J-Link ARM

JTAG Emulator for
ARM Cores

SEGGER

A product of SEGGER Microcontroller Systeme GmbH

Disclaimer

Copyright notice

Trademarks

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

Manual versions

This manual describes the J-Link ARM device.
For further information on topics or routines not yet specified, please contact us.

Manual version	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

Chapter 1

Introduction

This chapter describes the purpose of this manual and then gives a short overview about J-Link ARM.

1.1 About this document

This document describes J-Link ARM. It provides an overview over the major features of J-Link, gives you some background information about JTAG and ARM in general and describes J-Link ARM related software packages available from Segger. Finally, the chapter “Support” on page 59 helps to troubleshoot common problems.
For simplicity, we will refer to J-Link ARM as J-Link in this manual.

1.2 Overview

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

1.2.1 Features of J-Link ARM

- Any ARM7/ARM9 core supported, including thumb mode
- Download speed up to 650 kb/s*
- Seamless integration into the IAR workbench
- No power supply required, powered through USB
- Max. JTAG speed 12 MHz
- 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
- Live memory viewer (J-Mem) included
- J-Link TCP/IP server included, which allows using J-Link via TCP/IP networks
- RDI server 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
- J-Link ARM Developer Pack available

* = Measured with J-Link ARM Rev.5 with DCC mode.

1.2.2 J-Link ARM download speed

The following table lists J-Link ARM performance values (kByte/s) for writing to memory (RAM):

Revision	Via *DCC* 10Mhz JTAG clock	ARM7 *Memory* download 8MHz JTAG speed	ARM9 *Memory* download 4Mhz JTAG speed
Rev 1 — Rev 4	186.2	142.9	72
Rev 5	655.3	162.7	65.2

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

1.2.3 Specifications

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 ARM and Target	up to 12 MHz
Supported Target Voltage	1.2 - 3.3 V (5V adapter available)
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

1.3 Requirements

Host System

In order to use J-Link ARM you need a host system running Windows 2000 or Windows XP with Segger's custom USB driver.

Target System

An ARM7 or ARM9 target system is required. The system should have a 20-pin connector as defined by ARM Ltd. The individual pins are described in section "JTAG Connector" on page 10. Please note that Segger offers an optional adapter to use J-Link ARM with targets using 14 pin 0.1" mating JTAG connectors.

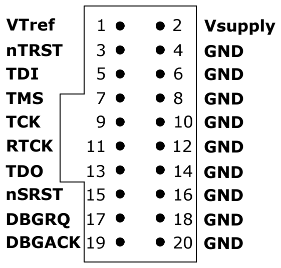
Chapter 2

Hardware

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

2.1 JTAG Connector

J-Link ARM has 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.



2.1.1 Pinout

The following table lists the J-Link ARM 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 controls the output logic levels to the target. It is normally fed from Vdd on 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 on the target JTAG port. Typically connected to nTRST on 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 on 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 on 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 on the target board. Typically connected to TCK on 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 ARM 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 on target CPU.
14	GND	-	Common ground
15	RESET	I/O	Target CPU reset signal
16	GND	-	Common ground
17	DBGRQ	NC	This pin is not connected in J-Link. It is reserved for compatibility with other equipment to be used as a debug request signal to the target system. Typically connected to DBGRQ if available, otherwise left open.

PIN	SIGNAL	TYPE	Description
18	GND	-	Common ground
19	DBGACK	NC	This pin is not connected in J-Link. It is reserved for compatibility with other equipment to be used as debug acknowledge signal from the target system. Typically left open.
20	GND	-	Common ground

2.2 Supported ARM Cores

J-Link ARM 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 if you need support.

