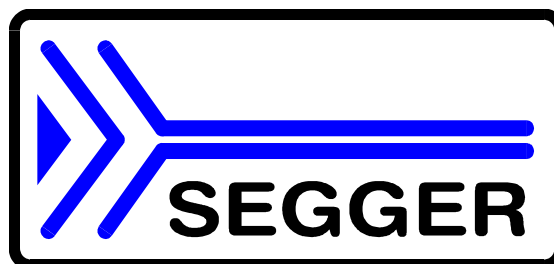


# *J-Link / J-Trace* *User's Guide*

Revision 15



**JTAG Emulators for  
ARM Cores**



**A product of SEGGER Microcontroller Systeme GmbH**

## Disclaimer

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

## Copyright notice

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

© 2006 SEGGER Microcontroller Systeme GmbH, Hilden / Germany

## Trademarks

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

## Contact address

SEGGER Microcontroller Systeme GmbH  
 Heinrich-Hertz-Str. 5  
 D-40721 Hilden  
 Germany  
 Tel. +49 2103-2878-0  
 Fax. +49 2103-2878-28  
 Email: support@segger.com  
 Internet: <http://www.segger.com>

## Revisions

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

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

Revision	Date	By	Explanation
15	060914	OO	Subchapter 1.5.1: Added target supply voltage and target supply current to specifications. Subchapter 5.2.1: Pictures of ways to connect J-Trace.
14	060818	TQ	Subchapter 4.7 "Using DCC for memory reads" added.
13	060711	OO	Subchapter 5.2.2: Corrected JTAG+Trace connector pinout table.
12	060628	OO	Subchapter 4.1: Added ARM966E-S to List of supported ARM cores.
11	060607	SK	Subchapter 5.5.2.2 changed. Subchapter 5.5.2.3 added.
10	060526	SK	ARM9 download speed updated. Subchapter 8.2.1: Screenshot "Start sequence" updated. Subchapter 8.2.2 "ID sequence" removed. Chapter "Support" and "FAQ" merged. Various improvements
9	060324	OO	Chapter "Literature and references" added. Chapter "Hardware": Added common information trace signals. Added timing diagram for trace. Chapter "Designing the target board for trace" added.
8	060117	OO	Chapter "Related Software": Added JLinkARM.dll. Screenshots updated.
7	051208	OO	Chapter Working with J-Link: Sketch added.

Revision	Date	By	Explanation
6	051118	OO	Chapter Working with J-Link: "Connecting multiple J-Links to your PC" added. Chapter Working with J-Link: "Multi core debugging" added. Chapter Background information: "J-Link firmware" added.
5	051103	TQ	Chapter Setup: "JTAG Speed" added.
4	051025	OO	Chapter Background information: "Flash programming" added. Chapter Setup: "Scan chain configuration" added. Some smaller changes.
3	051021	TQ	Performance values updated.
2	051011	TQ	Chapter "Working with J-Link" added.
1	050818	TW	Initial version.



# Table of Contents

1	Introduction .....	9
1.1	About this document .....	10
1.2	Typographic conventions .....	10
1.3	J-Link overview .....	10
1.3.1	Features of J-Link .....	10
1.4	J-Trace overview .....	11
1.4.1	Features of J-Trace .....	11
1.4.2	Test environment .....	12
1.5	Specifications .....	13
1.5.1	Specifications for J-Link .....	13
1.5.2	Specifications for J-Trace .....	13
1.5.3	Download speed .....	13
1.6	Requirements .....	14
2	Setup .....	15
2.1	Installing the J-Link USB driver .....	16
2.1.1	Verifying correct driver installation .....	18
2.2	Uninstalling the J-Link USB driver .....	19
2.3	Connecting the target system .....	21
2.3.1	Power-on Sequence .....	21
2.3.2	Verifying target device connection .....	21
2.3.3	Problems .....	21
2.4	Scan chain configuration .....	22
2.4.1	Sample configuration dialog boxes .....	22
2.4.2	Determining values for scan chain configuration .....	24
2.5	JTAG Speed .....	25
2.5.1	Fixed JTAG speed .....	25
2.5.2	Automatic JTAG speed .....	25
2.5.3	Adaptive clocking .....	25
3	J-Link and J-Trace related software .....	27
3.1	Free software (www.segger.com) .....	28
3.1.1	JLinkARM.dll (J-Link / J-Trace use with third party programs) .....	28
3.1.2	JLink.exe (Command line tool) .....	31
3.1.3	J-Link TCP/IP Server (Remote J-Link / J-Trace use) .....	32
3.1.4	J-Mem Memory Viewer .....	33
3.2	Additional software (Requires license) .....	34
3.2.1	J-Flash ARM (Program flash memory via JTAG) .....	34
3.2.2	RDI Support .....	35
3.3	Additional software packages .....	36
3.3.1	J-Link Software Developer Kit (SDK) .....	36
3.3.2	J-Link Flash Software Developer Kit (SDK) .....	36
3.3.3	JTAGLoad (Command line tool) .....	37
4	Working with J-Link and J-Trace .....	39
4.1	Supported ARM Cores .....	40
4.2	Reset strategies .....	40
4.2.1	Reset strategies in detail .....	40
4.3	Cache handling .....	41

4.3.1	Cache coherency .....	41
4.3.2	Cache clean area .....	41
4.3.3	Cache handling of ARM7 cores .....	41
4.3.4	Cache handling of ARM9 cores .....	41
4.4	Connecting multiple J-Links / J-Traces to your PC .....	42
4.4.1	How does it work? .....	42
4.4.2	Configuring multiple J-Links / J-Traces .....	43
4.4.3	Connecting to a J-Link / J-Trace with non default USB-Address .....	44
4.5	Multi-core debugging .....	45
4.5.1	How multi-core debugging works .....	45
4.5.2	Using multi-core debugging in detail .....	46
4.5.3	Things you should be aware of .....	48
4.6	Multiple devices in the scan chain .....	49
4.6.1	Configuration .....	49
4.7	Using DCC for memory access .....	50
4.7.1	What is required ? .....	50
4.7.2	Target DCC handler .....	50
4.7.3	Target DCC abort handler .....	50
4.7.4	Testing DCC memory access .....	50
<b>5</b>	<b>Hardware .....</b>	<b>53</b>
5.1	JTAG Connector .....	54
5.1.1	Pinout .....	54
5.1.2	Target board design for JTAG .....	55
5.2	JTAG+Trace connector .....	56
5.2.1	Connecting the target board .....	56
5.2.2	Pinout .....	57
5.2.3	Assignment of trace information pins between ETM architecture versions .....	58
5.2.4	Trace signals .....	59
5.3	RESET, nTRST .....	60
5.4	Adapters .....	61
5.4.1	JTAG 14 pin adapter .....	61
5.4.2	5 Volt adapter .....	61
5.5	J-Link hardware versions .....	63
5.5.1	How to determine the hardware version .....	63
5.5.2	Differences between different versions .....	64
5.6	J-Link OEM versions .....	65
5.6.1	List of OEM products .....	65
<b>6</b>	<b>Background information .....</b>	<b>67</b>
6.1	JTAG .....	68
6.1.1	Test access port (TAP) .....	68
6.1.2	Data registers .....	68
6.1.3	Instruction register .....	68
6.1.4	The TAP controller .....	69
6.2	The ARM core .....	71
6.2.1	Processor modes .....	71
6.2.2	Registers of the CPU core .....	71
6.2.3	ARM / Thumb instruction set .....	72
6.3	EmbeddedICE .....	73
6.3.1	Breakpoints and watchpoints .....	73
6.3.2	The ICE registers .....	74
6.4	Flash programming .....	75
6.4.1	How does flash programming via J-Link / J-Trace work ? .....	75
6.4.2	Data download to RAM .....	75
6.4.3	Data download via DCC .....	75
6.4.4	Available options for flash programming .....	75
6.5	J-Link / J-Trace firmware .....	77
6.5.1	Firmware update .....	77

6.5.2	Invalidating the firmware.....	78
7	Designing the target board for trace .....	79
7.1	Overview of high-speed board design .....	80
7.1.1	Avoiding stubs .....	80
7.1.2	Minimizing Signal Skew (Balancing PCB Track Lengths).....	80
7.1.3	Minimizing Crosstalk .....	80
7.1.4	Using impedance matching and termination .....	80
7.2	Terminating the trace signal.....	81
7.2.1	Rules for series terminators .....	81
7.3	Signal requirements.....	81
8	Support and FAQs .....	83
8.1	Troubleshooting .....	84
8.1.1	General procedure .....	84
8.1.2	Typical problem scenarios.....	84
8.2	Signal analysis .....	85
8.2.1	Start sequence.....	85
8.2.2	Troubleshooting .....	86
8.3	Contacting support .....	86
8.4	Frequently Asked Questions .....	87
9	Glossary.....	89
10	Literature and references.....	95





# Chapter 1

## Introduction

---

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

## 1.1 About this document

This document describes J-Link and J-Trace. It provides an overview over the major features of J-Link and J-Trace, gives you some background information about JTAG, ARM and Tracing in general and describes J-Link and J-Trace related software packages available from Segger. Finally, the chapter "Support and FAQs" on page 83 helps to troubleshoot common problems.

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

## 1.2 Typographic conventions

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
Keyword	Text that you enter at the command-prompt or that appears on the display (i.e. system functions, file- or pathnames).

## 1.3 J-Link overview

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

### 1.3.1 Features of J-Link

- USB 2.0 interface
- Any ARM7/ARM9 core supported, including thumb mode
- Automatic core recognition
- Maximum JTAG speed 12 MHz
- Download speed up to 720 Kbytes/second \*
- DCC speed up to 800 Kbytes/second \*
- Seamless integration into the IAR Embedded Workbench® IDE
- No power supply required, powered through USB
- Auto speed recognition
- 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 connector
- Optional 14-pin JTAG adapter available
- Wide target voltage range: 1.2V - 3.3V
- Optional adapter for 5V targets available
- USB and 20-pin ribbon cable included
- Memory viewer (J-Mem) included
- TCP/IP server included, which allows using J-Link via TCP/IP networks
- RDI DLL available, which allows using J-Link with RDI compliant software
- Flash programming software (J-Flash) available
- Flash DLL available, which allows using flash functionality in custom applications
- Software Developer Kit (SDK) available

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

## 1.4 J-Trace overview

J-Trace is a JTAG emulator designed for ARM cores which includes trace (ETM) support. It connects via USB to a PC running Microsoft Windows 2000 or XP. 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.

### 1.4.1 Features of J-Trace

- USB 2.0 interface
- Any ARM7/ARM9 core supported, including thumb mode
- Automatic core recognition
- Maximum JTAG speed 12 MHz
- Download speed up to 420 Kbytes/second \*
- DCC speed up to 600 Kbytes/second \*
- Seamless integration into the IAR Embedded Workbench® IDE
- No power supply required, powered through USB
- Auto speed recognition
- 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 connector, standard 38-pin JTAG+Trace connector
- Optional 14-pin JTAG adapter available
- Wide target voltage range: 3.0V - 3.6V
- Optional adapter for 5V targets available
- USB and 20-pin ribbon cable included
- Memory viewer (J-Mem) included
- TCP/IP server included, which allows using J-Trace via TCP/IP networks
- Flash programming software (J-Flash) available
- Flash DLL available, which allows using flash functionality in custom applications
- Software Developer Kit (SDK) available
- Full integration with the IAR C-SPY® debugger; advanced debugging features available from IAR C-SPY debugger.

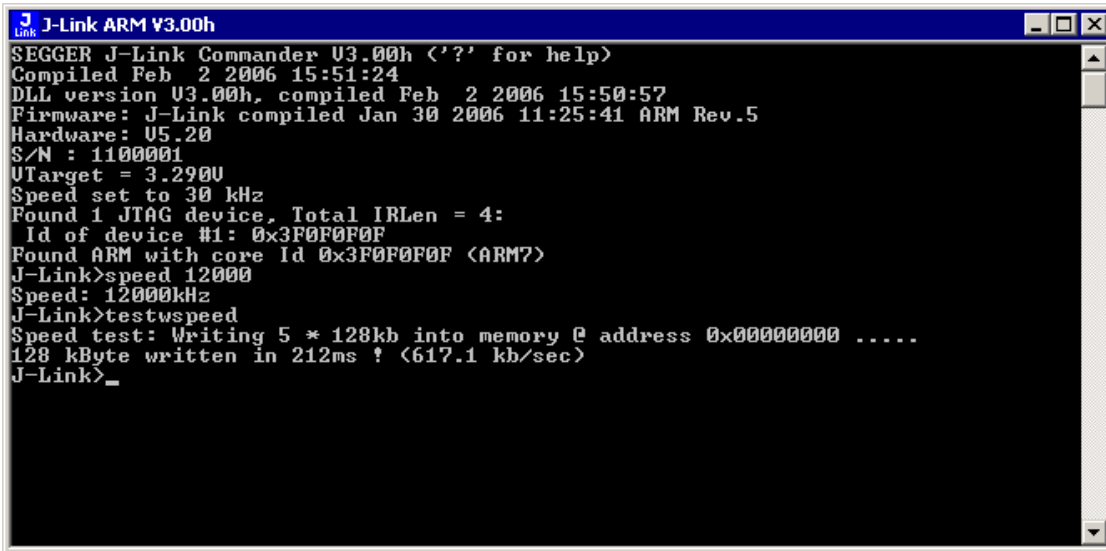
\* = Measured with J-Trace, ARM7 @ 50 MHz, 12MHz JTAG speed.

## 1.4.2 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.



```
J-Link ARM V3.00h
SEGGER J-Link Commander V3.00h ('?' for help)
Compiled Feb  2 2006 15:51:24
DLL version V3.00h, compiled Feb  2 2006 15:50:57
Firmware: J-Link compiled Jan 30 2006 11:25:41 ARM Rev.5
Hardware: V5.20
S/N : 1100001
Utarget = 3.290U
Speed set to 30 kHz
Found 1 JTAG device, Total IRLen = 4:
  Id of device #1: 0x3F0F0F0F
Found ARM with core Id 0x3F0F0F0F (ARM7)
J-Link>speed 12000
Speed: 12000kHz
J-Link>testwspeed
Speed test: Writing 5 * 128kb into memory @ address 0x00000000 .....
128 kByte written in 212ms ! (617.1 kb/sec)
J-Link>_
```

## 1.5 Specifications

### 1.5.1 Specifications for J-Link

Power Supply	USB powered <50mA
USB Interface	USB 2.0, full speed
Target Interface	JTAG 20-pin (14-pin adapter available)
Serial Transfer Rate between J-Link and Target	up to 12 MHz
Supported Target Voltage	1.2 - 3.3 V (5V adapter available)
Target supply voltage	4.5V .. 5V (if powered with 5V on USB)
Target supply current	Max. 300mA
Operating Temperature	+5°C ... +60°C
Storage Temperature	-20°C ... +65 °C
Relative Humidity (non-condensing)	<90% rH
Size (without cables)	100mm x 53mm x 27mm
Weight (without cables)	70g
Electromagnetic Compatibility (EMC)	EN 55022, EN 55024
Supported OS	Microsoft Windows 2000/XP

### 1.5.2 Specifications for J-Trace

Power Supply	USB powered < 300mA
USB Interface	USB 2.0, full speed
Target Interface	JTAG 20-pin (14-pin adapter available) JTAG+Trace: Mictor, 38-pin
Serial Transfer Rate between J-Trace and Target	up to 12 MHz
Supported Target Voltage	3.0 - 3.6 V (5V adapter available)
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
Electromagnetic Compatibility (EMC)	EN 55022, EN 55024
Supported OS	Microsoft Windows 2000/XP

### 1.5.3 Download speed

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

Hardware	Memory download via <i>DCC</i>	ARM7 <i>Memory download</i>	ARM9 <i>Memory download</i>
J-Link Rev. 1 – 4	185.0 Kbyte/s (4MHz JTAG)	150.0 Kbyte/s (4MHz JTAG)	75.0 Kbyte/s (4MHz JTAG)
J-Link Rev. 5	800.0 Kbyte/s (12MHz JTAG)	720.0 Kbyte/s (12MHz JTAG)	550.0 Kbyte/s (12MHz JTAG)
J-Trace Rev.1	600.0 Kbyte/s (12MHz JTAG)	420.0 Kbyte/s (12MHz JTAG)	280.0 Kbyte/s (12MHz JTAG)

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

## 1.6 Requirements

### Host System

In order to use J-Link or J-Trace you need a host system running Windows 2000 or Windows XP.

### Target System

An ARM7 or ARM9 target system is required. The system should have a 20-pin connector as defined by ARM Ltd. for a simple JTAG connection. The individual pins are described in section "JTAG Connector" on page 54. Note that Segger offers an optional adapter to use J-Link and J-Trace with targets using 14 pin 0.1" mating JTAG connectors.

To use tracing with J-Trace, you need a 38-pin connector on your target board as defined by ARM Ltd. and described under "[JTAG+Trace connector" on page 56](#). The individual pins are described in section "[Pinout" on page 57](#)."

# Chapter 2

## Setup

---

This chapter describes the setup procedure required in order to work with J-Link / J-Trace. This includes primarily the installation of a kernel mode J-Link USB driver in your host system.

## 2.1 Installing the J-Link USB driver

To install the J-Link USB driver, follow this procedure.

1. When your J-Link / J-Trace is plugged into your computer's USB port, or when the computer is first powered on after connecting J-Link / J-Trace, Windows will detect the new hardware. The wizard starts the installation of the driver.

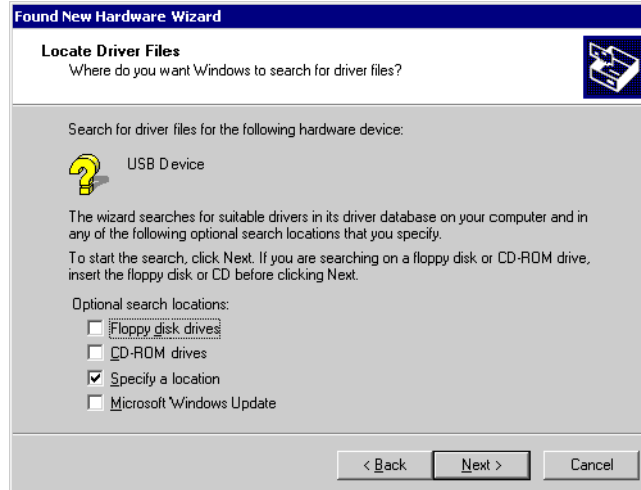


2. First select the Search for a suitable driver for my device (recommended) option, then click the Next > button.

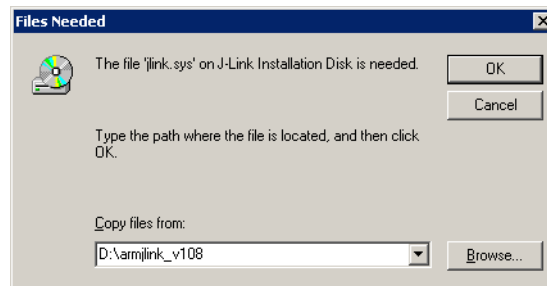




3. In the next step, you need to select the `Specify a location` option, and click the `Next >` button.



4. The wizard will ask you to specify the location of the correct driver files for the new device. Use the directory navigator to select `D:\armjlink_v108\` (or your chosen location) and confirm with a click on the `OK` button.



The wizard confirms your choice and starts to copy, when you click the `Next >` button.



At this point, the installation is complete. Click the `Finish` button to dismiss the installation.

## 2.1.1 Verifying correct driver installation

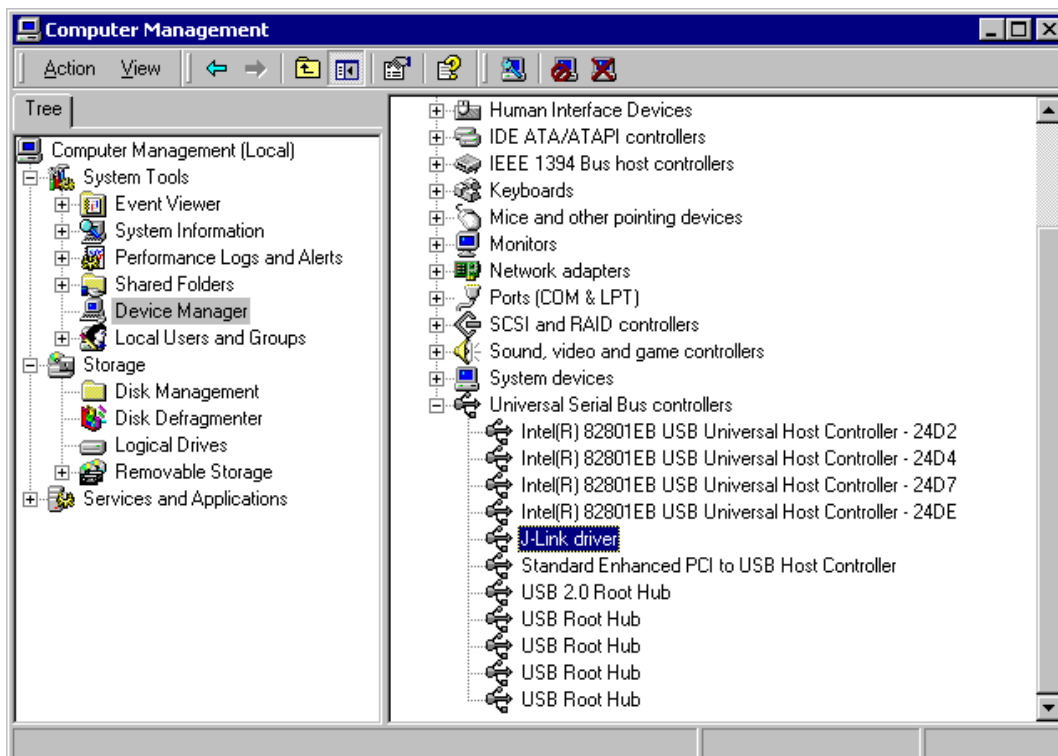
To verify the correct installation of the driver, disconnect and reconnect J-Link / J-Trace to the USB port. During the enumeration process which takes about 2 seconds, the LED on J-Link / J-Trace is flashing. After successful enumeration, the LED stays on permanently.

Start the provided sample application `JLink.exe`, which should display the compilation time of the J-Link firmware, the serial number, a target voltage of 0.000V if a target is not connected to J-Link / J-Trace, and the speed selection. The screenshot below should give you an idea.

```

JLink.exe
SEGGER J-Link Commander V2.66.02. '?' for help.
Compiled 17:40:01 on Aug  8 2005.
DLL version V2.66b, compiled Aug  8 2005 17:39:51
Firmware: J-Link compiled Jul 25 2005 17:16:33 -- Update --
S/N : 
Utarget = 0.0000
Speed set to 30 kHz
J-Link>
  
```

In addition to this you may 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 new dialog box is opened and should report: This device is working properly. If you experience problems, have a look at chapter "Support and FAQs" on page 83 for help.

## 2.2 Uninstalling the J-Link USB driver

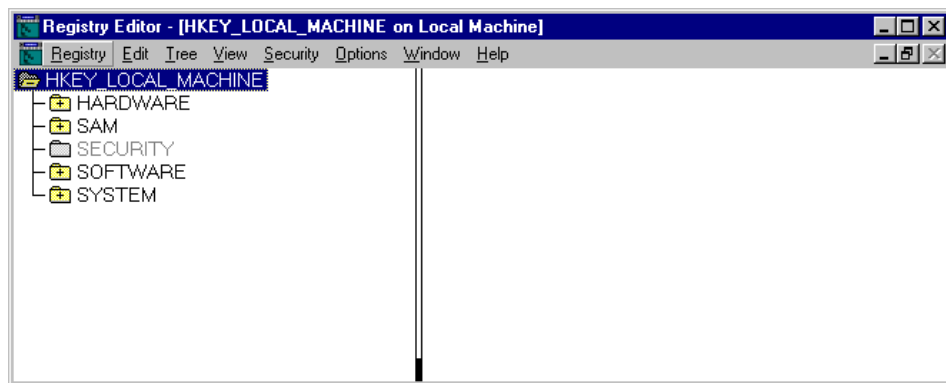
If J-Link / J-Trace is not properly recognized by windows and therefore does not enumerate, it make sense to uninstall the J-Link USB driver.

Indicators for this behavior are:

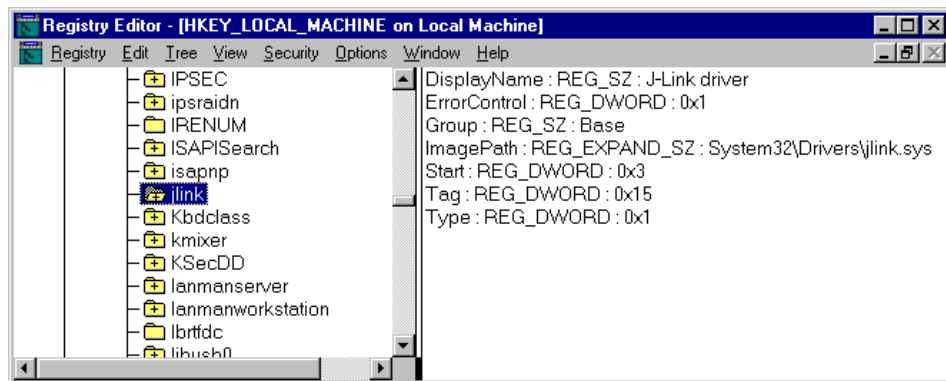
- The LED on the J-Link / J-Trace is rapidly flashing.
- The J-Link / J-Trace is recognized as Unknown Device by Windows.

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

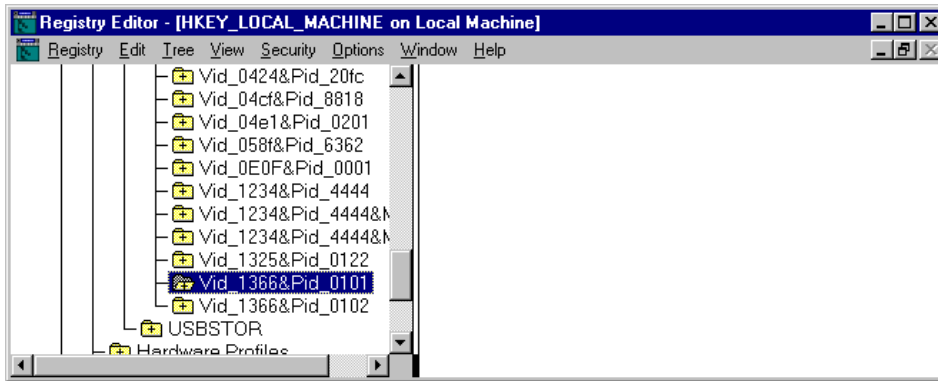
1. Disconnect J-Link / J-Trace from PC.
2. Delete JLink.inf and JLink.PNF from the \Windows\Inf or \Winnt\Inf directory.
3. Delete JLink.Sys from the \Windows\System32\Drivers or \Winnt\System32\Drivers directory.
4. Click the Start button in the taskbar and select Run... to run a command.
5. Enter regedt32 in the open edit field. The Registry Editor will open.
6. Select HKEY\_LOCAL\_MACHINE window.



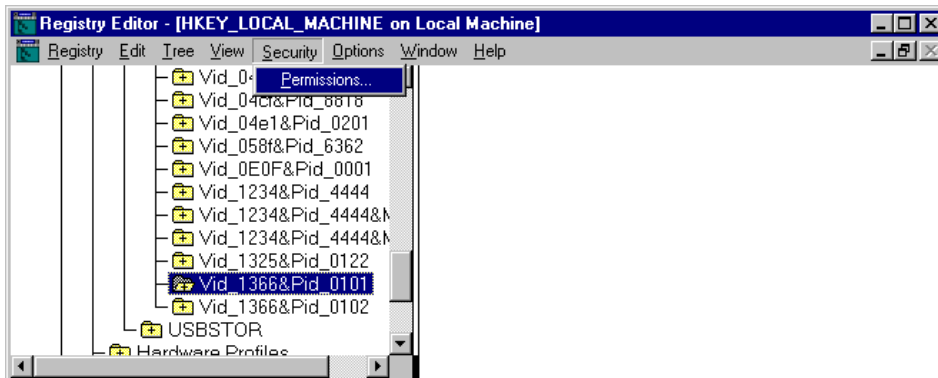
7. Select System\CurrentControlSet\Services\Jlink and delete the Jlink branch.



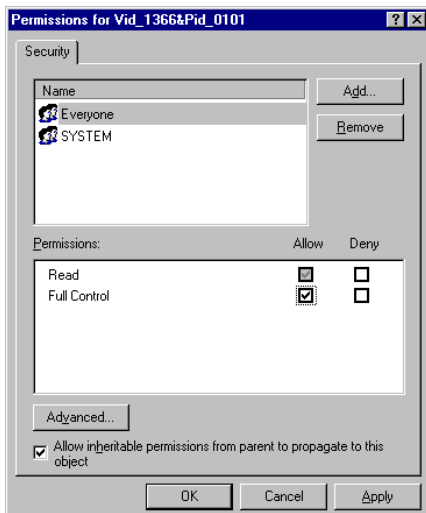
8. Select `System\CurrentControlSet\Enum\USB\Vid_1366&Pid_0101`.



9. Open the Security menu and choose Permission



10. Give group Everyone full control by clicking the checkbox Allow on Full Control.



11. Save the changes by clicking OK.  
 12. Now delete the `Vid_1366&Pid_0101` branch.  
 13. If the following branch  
`System\CurrentControlSet\Enum\USB\Vid_0000&Pid_0000`  
 is also available. Repeat the step that was made for  
`System\CurrentControlSet\Enum\USB\Vid_1366&Pid_0101`.  
 14. Close the Registry Editor window and restart the computer.

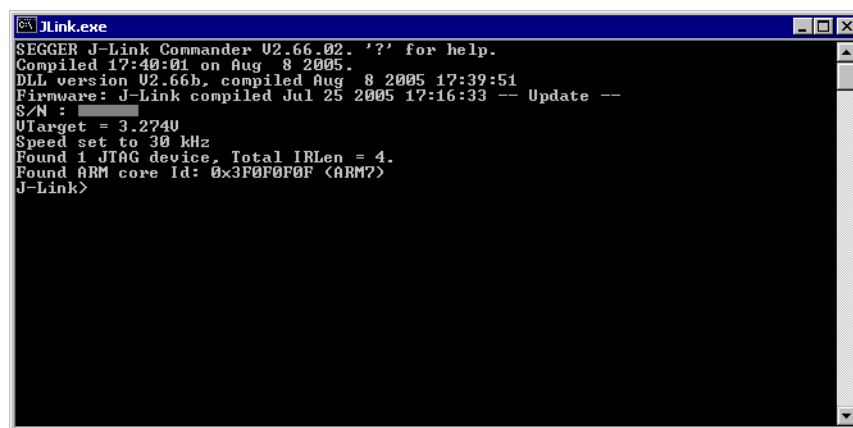
## 2.3 Connecting the target system

### 2.3.1 Power-on Sequence

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

### 2.3.2 Verifying target device connection

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



```

SEGGER J-Link Commander V2.66.02. '?' for help.
Compiled 17:40:01 on Aug  8 2005.
DLL version V2.66b, compiled Aug  8 2005 17:39:51
Firmware: J-Link compiled Jul 25 2005 17:16:33 -- Update --
S/N :
UTarget = 3.274U
Speed set to 30 kHz
Found 1 JTAG device, Total IRLen = 4.
Found ARM core Id: 0x3F0F0F0F <ARM7>
J-Link>
  
```

### 2.3.3 Problems

If you experience problems with any of the steps described above, it is recommended you read the chapter "Support and FAQs" on page 83 for troubleshooting tips. If you still do not find appropriate help there and your J-Link / J-Trace is an original Segger product, you may contact Segger support via e-mail. Provide the necessary information about your target processor, board etc. and we will try to solve your problem. A checklist of the required information together with the contact information can be found in chapter "Support and FAQs" on page 83 as well.