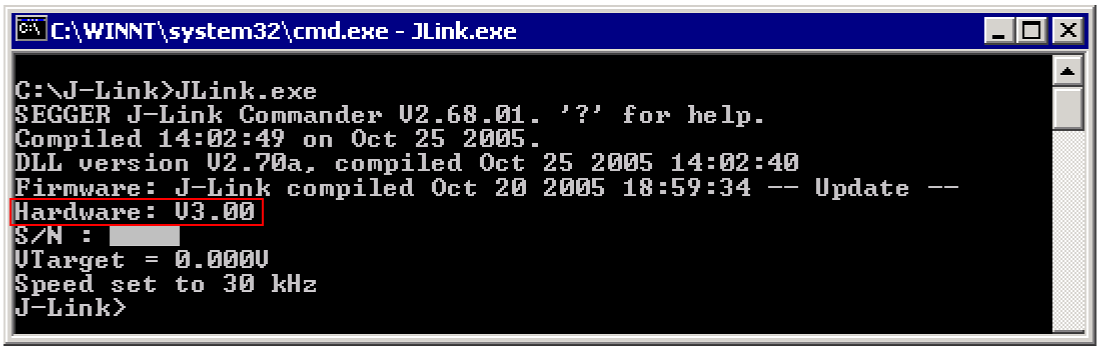
- ARM7TDMI (Rev 1)
- ARM7TDMI (Rev 3)
- ARM7TDMI-S (Rev 4)
- ARM720T
- ARM920T
- ARM922T
- ARM926EJ-S
- ARM946E-S

2.3 Hardware versions

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

2.3.1 How to determine the hardware version

In order to 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, please start J-Link.exe. As part of the initial message it displays the hardware version.



```
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 : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>
```

2.3.2 Differences between different versions

2.3.2.1 Versions 1-4

These J-Links use a 16 bit CISC CPU. Maximum download speed is app. 150 kB / sec.

JTAG speed

Maximum JTAG frequency is 8 MHz; possible JTAG speeds are:
8 MHz / n, where n is 1,2, ..., resutling in speeds of:

8.000 MHz (n = 1)
4.000 MHz (n = 2)
2.666 MHz (n = 3)
2.000 MHz (n = 4)
1.600 MHz (n = 5)
1.333 MHz (n = 6)
1.142 MHz (n = 7)
1.000 MHz (n = 8)

Adaptive clocking is not supported.

2.3.2.2 Version 5.0

Uses a 32 bit RISC CPU. Maximum download speed (using DCC) is over 650 kB / sec.

JTAG speed.

Maximum JTAG frequency is 12 MHz; possible JTAG speeds are:
48 MHz / n, where n is 4,5, ..., resutling 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)

Target Interface

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

2.3.2.3 Version 5.2

Uses a 32 bit RISC CPU. Maximum download speed (using DCC) is over 650 kB / sec.

JTAG speed.

Possible JTAG speeds are identical to these of version 5.0.

Target Interface

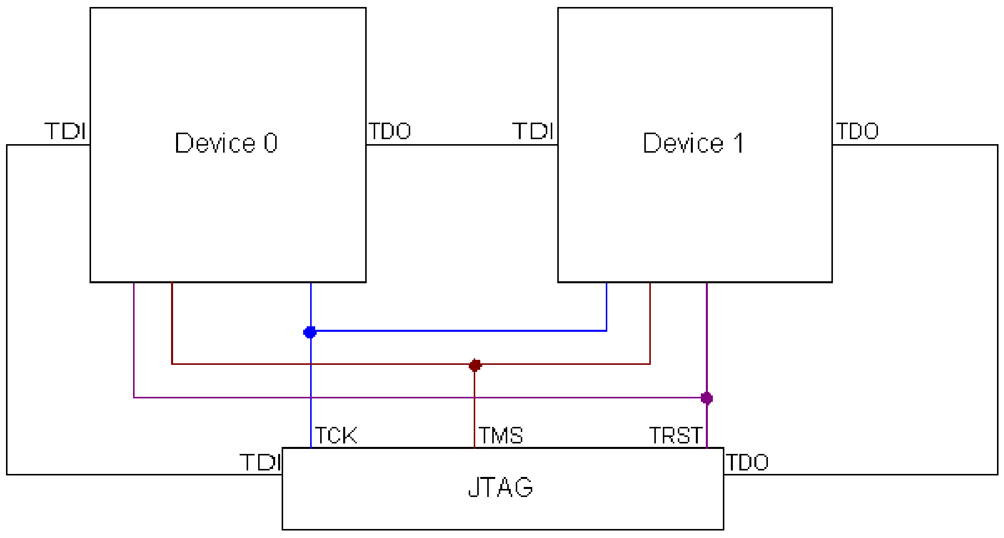
nTRST is push-pull type
RESET is open drain

2.4 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 person doing 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.