## 2.4 Scan chain configuration

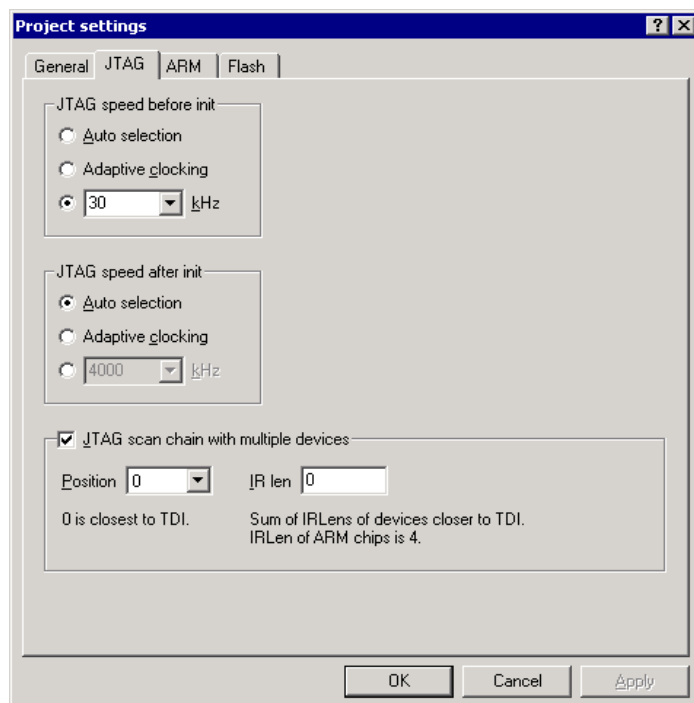
By default, only one ARM device is assumed to be in the JTAG scan chain. If you have multiple devices in the scan chain, you must properly configure it. To do so, you have to specify the exact position of the ARM device that should be addressed. Configuration of the scan is done by the target application.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

### 2.4.1 Sample configuration dialog boxes

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

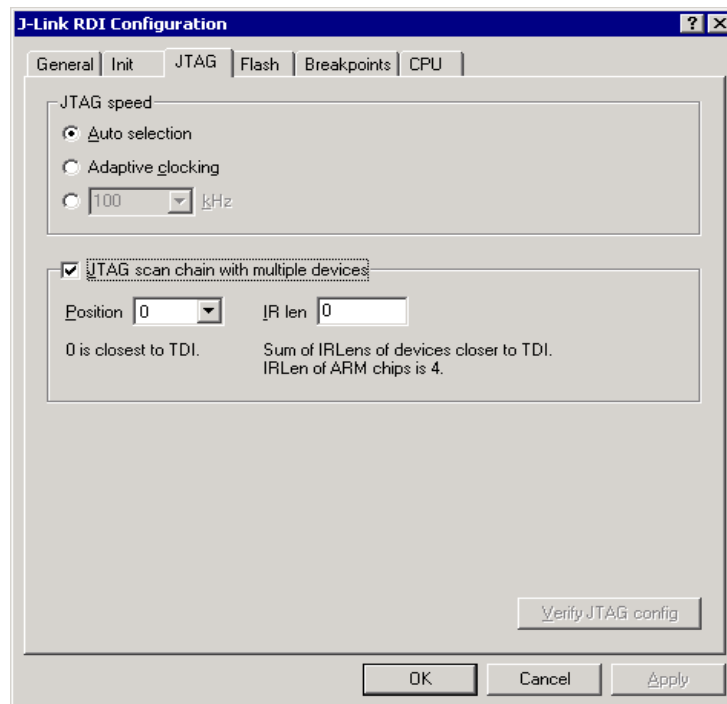
#### SEGGER J-Flash configuration dialog

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



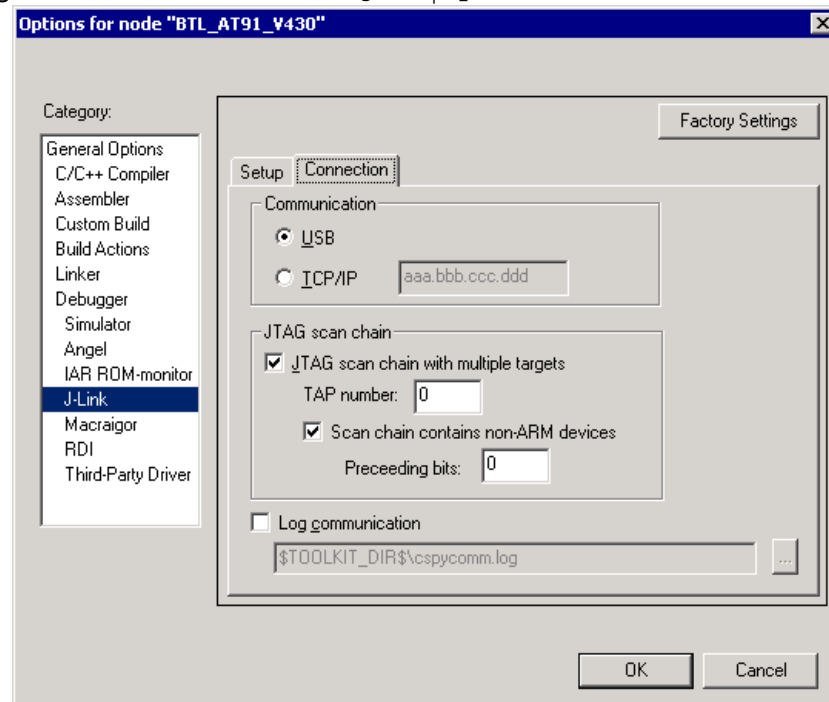
## SEGGER J-Link RDI configuration dialog box

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



## IAR J-Link configuration dialog box

This dialog can be found under **Project | Options**.



## 2.4.2 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:

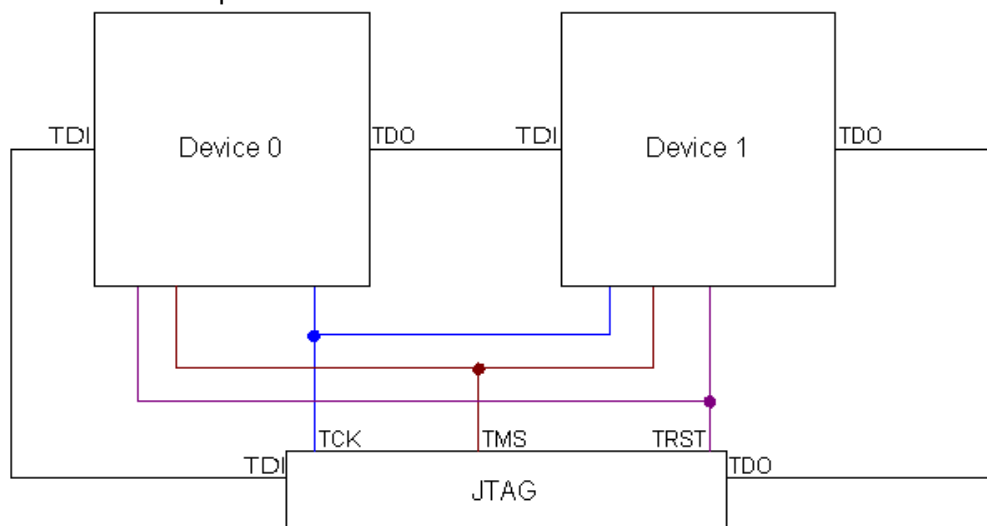
- The position of the target device in the scan chain
- The total number of bits in the instruction registers of the devices before the target device (IR len).

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

The target device is marked in blue.



## 2.5 JTAG Speed

There are basically three types of speed settings:

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

These are explained below.

### 2.5.1 Fixed JTAG speed

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

### 2.5.2 Automatic JTAG speed

Selects the maximum JTAG speed handled by the TAP controller.

**NOTE:**

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

### 2.5.3 Adaptive clocking

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

**NOTE:**

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



# Chapter 3

## J-Link and J-Trace related software

---

This chapter describes Segger's J-Link / J-Trace related software portfolio. The table below lists the available software packages.

Software	Description
JLinkARM.dll	DLL to use J-Link / J-Trace with third-party programs.
JLink.exe	Free command-line tool with basic functionality for target analysis.
J-Link TCP/IP Server	Free utility which provides the possibility to use J-Link / J-Trace remotely via TCP/IP.
J-Mem memory viewer	Free target memory viewer. Shows the memory content of a running target and allows editing as well.
J-Flash	Stand-alone flash programming application.
RDI support with Flash download and Flash breakpoints.	Provides Remote Debug Interface (RDI) support. Flash breakpoints provide the ability to set an unlimited number of software breakpoints in flash memory areas. Flash download allows an arbitrary debugger to write into flash memory.
J-Link Software Developer Kit (SDK)	The J-Link Software Developer Kit is needed if you want to write your own program with J-Link / J-Trace.
J-Link Flash Software Developer Kit (SDK)	An enhanced version of the JLinkARM.DLL, which contains additional API functions for flash programming.
JTAGLoad	Command line tool that opens an svf file and sends the data in it via J-Link / J-Trace to the target.

## 3.1 Free software ([www.segger.com](http://www.segger.com))

This kind of J-Link / J-Trace related software is shipped together with J-Link / J-Trace and may also be downloaded from [www.segger.com](http://www.segger.com). No additional license is required to use this software.

### 3.1.1 JLinkARM.dll (J-Link / J-Trace use with third party programs)

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

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

#### 3.1.1.1 Updating the DLL in third-party programs

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

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

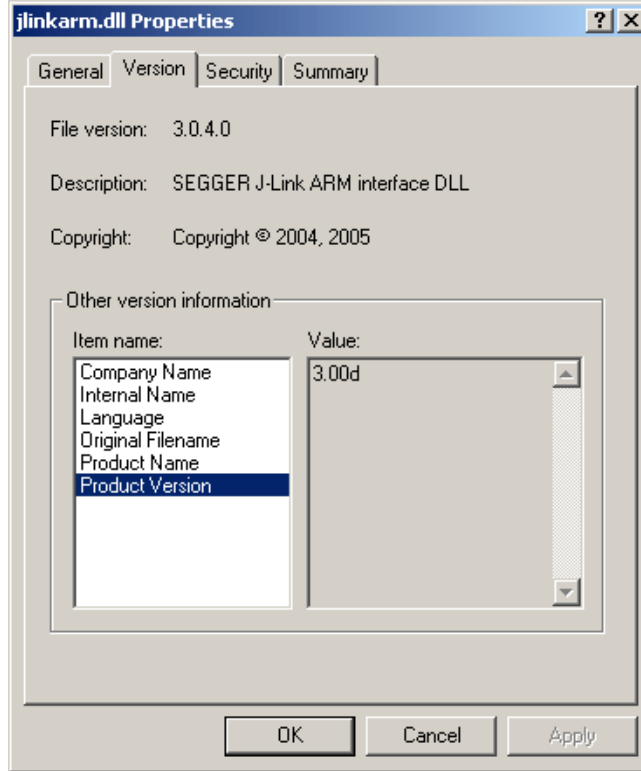
IAR Embedded Workbench is a high performance integrated development environment with editor, compiler, linker, debugger. The compiler generates very efficient code and is widely used. It comes with the J-LinkARM.dll in the arm\bin subdirectory of the installation directory. To update this DLL, you should 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 4.0\arm\bin\`.

After updating the DLL, it is advisable to verify that the new DLL is loaded as described in "Determining which DLL is used by a program" on page 30.

### 3.1.1.3 Determining the version of JLinkArm.dll

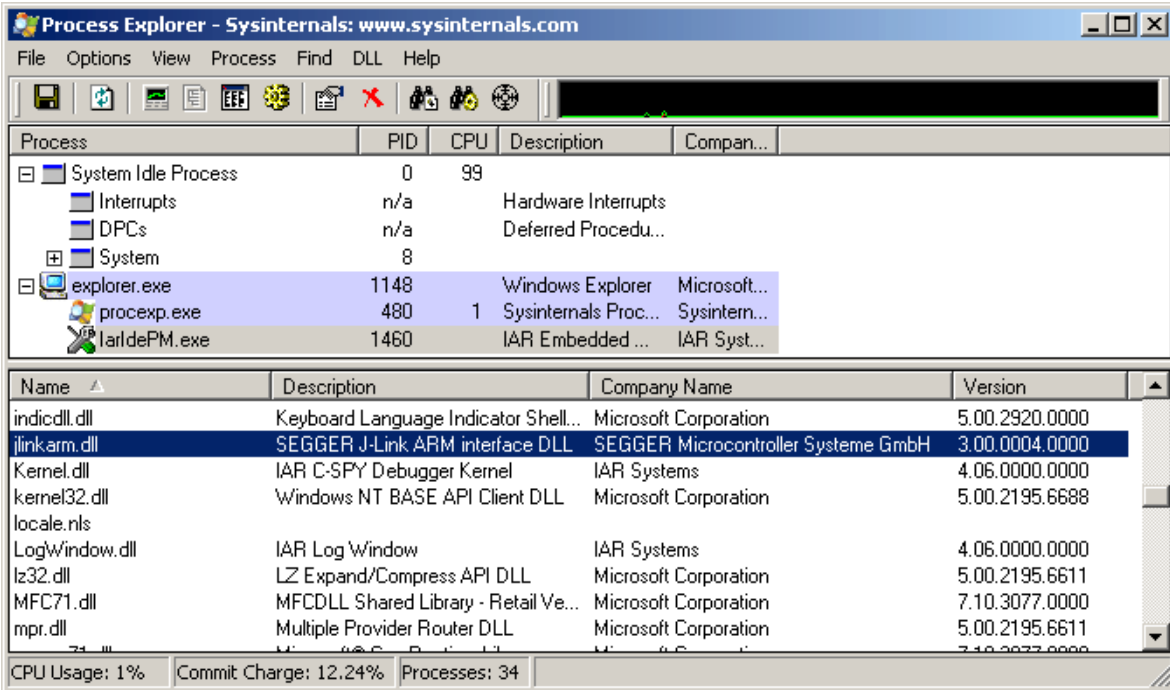
To determine which version of the JLinkARM.dll you are facing, the DLL version can be viewed by right clicking the DLL in explorer, and choosing Properties from the context menu. Click the Version tab to display information about the Product Version.



### 3.1.1.4 Determining which DLL is used by a program

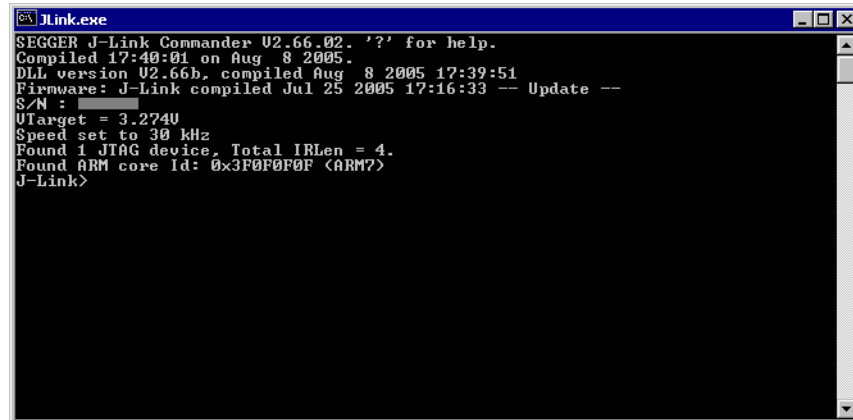
To verify that the program you are working with is using the DLL you expect it to use, you can investigate which DLLs are loaded by your program with tools like Sysinternals' Process Explorer. It shows you details about the DLLs, used by your program, such as manufacturer and version.

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



### 3.1.2 JLink.exe (Command line tool)

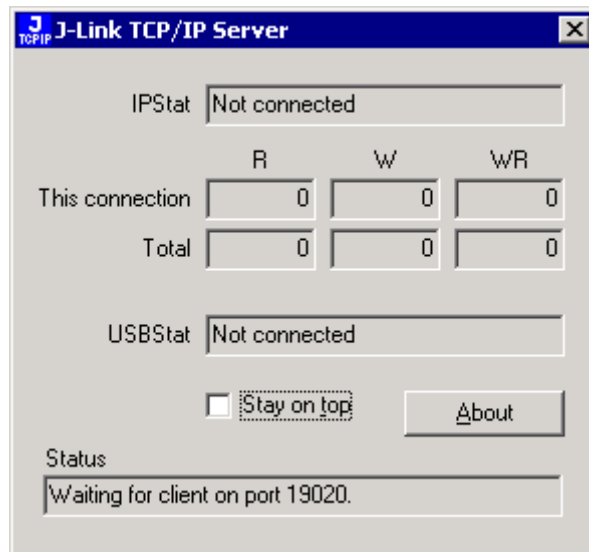
JLink.exe is a tool that can be used to verify proper installation of the USB driver and to verify the connection to the ARM chip, as well as for simple analysis of the target system. It permits some simple commands, such as memory dump, halt, step, go and ID-check, as well as some more in-depths analysis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



```
JLink.exe
SEGGER J-Link Commander V2.66.002. '?' for help.
Compiled 17:40:01 on Aug  8 2005.
DLL version V2.66b, compiled Aug  8 2005 17:39:51
Firmware: J-Link compiled Jul 25 2005 17:16:33 -- Update --
S/N :
Utarget = 3.274U
Speed set to 30 kHz
Found 1 JTAG device, Total IRLen = 4.
Found ARM core Id: 0x3F0F0F0F <ARM7>
J-Link>
```

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

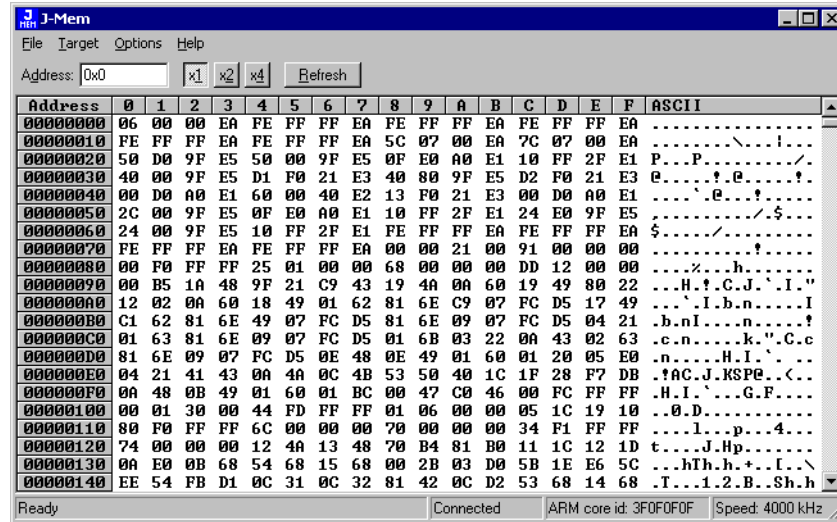
The J-Link TCP/IP server allows using J-Link / J-Trace remotely via TCP/IP. This enables you to connect to and fully use a J-Link / J-Trace from another computer. Performance is just slightly (about 10%) lower than with direct USB connection.





### 3.1.4 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. The type of access for both read and write access can be selected to be 8/16/32 bit. It works nicely when modifying SFRs, especially because it writes the SFR only after the complete value has been entered.

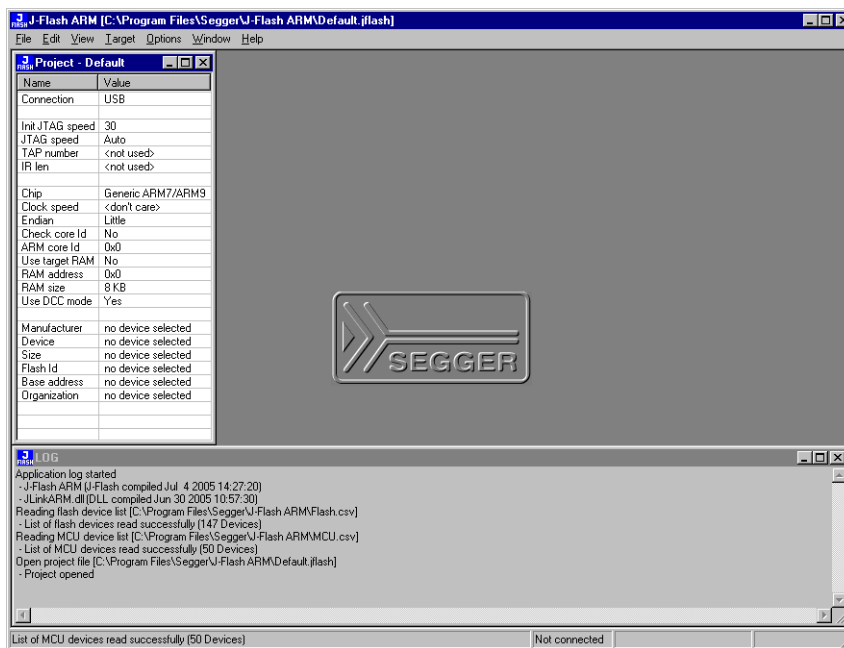


## 3.2 Additional software (Requires license)

The software described in this section requires separate licenses from Segger. You can download the software from [www.segger.com](http://www.segger.com). Evaluation licenses are available free of charge. For further information go to our website or contact us directly.

### 3.2.1 J-Flash ARM (Program flash memory via JTAG)

J-Flash ARM is a software running on Windows 2000/XP 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. Even without a license key you can still use J-Flash ARM to open project files, read from connected devices, blank check target memory, verify data files and so on. However, to actually program devices via J-Flash ARM and J-Link / J-Trace you are required to obtain a license key from us.



#### Features

- Works with any ARM7/ARM9 chip
- ARM microcontrollers (internal flash) supported
- Most external flash chips can be programmed
- High-speed programming: up to 200 Kbytes/second (depends on flash device)
- Very high-speed blank check: Approximately 16 Mbytes/sec (depends on target)
- Smart read-back: Only non-blank portions of flash transferred and saved
- Easy to use, comes with projects for standard eval boards.

## 3.2.2 RDI Support

The J-Link RDI software is an RDI interface for J-Link. It makes it possible to use J-Link with any RDI compliant debugger. The main part of the software is an RDI-compliant DLL, which needs to be selected in the debugger. There are two additional features available which build on the RDI software foundation. Each additional features requires an RDI license in addition to its own license.

### 3.2.2.1 Flash download

The J-Link RDI software contains a flash loader. On top of that, the flash download software lets flash memory appear as RAM to a debugger. This enables you to use any arbitrary debugger which normally only allows downloads into RAM to work with flash memory as well. 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.

The advantage you gain is the possibility to transparently use your toolchain of choice—that may not contain a flash loader—with flash memory that can be programmed as if it were RAM.

### 3.2.2.2 Flash breakpoints

The J-Link RDI software contains an additional feature, called Flash breakpoints (short FlashBPs). This feature requires an additional license. It adds the ability to set an unlimited number of software breakpoints in flash memory areas, rather than just the 2 hardware breakpoints permitted by the ICE. Setting the breakpoints in flash is executed very fast using a RAM code specifically designed for this purpose; on chips with fast flash, the difference between breakpoints in RAM and flash is unnoticeable.

#### How do breakpoints work?

ARM7 and ARM9 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 by replacing the instruction where the breakpoint is set with a special value. Additional software breakpoints do not require additional hardware units in the processor, since simply more instructions are replaced. This is a standard procedure that most debuggers are capable of, however, it requires the program to be located in RAM.

#### What is special about software breakpoints in flash?

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

#### How does this work?

Basically, it is very simple the J-Link RDI software reprograms a sector of the flash to set or clear a breakpoint.

#### What performance can I expect?

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

### How is this performance achieved?

We have put a lot of effort in making FlashBPs 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 software and hardware breakpoints allows us to use hardware breakpoints a lot of times, especially when the debugger is source-level stepping, avoiding reprogramming of 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, maximizing the life time of the flash. All resources of the ARM micro are available to the application program, no memory is lost for debugging. All of the optimizations described above can be disabled.

## 3.3 Additional software packages

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

### 3.3.1 J-Link Software Developer Kit (SDK)

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

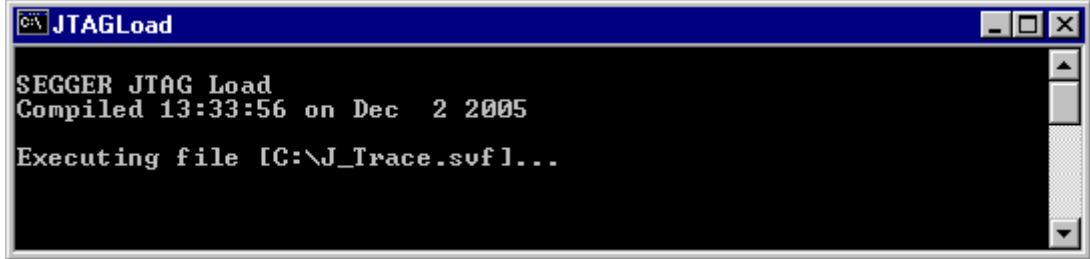
Files	Contents
GLOBAL.h JLinkARMDLL.h	Header files that must be included to use the DLL functions. These files contain the defines, typedef names and function declarations.
JLinkARM.lib	Library contains the exports of the JLinkDLL.
JLinkARM.dll	The DLL itself.
Main.c	Sample application, which calls some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

### 3.3.2 J-Link Flash Software Developer Kit (SDK)

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

### 3.3.3 JTAGLoad (Command line tool)

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





# Chapter 4

## Working with J-Link and J-Trace

---

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

## 4.1 Supported ARM Cores

J-Link / J-Trace has been tested with the following cores, but should work with any ARM7/ARM9 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

## 4.2 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.

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

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

### 4.2.1 Reset strategies in detail

#### 4.2.1.1 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.

#### 4.2.1.2 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.



### 4.2.1.3 Software, for Analog Devices ADuC7xxx MCUs

The following sequence is executed:

- The CPU is halted
- A software reset sequence is downloaded to RAM
- A breakpoint at address 0 is set
- The software reset sequence is executed.

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

## 4.3 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.

### 4.3.1 Cache coherency

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

ARM 9 chips have no hardware to keep the caches coherent, so that this is the responsibility of the software.

### 4.3.2 Cache clean area

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

### 4.3.3 Cache handling of ARM7 cores

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

### 4.3.4 Cache handling of ARM9 cores

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

#### When entering debug state

J-Link / J-Trace performs the following:

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

#### When leaving debug state

J-Link / J-Trace performs the following:

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

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

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

You can connect up to 4 J-Links / J-Traces to your PC. In this case, all J-Links / J-Traces must have different USB-addresses. The default USB-address is 0.

In order to do this, 3 J-Links / J-Traces must be configured as described below. Every J-Link / J-Trace need its own J-Link USB driver which can be downloaded from [www.segger.com](http://www.segger.com).

This feature is supported by J-Link Rev. 5.0 and up and by J-Trace.

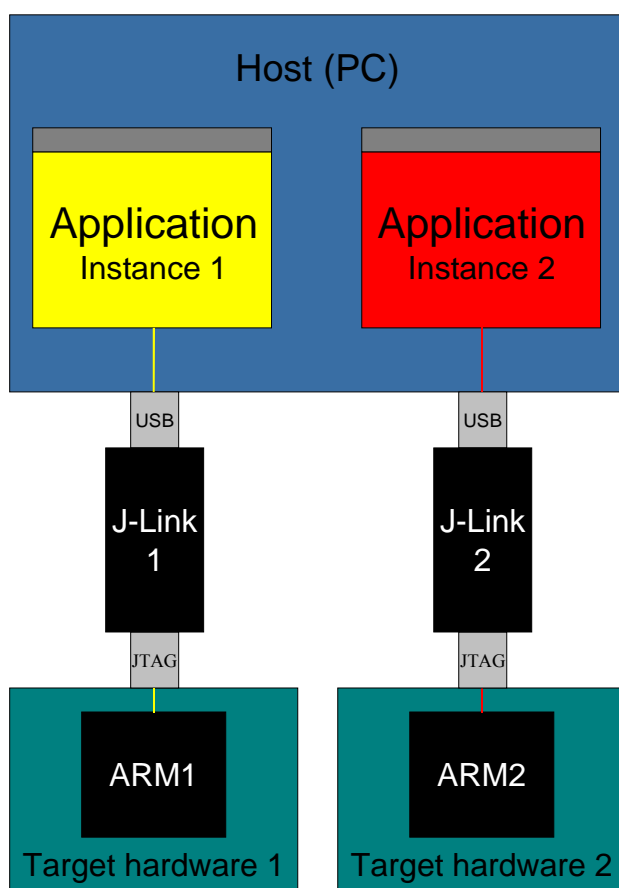
### 4.4.1 How does it work?

USB devices are identified by the OS by their product id, vendor id and serial number. The serial number reported by J-Links / J-Traces is always the same. The product id depends on the configured USB-address.

- The vendor id (VID) representing SEGGER is always 1366
- The product id (PID) for J-Link / J-Trace #1 is 101
- The product id (PID) for J-Link / J-Trace #2 is 102 and so on.

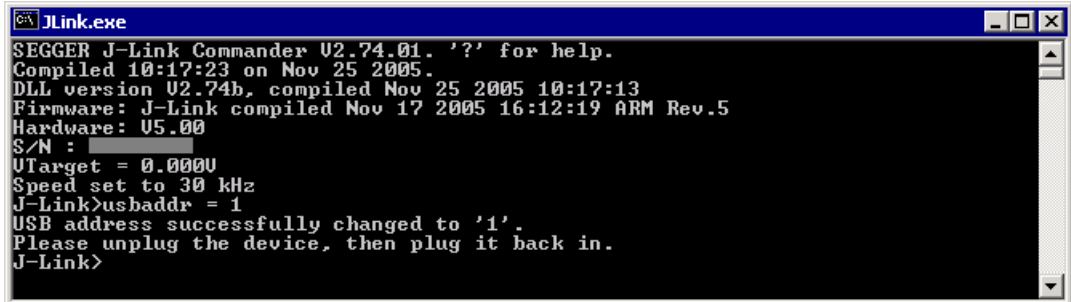
A different PID means that J-Link / J-Trace is identified as a different device, requiring a new driver.

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



## 4.4.2 Configuring multiple J-Links / J-Traces

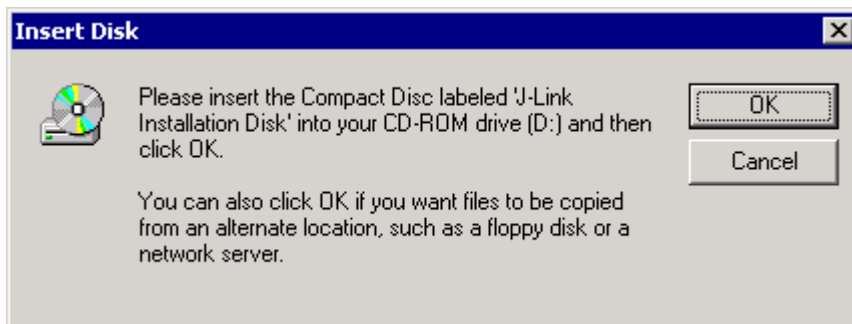
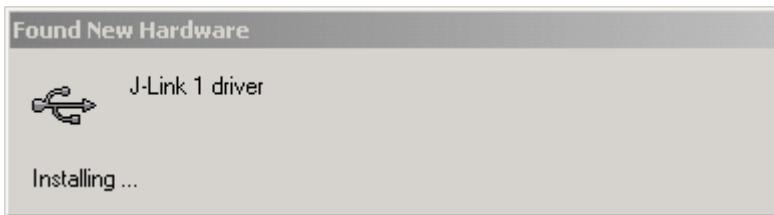
1. Start `JLink.exe` to view your hardware version. Your J-Link needs to be V5.0 or up to continue. For J-Trace the Version does not matter.
2. Type `usbaddr = 1` to set the J-Link / J-Trace #1.



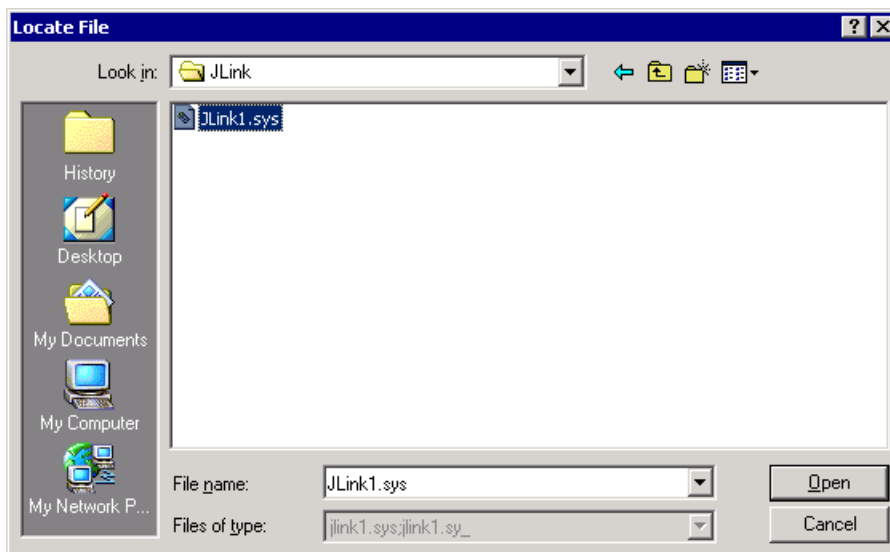
```

SEGGER J-Link Commander V2.74.01. '?' for help.
Compiled 10:17:23 on Nov 25 2005.
DLL version V2.74b, compiled Nov 25 2005 10:17:13
Firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5
Hardware: V5.00
S/N : 
VTarget = 0.0000
Speed set to 30 kHz
J-Link>usbaddr = 1
USB address successfully changed to '1'.
Please unplug the device, then plug it back in.
J-Link>
  
```

3. Unplug J-Link / J-Trace and then plug it back in.
4. The system will recognize a new J-Link / J-Trace and will prompt for a driver.

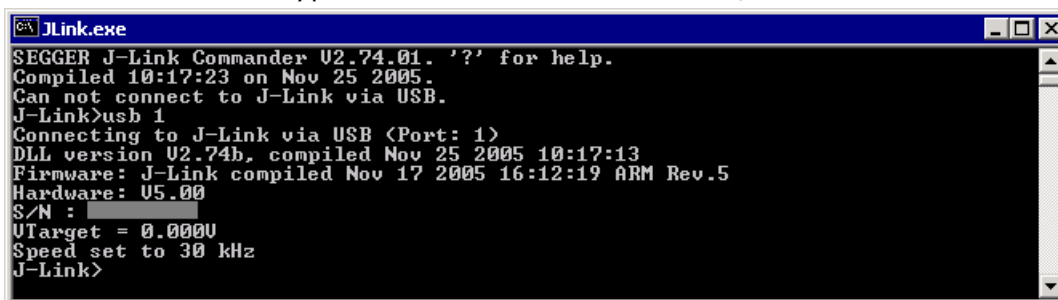


5. Click **OK** and browse to the J-Link USB driver for your new J-Link / J-Trace. For your second J-Link / J-Trace this would be `JLink1.sys`, for your third J-Link / J-Trace this would be `JLink2.sys`.



### 4.4.3 Connecting to a J-Link / J-Trace with non default USB-Address

Restart JLink.exe and type `usb 1` to connect to J-Link / J-Trace #1.



```
JLink.exe
SEGGER J-Link Commander V2.74.01. '?' for help.
Compiled 10:17:23 on Nov 25 2005.
Can not connect to J-Link via USB.
J-Link>usb 1
Connecting to J-Link via USB (Port: 1)
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 = 0.0000U
Speed set to 30 kHz
J-Link>
```

You may connect other J-Links / J-Traces to your PC and connect to them as well. To connect to an unconfigured J-Link / J-Trace (with default address "0"), restart JLink.exe or type `usb 0`.

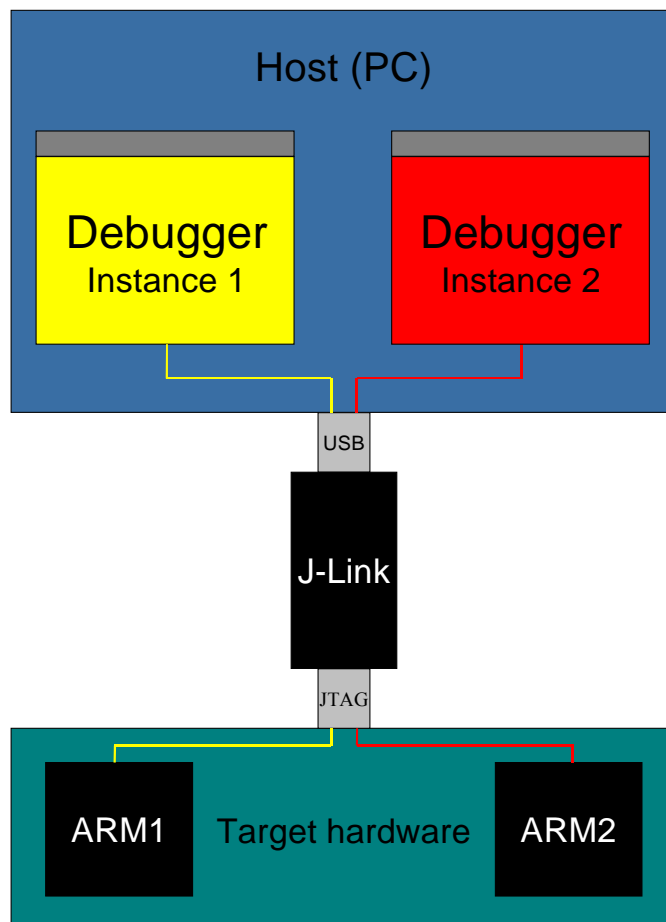
## 4.5 Multi-core debugging

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

### 4.5.1 How multi-core debugging works

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

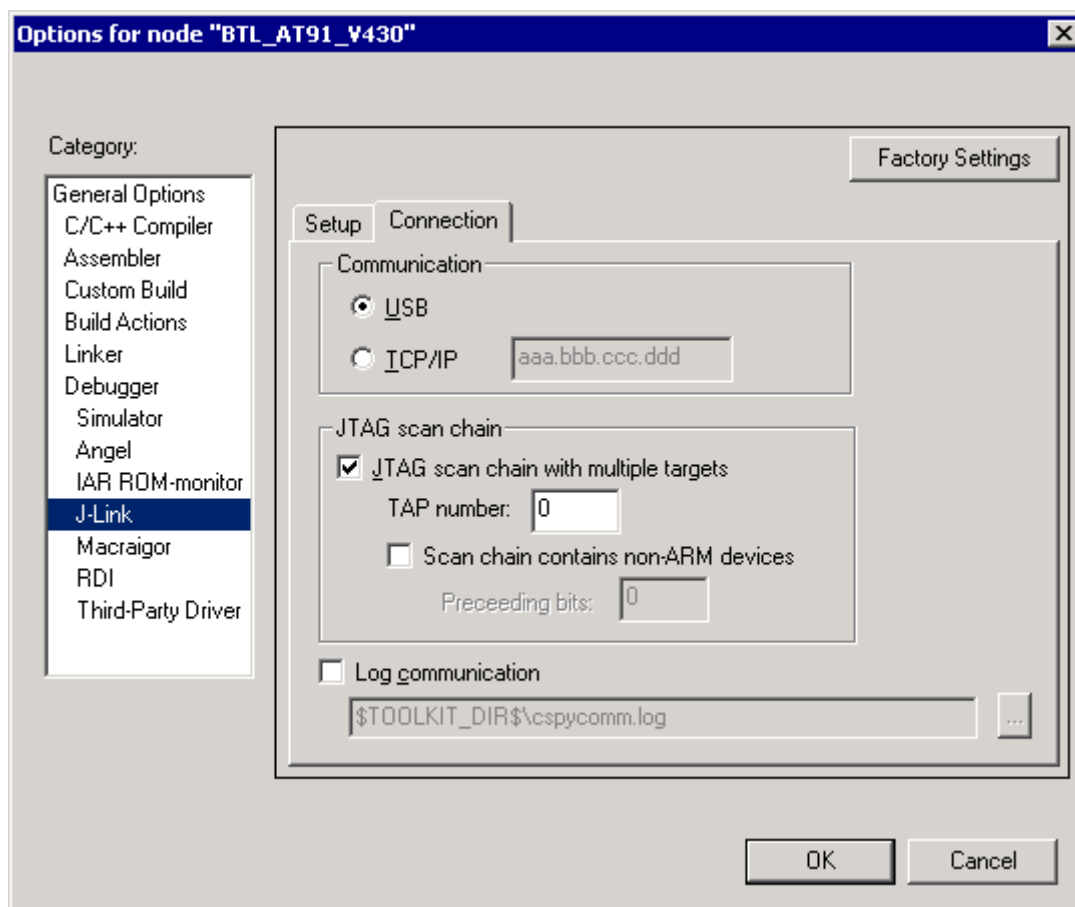
The following figure shows a host, debugging two ARM cores with two instances of the same debugger.



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

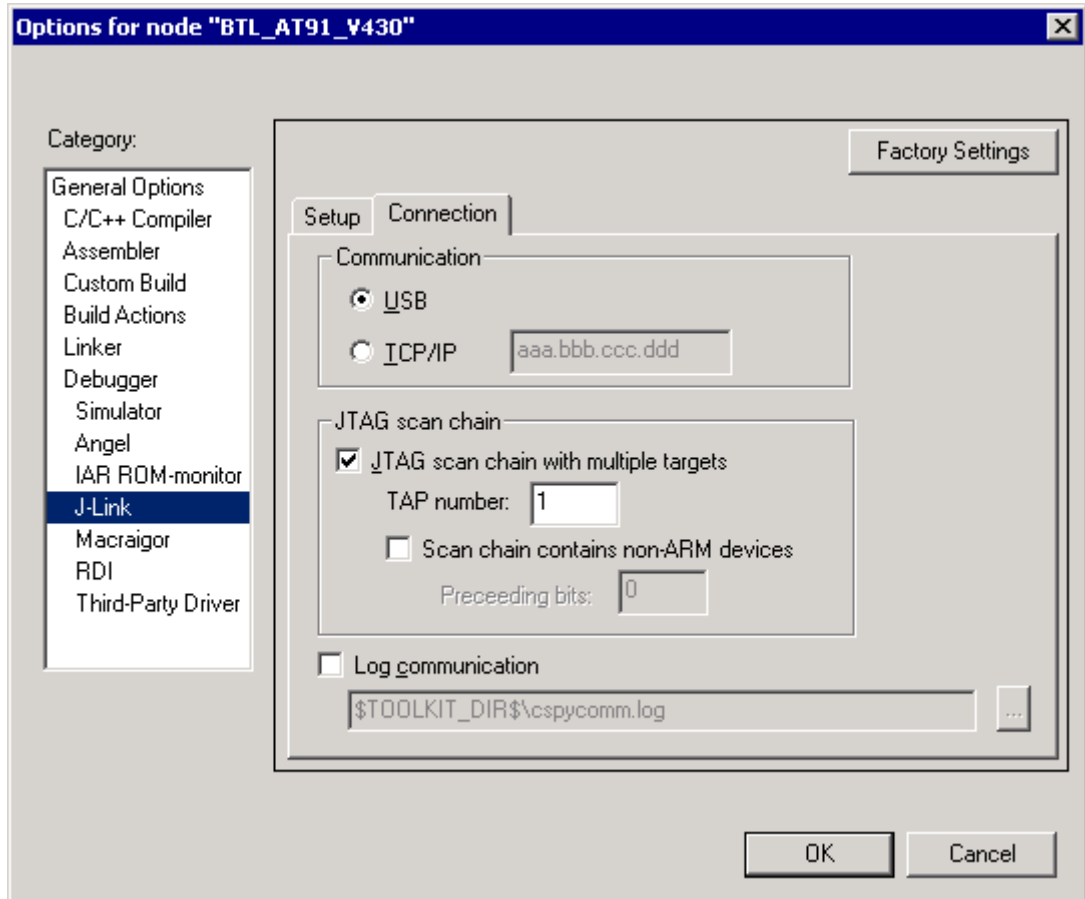
## 4.5.2 Using multi-core debugging in detail

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



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

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



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

Cores to debug are marked in blue.

## 4.5.3 Things you should be aware of

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

### 4.5.3.1 JTAG speed

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

For example:

Core #1: 2MHz maximum JTAG speed

Core #2: 4MHz maximum JTAG speed

Scan chain: 2MHz maximum JTAG speed

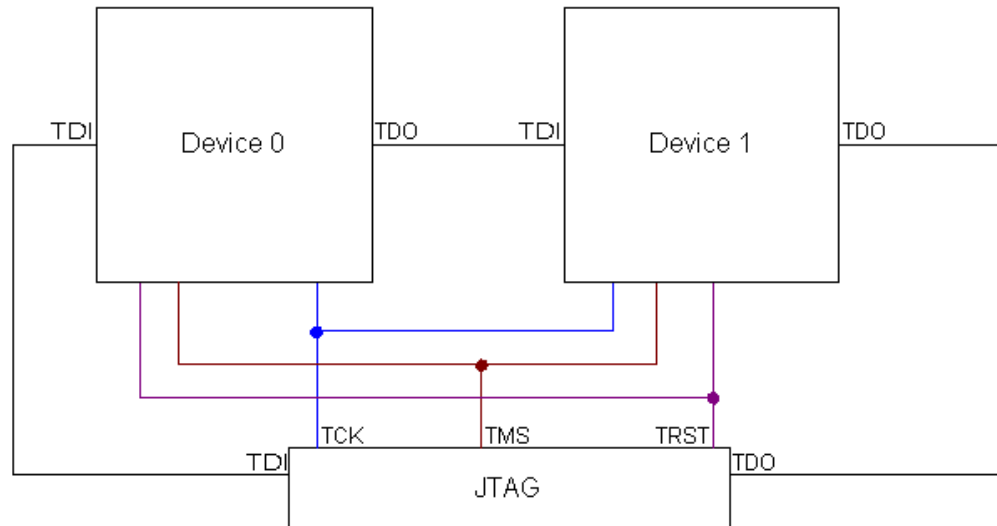
### 4.5.3.2 Resetting the target

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



## 4.6 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.

### 4.6.1 Configuration

The configuration of the scan chain depends on the application used. Read "Scan chain configuration" on page 22 for further instructions and configuration examples.

## 4.7 Using DCC for memory access

The ARM7/9 architecture requires cooperation of the CPU to access memory. 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 in debug mode) before accessing memory. Even if the CPU is restarted after the memory access, the real time behaviour 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 "Samples\DCC\IAR" directory of the package.

### 4.7.1 What is required ?

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

An application program that uses DCC is JLink.exe.

### 4.7.2 Target DCC handler

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

### 4.7.3 Target DCC abort handler

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

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

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

### 4.7.4 Testing DCC memory access

To test if your target system properly calls the DCC handler you can use JLink.exe. The output should be similar to the below:

```
SEGGER J-Link Commander V3.38c ('?' for help)
Compiled Aug 18 2006 18:48:43
DLL version V3.38c, compiled Aug 18 2006 18:48:40 -- Debug --
Firmware: J-Link compiled Aug 14 2006 14:58:04 ARM Rev.5
Hardware: V5.40
S/N : 1
VTarget = 3.306V
Speed set to 30 kHz
Found 1 JTAG device, Total IRLen = 4:
```

```

Id of device #0: 0x3F0F0F0F
Found ARM with core Id 0x3F0F0F0F (ARM7)
J-Link>mem 0,40
Reading memory using DCC...
00000000 = 0F 00 00 EA FE FF FF EA FE FF FF EA FE FF FF EA
00000010 = FE FF FF EA FE FF FF EA 1C 00 00 EA 00 90 A0 E1
00000020 = 04 01 98 E5 D3 F0 21 E3 0E 50 2D E9 0F E0 A0 E1
00000030 = 10 FF 2F E1 0E 50 BD E8 D1 F0 21 E3 09 00 A0 E1
J-Link>h
PC: (R15) = 000002EA, CPSR = 80000033 (SVC mode, THUMB)
R0 = 00005BB4, R1 = 0000BBB2, R2 = FFFFD000, R3 = 000003B8
R4 = FFFFFFF430, R5 = 00000001, R6 = 00000002, R7 = 00000000
USR: R8 =00000000, R9 =00000000, R10=00000000, R11 =00000000, R12 =00000000
    R13=00000000, R14=00000000
FIQ: R8 =FFFFFF00, R9 =00000000, R10=00000000, R11 =00000000, R12 =00000000
    R13=00000000, R14=00000000, SPSR=00000010
SVC: R13=00003F8C, R14=000002DF, SPSR=00000010
ABT: R13=00000000, R14=00000000, SPSR=00000010
IRQ: R13=00004000, R14=00000000, SPSR=00000010
UND: R13=00000000, R14=00000000, SPSR=00000010
J-Link>mem 0,40
00000000 = 0F 00 00 EA FE FF FF EA FE FF FF EA FE FF FF EA
00000010 = FE FF FF EA FE FF FF EA 1C 00 00 EA 00 90 A0 E1
00000020 = 04 01 98 E5 D3 F0 21 E3 0E 50 2D E9 0F E0 A0 E1
00000030 = 10 FF 2F E1 0E 50 BD E8 D1 F0 21 E3 09 00 A0 E1

```



# Chapter 5

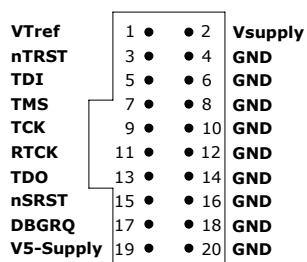
## Hardware

---

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

## 5.1 JTAG Connector

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.



### 5.1.1 Pinout

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	Vsupply	NC	This pin is not connected in J-Link. It is reserved for compatibility with other equipment. Connect to Vdd or leave open in target system.
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.
4	GND	-	Common ground.
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 target CPU.
6	GND	-	Common ground.
7	TMS	Output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of target CPU.
8	GND	-	Common ground.
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 target CPU.
10	GND	-	Common ground.
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.
12	GND	-	Common ground.
13	TDO	Input	JTAG data output from target CPU. Typically connected to TDO of target CPU.
14	GND	-	Common ground.
15	RESET	I/O	Target CPU reset signal.
16	GND	-	Common ground.
17	DBGREQ	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 DBGREQ if available, otherwise left open.
18	GND	-	Common ground.
19	5V-Supply	Output	This pin is used to supply power to some eval boards. Not all J-Links supply power on this pin, only the KS (Kickstart) versions. Typically left open on target hardware.
20	GND	-	Common ground.

## 5.1.2 Target board design for JTAG

We strongly advise following the recommendations given by the chip manufacturer. These recommendations are normally in line with the recommendations given in the table below. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

SIGNAL	CONNECTION
VTref	It is normally fed from Vdd of the target board and must not have a series resistor.
nTRST	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.
TDI	Typically connected to TDI of target CPU. It is recommended that this pin is pulled to a defined state of the target board.
TMS	Typically connected to TMS of target CPU. It is recommended that this pin is pulled up.
TCK	Typically connected to TCK of target CPU. It is recommended that this pin is pulled to a defined state on the target board.
RTCK	Typically connected to RTCK if available, otherwise to GND.
TDO	Typically connected to TDO of target CPU.
RESET	Target CPU reset signal.
DBGKQ	Typically connected to DBGKQ if available, otherwise left open.
5V-Supply	Typically left open.

### 5.1.2.1 Pull-up/pull-down resistors

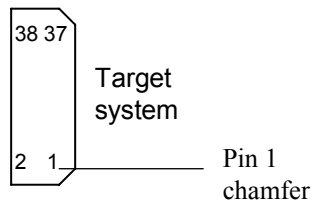
Unless otherwise specified by developer's manual, pull-ups/pull-downs are recommended to be between 2.2 kOhms and 47 kOhms.

## 5.2 JTAG+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

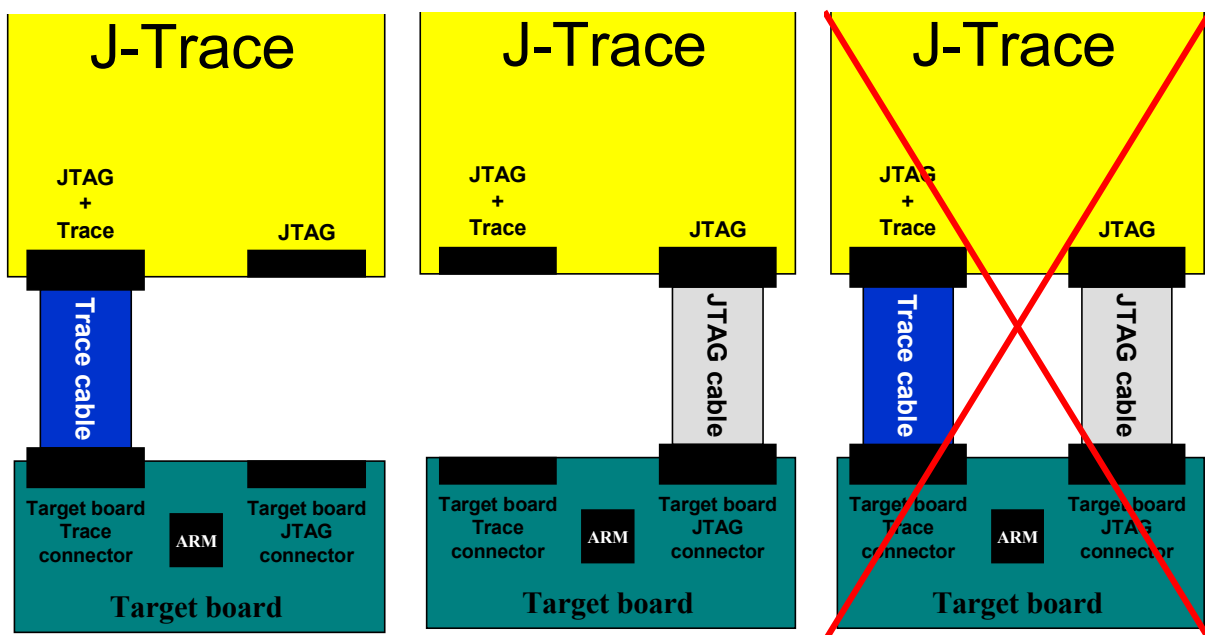


J-Trace can capture the state of signals PIPESTAT[2:0], TRACESYNC and TRACEPKT[n:0] at each rising edge of each TRACECLK or on each alternate rising or falling edge.

### 5.2.1 Connecting the target board

J-Trace connects to the target board via a 38-pin trace cable. This cable has a receptacle on the one side, and a plug on the other side. Alternatively J-Trace can be connected with a 14-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.**





## 5.2.2 Pinout

The following table lists the JTAG+Trace connector pinout. It is compatible with the "Trace Port Physical Interface" described in [ETM], 8.2.2 "Single target connector pinout".

PIN	SIGNAL	Description
1	NC	No connect.
2	NC	No connect.
3	NC	No connect.
4	NC	No connect.
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	nSRST	Open-collector output from the run control to the target system reset.
10	EXTTRIG	Optional external trigger signal to the Embedded trace Macrocell (ETM).
11	TDO	Test data output from target JTAG port.
12	VTRef	Signal level reference.
13	RTCK	Return test clock from the target JTAG port.
14	VSupply	Supply voltage.
15	TCK	Test clock to the run control unit from the JTAG port.
16	Trace signal 12	Trace signal.
17	TMS	Test mode select from run control to the JTAG port.
18	Trace signal 11	Trace signal.
19	TDI	Test data input from run control to the JTAG port.
20	Trace signal 10	Trace signal.
21	nTRST	Active-low JTAG reset
22	Trace signal 9	Trace signal.
23	Trace signal 20	Trace signal.
24	Trace signal 8	Trace signal.
25	Trace signal 19	Trace signal.
26	Trace signal 7	Trace signal.
27	Trace signal 18	Trace signal.
28	Trace signal 6	Trace signal.
29	Trace signal 17	Trace signal.
30	Trace signal 5	Trace signal.
31	Trace signal 16	Trace signal.
32	Trace signal 4	Trace signal.
33	Trace signal 15	Trace signal.
34	Trace signal 3	Trace signal.
35	Trace signal 14	Trace signal.
36	Trace signal 2	Trace signal.
37	Trace signal 13	Trace signal.
38	Trace signal 1	Trace signal.

### 5.2.3 Assignment of trace information pins between ETM architecture versions

The following table show different names for the trace signals depending on the ETM architecture version.

Trace signal	ETMv1	ETMv2	ETMv3
Trace signal 1	PIPESTAT[0]	PIPESTAT[0]	TRACEDATA[0]
Trace signal 2	PIPESTAT[1]	PIPESTAT[1]	TRACECTL
Trace signal 3	PIPESTAT[2]	PIPESTAT[2]	Logic 1
Trace signal 4	TRACESYNC	PIPESTAT[3]	Logic 0
Trace signal 5	TRACEPKT[0]	TRACEPKT[0]	Logic 0
Trace signal 6	TRACEPKT[1]	TRACEPKT[1]	TRACEDATA[1]
Trace signal 7	TRACEPKT[2]	TRACEPKT[2]	TRACEDATA[2]
Trace signal 8	TRACEPKT[3]	TRACEPKT[3]	TRACEDATA[3]
Trace signal 9	TRACEPKT[4]	TRACEPKT[4]	TRACEDATA[4]
Trace signal 10	TRACEPKT[5]	TRACEPKT[5]	TRACEDATA[5]
Trace signal 11	TRACEPKT[6]	TRACEPKT[6]	TRACEDATA[6]
Trace signal 12	TRACEPKT[7]	TRACEPKT[7]	TRACEDATA[7]
Trace signal 13	TRACEPKT[8]	TRACEPKT[8]	TRACEDATA[8]
Trace signal 14	TRACEPKT[9]	TRACEPKT[9]	TRACEDATA[9]
Trace signal 15	TRACEPKT[10]	TRACEPKT[10]	TRACEDATA[10]
Trace signal 16	TRACEPKT[11]	TRACEPKT[11]	TRACEDATA[11]
Trace signal 17	TRACEPKT[12]	TRACEPKT[12]	TRACEDATA[12]
Trace signal 18	TRACEPKT[13]	TRACEPKT[13]	TRACEDATA[13]
Trace signal 19	TRACEPKT[14]	TRACEPKT[14]	TRACEDATA[14]
Trace signal 20	TRACEPKT[15]	TRACEPKT[15]	TRACEDATA[15]

## 5.2.4 Trace signals

Data transfer is synchronized by TRACECLK.

### 5.2.4.1 Signal levels

The maximum capacitance presented by J-Trace at the trace port connector, including the connector and interfacing logic, is less than 6pF. The trace port lines have a matched impedance of 50.

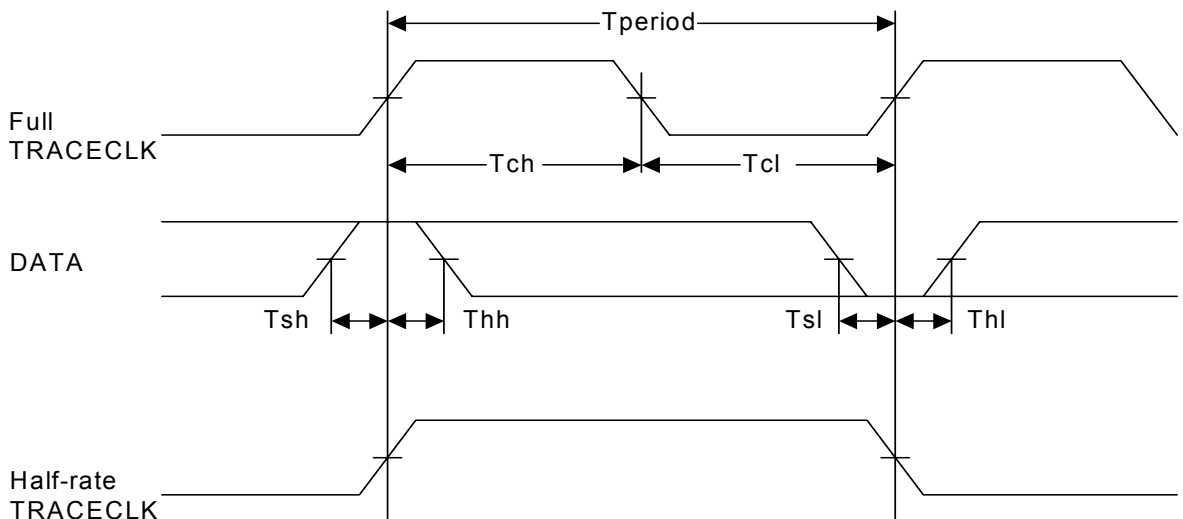
The J-Trace unit will operate with a target board that has a supply voltage range of 3.0V-3.6V.

### 5.2.4.2 Clock frequency

For capturing trace port signals synchronous to TRACECLK, J-Trace supports a TRACECLK frequency of up to 200MHz. The following table shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACECLK.

Parameter	Min.	Max.	Explanation
Tperiod	5ns	1000ns	Clock period
Fmax	1MHz	200MHz	Maximum trace frequency
Tch	2.5ns	-	High pulse width
Tcl	2.5ns	-	Low pulse width
Tsh	2.5ns	-	Data setup high
Thh	1.5ns	-	Data hold high
Tsl	2.5ns	-	Data setup low
Thl	1.5ns	-	Data hold low

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.

## 5.3 RESET, nTRST

The TAP controller and ICE logic is reset independently from the ARM core with nTRST (DBGnTRST on synthesizable cores). For the ARM core to operate correctly, it is essential that both signals are asserted after power-up.

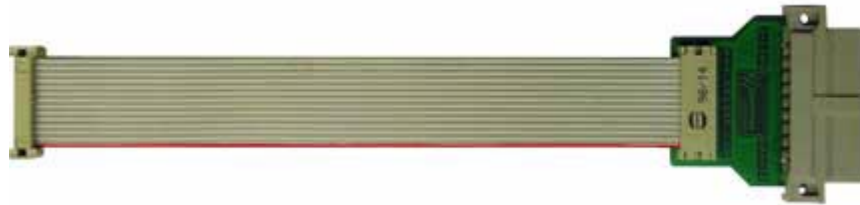
The advantage of having separate connection to the two reset signals is that it allows the developer performing software debug to set up breakpoints which are retained by the ICE logic even when the core is reset. (For example, at address 0, to allow the code to be single-stepped as soon as it comes out of reset). This can be particularly useful when first trying to bring up a board with a new ASIC.

You may tie (DBG)nTRST to the core reset, but this removes some of the flexibility and usefulness of the debug tools. What some designers facing similar pin constraints have done is to implement some kind of reset circuit within their device. This typically will assert both nTRST and the core reset for the initial power-on reset, but subsequent 'warm' resets, where the power to the device is maintained, will cause only the core reset to go LOW.

## 5.4 Adapters

### 5.4.1 JTAG 14 pin adapter

An adapter is available to use J-Link / J-Trace with targets using this 14 pin 0.1" mating JTAG connector.

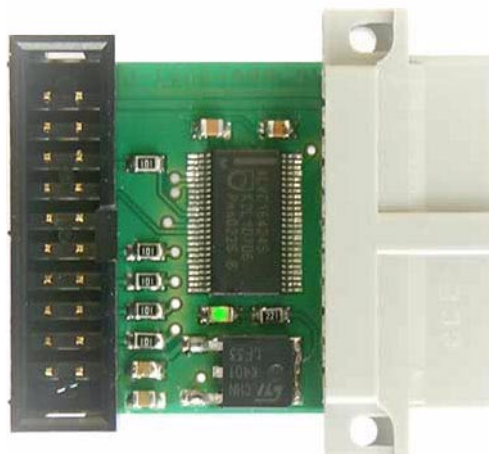


The following table shows the mapping between the 14 pin adapter and the standard 20 pin JTAG interface.

PIN	Signal	Pin no. on 20 pin JTAG
1	VTref	1
2	GND	GND
3	nTRST	3
4	GND	GND
5	TDI	5
6	GND	GND
7	TMS	7
8	GND	GND
9	TCK	9
10	GND	GND
11	TDO	13
12	RESET	15
13	VTref	1
14	GND	GND

### 5.4.2 5 Volt adapter

The 5V adapter extends the voltage range of J-Link / J-Trace (and other, pin-compatible JTAG probes) to 5V. Most targets have JTAG signals at voltage levels between 1.2V and 3.3V for J-Link and 3.0V up to 3.6V for J-Trace. These targets can be used with J-Link / J-Trace without a 5V adapter. Higher voltages are common primarily in the automotive sector.



### 5.4.2.1 Technical data

- 20 pin connector, female (plugs into J-Link / J-Trace)
- 20 pin connector male, for target ribbon cable
- LED shows power status
- Adapter is powered by target
- Power consumption < 20 mA Target supply voltage: 3.3V - 5V
- Maximum JTAG-frequency: 10 MHz

### 5.4.2.2 Compatibility note

The J-Link 5V adapter is compatible to J-Link revisions 4 or newer and J-Trace. Using an older revision of J-Link together with a 5V adapter will not output a reset signal to your target, because older J-Link versions were not able to drive high level on Reset and TRST to target. To actually determine if your J-Link is compatible to the 5V adapter, you may check whether J-Link outputs a reset signal (active high) to your target CPU.

### 5.4.2.3 Usage

The 5 volt adapter should be plugged directly into J-Link / J-Trace with the 20-pin female connector. The target ribbon cable is then attached to the 20-pin male connector of the adapter. The picture below shows a J-Link with a connected 5-volt adapter.



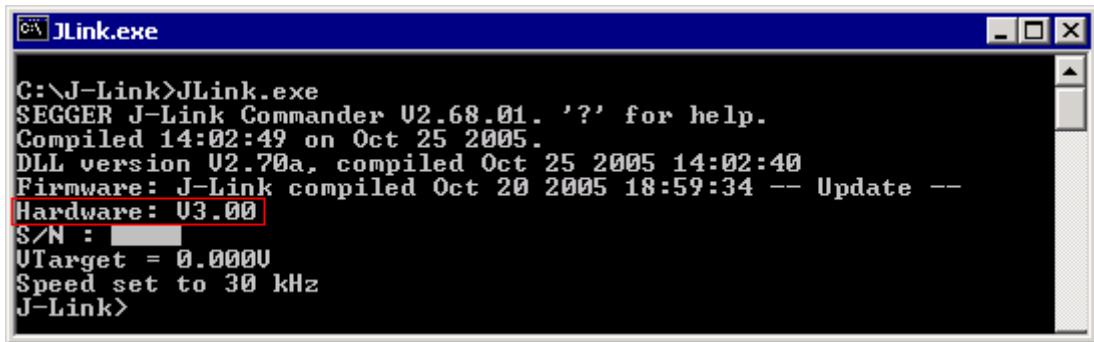
## 5.5 J-Link hardware versions

J-Link hardware has been continually improved. Several different versions in different housings are available on the market.

### 5.5.1 How to determine the hardware version

In order to determine the hardware version of your J-Link, the first step should be to look at the label at the bottom side of the unit. Newer J-Links have the hardware version printed on the back label.

If this is not the case with your J-Link, start `JLink.exe`. As part of the initial message it displays the hardware version.

A screenshot of a Windows command prompt window titled "JLink.exe". The window shows the following text:

```
C:\J-Link>JLink.exe
SEGGER J-Link Commander V2.68.01. '?' for help.
Compiled 14:02:49 on Oct 25 2005.
DLL version V2.70a, compiled Oct 25 2005 14:02:40
Firmware: J-Link compiled Oct 20 2005 18:59:34 -- Update --
Hardware: V3.00
S/N : 
VTarget = 0.0000
Speed set to 30 kHz
J-Link>
```

The text "Hardware: V3.00" is highlighted with a red rectangular box.

## 5.5.2 Differences between different versions

### 5.5.2.1 Versions 1-4

These J-Links use a 16-bit CISC CPU. Maximum download speed is approximately 150 Kbytes/second.

#### JTAG speed

Maximum JTAG frequency is 4 MHz; possible JTAG speeds are: 16 MHz / n, where n is 4,5, ..., resulting in speeds of:

4.000 MHz (n = 4)  
3.200 MHz (n = 5)  
2.666 MHz (n = 6)  
2.285 MHz (n = 7)  
2.000 MHz (n = 8)  
1.777 MHz (n = 9)  
1.600 MHz (n = 10)

Adaptive clocking is not supported.

### 5.5.2.2 Version 5.0

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)

Adaptive clocking is supported.

#### Target Interface

nTRST is push-pull type  
RESET is push-pull type

### 5.5.2.3 Version 5.2

Identical to 5.0 with the following exception:

#### Target Interface

nTRST is push-pull type  
RESET is open drain

### 5.5.2.4 Version 5.3

Identical to 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.



## 5.6 J-Link OEM versions

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

### 5.6.1 List of OEM products

The following table lists OEM versions of J-Link as well as the major differences between them and J-Link.

Product	Limitation	Features
Analog Devices mIDASLink	Works with Analog Devices chips only.	RDI, FlashDL, FlashBP
IAR J-Link		
IAR J-Link KS	Download speed limited.	
Atmel SAM-ICE	Works with ATMEL chips only.	RDI



# Chapter 6

## 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).

## 6.1 JTAG

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

### 6.1.1 Test access port (TAP)

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

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

### 6.1.2 Data registers

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

#### Bypass data register

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

#### Boundary-scan data register

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

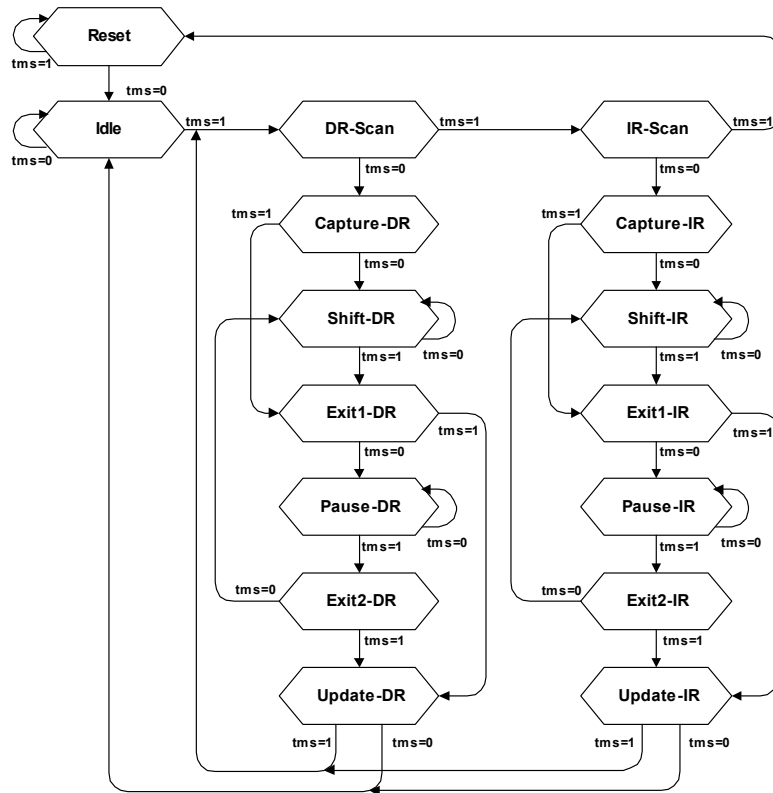
### 6.1.3 Instruction register

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

## 6.1.4 The TAP controller

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

### TAP controller state diagram



### 6.1.4.1 State descriptions

#### Reset

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

#### Idle

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

#### DR-Scan

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

#### IR-Scan

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

#### Capture-DR

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

#### Shift-DR

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

**Exit1-DR**

Temporary controller state.

**Pause-DR**

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

**Exit2-DR**

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

**Update-DR**

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

**Capture-IR**

Instructions may be loaded in parallel into the instruction register.

**Shift-IR**

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

**Exit1-IR**

Temporary controller state.

**Pause-IR**

Wait state that temporarily halts the instruction shifting.

**Exit2-IR**

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

**Update-IR**

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

## 6.2 The ARM core

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

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

### 6.2.1 Processor modes

The ARM architecture supports seven processor modes.

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

### 6.2.2 Registers of the CPU core

The CPU core has the following registers:

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

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

The ARM core has a total of 37 registers:

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

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

### 6.2.3 ARM / Thumb instruction set

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

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



## 6.3 EmbeddedICE

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

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

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

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

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

### 6.3.1 Breakpoints and watchpoints

#### Breakpoints

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

#### Watchpoints

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

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

#### Software / hardware breakpoints

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

## 6.3.2 The ICE registers

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

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

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

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

For more information about EmbeddedICE, see the technical reference manual of your ARM CPU. ([www.arm.com](http://www.arm.com))

## 6.4 Flash programming

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

### 6.4.1 How does flash programming via J-Link / J-Trace work ?

This requires extra code. This extra code typically downloads a program into the RAM of the target system, which is able to erase and program the flash. This program is called RAM code and "knows" how to program the flash; it contains an implementation of the flash programming algorithm for the particular flash. Different flash chips have different programming algorithms; the programming algorithm also depends on other things such as 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.

### 6.4.2 Data download to RAM

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

### 6.4.3 Data download via DCC

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

### 6.4.4 Available options for flash programming

There are different solutions available to program internal or external flashes connected to ARM cores using J-Link / J-Trace. The different solutions have different fields of application, but of course also some overlap.

#### 6.4.4.1 J-Flash - Complete flash programming solution

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

### 6.4.4.2 JLinkArmFlash.dll - A DLL with flash programming capabilities

An enhanced version of the JLinkARM.DLL, which has add. API functions. The additional API functions allow loading and programming a data file. This DLL comes with a sample executable, as well as the source code of this executable and a project file. This can be an interesting option if you want to write your own programs for production purposes. This DLL also requires an extra license from SEGGER; contact us for more information.

Output of Sample program:

```
SEGGER JLinkARMFlash for ST STR710FR2T6 V1.00.00  
Compiled 11:16:22 on May 4 2005.
```

```
This program and the DLL are (c) Copyright 2005 SEGGER, www.segger.com
```

```
Connecting to J-Link  
Resetting target  
Loading data file... 1060 bytes loaded.  
Erasing required sectors... O.K. - Completed after 0.703 sec  
Programming... O.K. - Completed after 0.031 sec  
Verifying... O.K. - Completed after 0.031 sec
```

### 6.4.4.3 RDI flash loader: Allows flash download from any RDI-compliant tool chain

RDI, (Remote debug interface) is a standard for "debug transfer agents" such as J-Link. It allows using J-Link from any RDI compliant debugger. RDI by itself does not include download to flash. To debug in flash, you need to somehow program your application program (debuggee) into the flash. You can use J-Flash for this purpose, use the flash loader supplied by the debugger company (if they supply a matching flash loader) or use the flash loader integrated in the J-Link RDI software. The RDI software as well as the RDI flash loader require licenses from SEGGER.

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

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

### 6.4.4.5 Write your own flash loader

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

## 6.5 J-Link / J-Trace firmware

The heart of J-Link / J-Trace is a microcontroller. The firmware is the software executed by the microcontroller inside of the J-Link / J-Trace. The J-Link / J-Trace firmware sometimes needs to be updated. This firmware update is performed automatically as necessary by the JLinkARM.dll.

### 6.5.1 Firmware update

Every time you connect to J-Link / J-Trace, JLinkARM.dll checks if its embedded firmware is newer than the one used in the J-Link / J-Trace. It 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: V5.00
S/N :
Utarget = 0.0000
Speed set to 30 kHz
J-Link>

```

The red box identifies the new firmware.

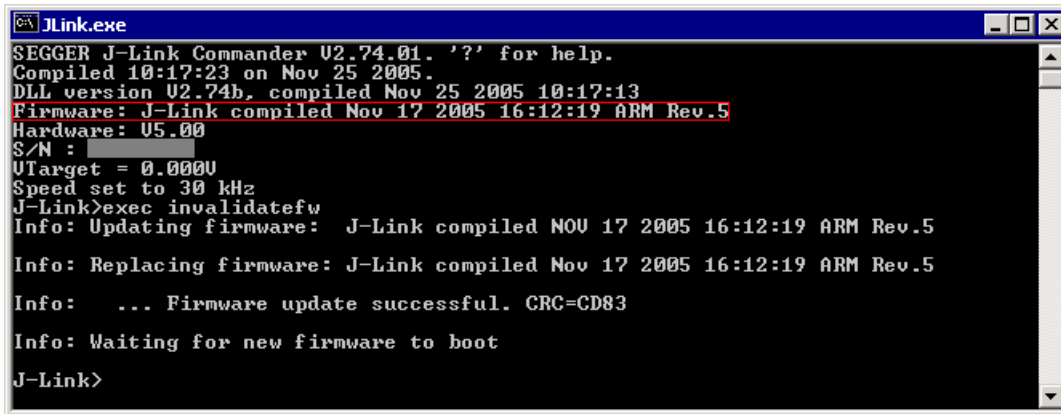
The green box identifies the old firmware which has been replaced.

## 6.5.2 Invalidating the firmware

Downdating J-Link / J-Trace is not done automatically thru an old JLinkARM.dll. J-Link / J-Trace will continue using its current, newer firmware when using older versions of the JLinkARM.dll.

Downdating J-Link / J-Trace is not recommended, you do it at your own risk! Note that the firmware embedded in older versions of JLinkARM.dll may not run 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`.

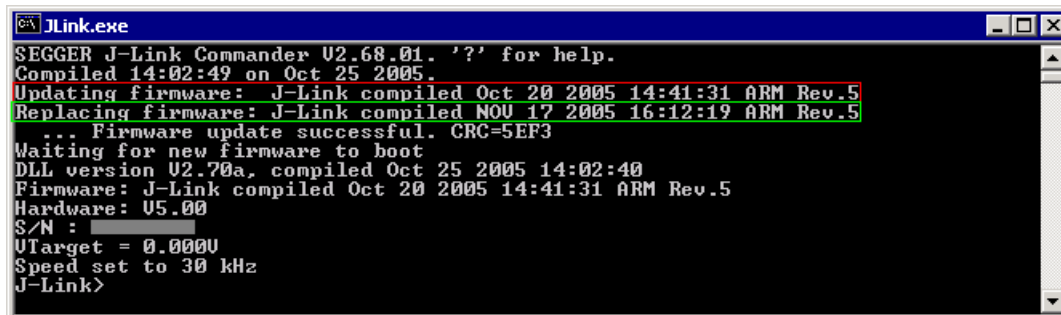


```

C:\> 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>
  
```

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.



```

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>
  
```

The red box identifies the new firmware.

The green box identifies the old firmware which has been replaced.

# Chapter 7

## Designing the target board for trace

---

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

## 7.1 Overview of high-speed board design

Failure to observe high-speed design rules when designing a target system containing an ARM Embedded Trace Macrocell (ETM) trace port can result in incorrect data being captured by J-Trace. You must give serious consideration to high-speed signals when designing the target system.

The signals coming from an ARM ETM trace port can have very fast rise and fall times, even at relatively low frequencies.

Note: These principles apply to all of the trace port signals (TRACEPKT[0:15], PIPESTAT[0:2], TRACESYNC), but special care must be taken with TRACECLK.

### 7.1.1 Avoiding stubs

Stubs are short pieces of track that tee off from the main track carrying the signal to, for example, a test point or a connection to an intermediate device. Stubs cause impedance discontinuities that affect signal quality and must be avoided.

Special care must therefore be taken when ETM signals are multiplexed with other pin functions and where the PCB is designed to support both functions with differing tracking requirements.

### 7.1.2 Minimizing Signal Skew (Balancing PCB Track Lengths)

You must attempt to match the lengths of the PCB tracks carrying all of TRACECLK, PIPESTAT, TRACESYNC, and TRACEPKT from the ASIC to the mictor connector to within approximately 0.5 inches (12.5mm) of each other. Any greater differences directly impact the setup and hold time requirements.

### 7.1.3 Minimizing Crosstalk

Normal high-speed design rules must be observed. For example, do not run dynamic signals parallel to each other for any significant distance, keep them spaced well apart, and use a ground plane and so forth. Particular attention must be paid to the TRACECLK signal. If in any doubt, place grounds or static signals between the TRACECLK and any other dynamic signals.

### 7.1.4 Using impedance matching and termination

Termination is almost certainly necessary, but there are some circumstances where it is not required. The decision is related to track length between the ASIC and the JTAG+Trace connector, see ["Terminating the trace signal" on page 81](#) for further reference.



## 7.2 Terminating the trace signal

To terminate the trace signal, you can choose between three termination options:

- Matched impedance
- Series (source) termination
- DC parallel termination.

### Matched impedance

Where available, the best termination scheme is to have the ASIC manufacturer match the output impedance of the driver to the impedance of the PCB track on your board. This produces the best possible signal.

### Series (source) termination

This method requires a resistor fitted in series with signal. The resistor value plus the output impedance of the driver must be equal to the PCB track impedance.

### DC parallel termination

This requires either a single resistor to ground, or a pull-up/pull-down combination of resistors (Thevenin termination), fitted at the end of each signal and as close as possible to the JTAG+Trace connector. If a single resistor is used, its value must be set equal to the PCB track impedance. If the pull-up/pull-down combination is used, their resistance values must be selected so that their parallel combination equals the PCB track impedance.

#### Caution:

At lower frequencies, parallel termination requires considerably more drive capability from the ASIC than series termination and so, in practice, DC parallel termination is rarely used.

### 7.2.1 Rules for series terminators

Series (source) termination is the most commonly used method. The basic rules are:

1. The series resistor must be placed as close as possible to the ASIC pin (less than 0.5 inches).
2. The value of the resistor must equal the impedance of the track minus the output impedance of the output driver. So for example, a 50 PCB track driven by an output with a 17 impedance, requires a resistor value of 33.
3. A source terminated signal is only valid at the end of the signal path. At any point between the source and the end of the track, the signal appears distorted because of reflections. Any device connected between the source and the end of the signal path therefore sees the distorted signal and might not operate correctly. Care must be taken not to connect devices in this way, unless the distortion does not affect device operation.

## 7.3 Signal requirements

The table below lists the specifications that apply to the signals as seen at the JTAG+Trace connector.

Signal	Value
Fmax	200MHz
Ts setup time (min.)	2.0ns
Th hold time (min.)	1.0ns
TRACECLK high pulse width (min.)	1.5ns
TRACECLK high pulse width (min.)	1.5ns



# Chapter 8

## 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.

## 8.1 Troubleshooting

### 8.1.1 General procedure

If you experience problems with a 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. Power-off target.
4. Re-connect J-Link / J-Trace with host system (attach USB cable).
5. Power-on target.
6. Try your target application again. If the problem vanished, you are done; otherwise continue.
7. Close all running applications on your host system again.
8. Disconnect the J-Link / J-Trace device from USB.
9. Power-off target.
10. Re-connect J-Link / J-Trace with host system (attach USB cable).
11. Power-on target.
12. Start `JLink.exe`.
13. If `JLink.exe` reports 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 8.2.
15. If your problem persists and you own an original product (not an OEM version), see section "Contacting support" on page 86.

### 8.1.2 Typical problem scenarios

#### **J-Link / J-Trace LED is off**

*Meaning:*

The USB connection does not work.

*Remedy:*

Check the USB connection. Try to re-initialize J-Link / J-Trace by disconnecting and reconnecting it. Make sure that the connectors are firmly attached. Check the cable connections on your J-Link / J-Trace and the computer. If this does not solve the problem, check if your cable is defective. 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 15.

#### **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:*

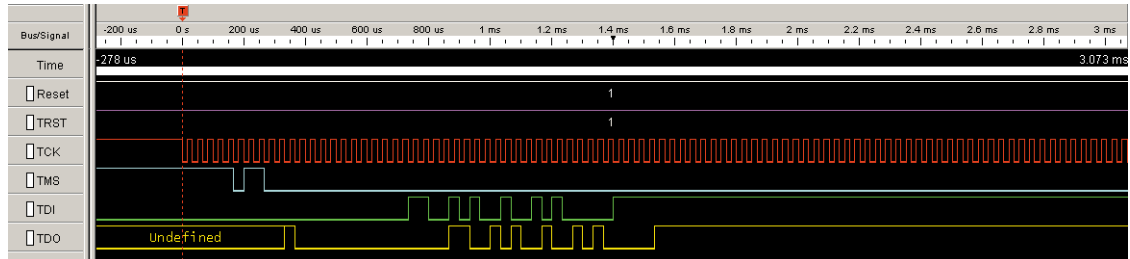
Follow the steps described in section 8.1.1.

## 8.2 Signal analysis

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

### 8.2.1 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:

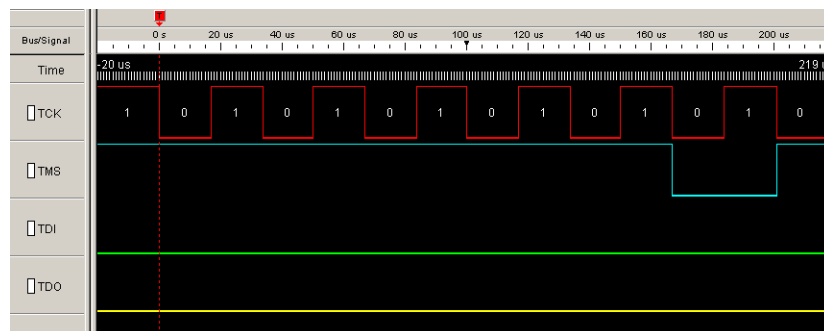
- 5 clocks: TDI low, TMS high. Brings TAP controller into RESET state
- 1 clock: TDI low, TMS low: Brings TAP controller into IDLE state
- 2 clocks: TDI low, TMS high: Brings TAP controller into IR-SCAN state
- 2 clocks: TDI low, TMS low: Brings TAP controller into SHIFT-IR state
- 32 clocks: TMS low, 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.



## 8.2.2 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.

## 8.3 Contacting support

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section "General procedure" on page 84. You may also try your J-Link / J-Trace with another PC and if possible with another target system to see if it works there. If the device functions correctly, the USB setup on the original machine or your target hardware is the source of the problem, not J-Link / J-Trace.

If you need to contact support, send the following information to [support@segger.com](mailto:support@segger.com):

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

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

## 8.4 Frequently Asked Questions

- Q: Which CPUs are supported?  
 A: J-Link / J-Trace should work with any ARM7 / ARM9 core. For a list of supported cores, see section "Supported ARM Cores" on page 40.
- Q: What is the maximum JTAG speed supported by J-Link / J-Trace?  
 A: J-Link's/J-Trace's maximum supported JTAG speed is 12MHz.
- Q: What is the maximum download speed?  
 A: The maximum download speed is currently about 600 Kbytes/second when downloading into RAM; Communication with a RAM-image via DCC can be a lot faster. However, the actual speed depends on various factors, such as JTAG, clock speed, host CPU core etc.
- Q: Can I access individual ICE registers via J-Link / J-Trace?  
 A: Yes, you can access all individual ICE registers via J-Link / J-Trace.
- Q: I want to write my own application and use J-Link / J-Trace. Is this possible?  
 A: Yes. We offer a dedicated Software Developer Kit (SDK). See section "J-Link Software Developer Kit (SDK)" on page 36 for further information.
- Q: Can I use J-Link / J-Trace to communicate with a running target via DCC?  
 A: Yes. The DLL includes functions to communicate via DCC. However, you can also program DCC communication yourself by accessing the relevant ICE registers through J-Link / J-Trace.
- 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.
- Q: J-Link / J-Trace is quite inexpensive. What is the advantage of some more expensive JTAG probes?  
 A: Some of the more expensive JTAG probes offered by other manufacturers support higher download speeds or an ethernet interface. The functionality is similar, there is no real advantage of using more expensive probes. J-Link / J-Trace is a suitable solution for the majority of development tasks as well as for production purposes. Some features that are available for J-Link / J-Trace, such as a DLL, exposing the full functionality of the emulator, flash download and flash breakpoints are not available for most of these emulators.
- Q: Does J-Link support the embedded trace macro (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.
- Q: Why does J-Link / J-Trace - in contrast to most other JTAG emulators for ARM cores - not require the user to specify a cache clean area?  
 A: 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.





# Chapter 9

## Glossary

---

This chapter explains 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. Only write-back caches can have dirty data, because a write-through cache writes data to the cache and to main memory simultaneously. The process of writing dirty data to main memory is called cache cleaning.

**Dynamic Linked Library (DLL)**

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

**EmbeddedICE**

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

**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. Memory coherency is made difficult when there are multiple possible physical locations that are involved, such as a system that has main memory, a write buffer and a cache.

**Memory management unit (MMU)**

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

**Memory Protection Unit (MPU)**

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

**Multi-ICE**

Multi-processor EmbeddedICE interface. ARM registered trademark.

**nSRST**

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

**nTRST**

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

**Open collector**

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

**Processor Core**

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

**Program Status Register (PSR)**

Contains some information about the current program and some information about the current processor. Often, therefore, also referred to as Processor Status Register. Is also referred to as Current PSR (CPSR), to emphasize the distinction between it and the Saved PSR (SPSR). The SPSR holds the value the PSR had when the current function was called, and which will be restored when control is returned.

**Remapping**

Changing the address of physical memory or devices after the application has started

executing. This is typically done to allow RAM to replace ROM once the initialization has been done.

### **Remote Debug Interface (RDI)**

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

### **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.



# Chapter 10

## 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> ).



# Index

---

## Numerics

5 volt adapter .....61

## A

Adaptive clocking .....90

Application Program Interface .....90

### ARM

Processor modes .....71

Registers .....71

Thumb instruction set .....72

## B

Big-endian .....90

## C

Cache cleaning .....90

Coprocessor .....90

## D

Dirty data .....90

Dynamic Linked Library (DLL) .....90

## E

EmbeddedICE ..... 73, 90

## F

Flash breakpoints .....35

## H

Halfword .....90

Host .....90

## I

ICache .....90

ICE Extension Unit .....90

ID .....90

IEEE 1149.1 .....90

Image .....90

In-Circuit Emulator .....91

Instruction Register ..... 91  
 IR ..... 91

**J**

J-Flash ..... 34  
 J-Link  
   Adapters ..... 61  
   Developer Pack DLL ..... 36  
   Features ..... 10–11  
   Specifications ..... 13  
   Supported chips ..... 40, 60  
 J-Link ARM Flash DLL ..... 36  
 Joint Test Action Group (JTAG) ..... 91  
 JTAG ..... 68  
   PINs ..... 54  
   TAP controller ..... 69  
 JTAG 14 pin adapter ..... 61

**L**

Little-endian ..... 91

**M**

Memory coherency ..... 91  
 Memory management unit (MMU) ..... 91  
 Memory Protection Unit (MPU) ..... 91  
 Multi-ICE ..... 91

**N**

nSRST ..... 91  
 nTRST ..... 54–55, 91

**O**

Open collector ..... 91

**P**

Processor Core ..... 91  
 Program Status Register (PSR) ..... 91

**R**

RDI Support ..... 35  
 Remapping ..... 91  
 Remote Debug Interface (RDI) ..... 92  
 RTCK ..... 92  
 RTOS ..... 92

**S**

Scan Chain ..... 92  
 Semihosting ..... 92  
 Support ..... 83, 89  
 SWI ..... 92

**T**

TAP Controller ..... 92  
 Target ..... 92  
 TCK ..... 54–55, 92  
 TCP/IP Server ..... 32  
 TDI ..... 54–55, 92  
 TDO ..... 54–55, 92  
 Test Access Port (TAP) ..... 92  
 Transistor-transistor logic (TTL) ..... 92

**W**

Watchpoint ..... 73, 92  
 Word ..... 93