2.5 Multiple devices in the scan chain

J-Link ARM 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 drawing below, 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.

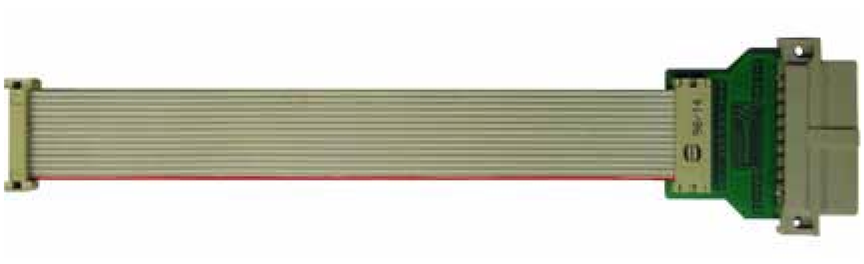
2.5.1 Configuration

The configuration of the scan chain depends on the application used. Please read “Scan chain configuration” on page 24 for further instructions and configuration examples.

2.6 Adapters

2.6.1 JTAG 14 pin adapter

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

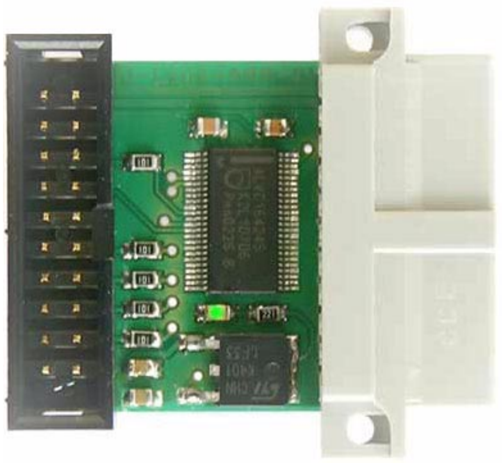


The table below shows the mapping between the 14 pin adapter and the standard 20 pin JTAG interface.

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

2.6.2 5 Volt adapter

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



2.6.2.1 Technical data

- 20 pin connector, female (plugs into J-Link ARM)
- 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
- Max. JTAG-frequency: 10 MHz

2.6.2.2 Compatibility note

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

2.6.2.3 Usage

The 5 volt adapter should be plugged directly into J-Link ARM 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 ARM with a connected 5 volt adapter.

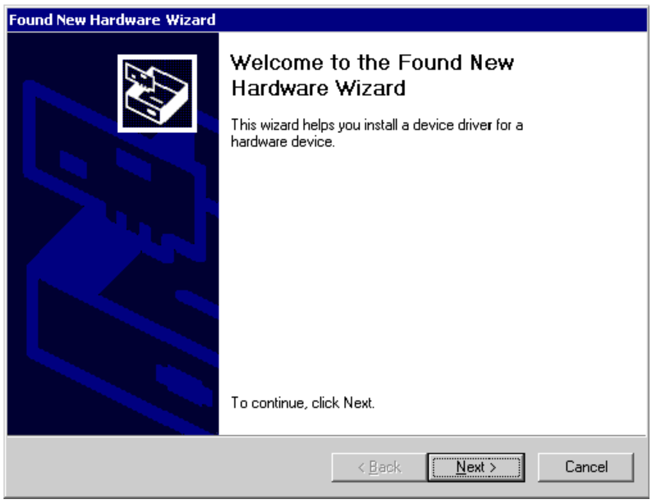
Chapter 3

Setup

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

3.1 Installing the USB driver

When your J-Link ARM is plugged into your computer's USB port, or when the computer is first powered on after connecting J-Link ARM, Windows will detect the new hardware.



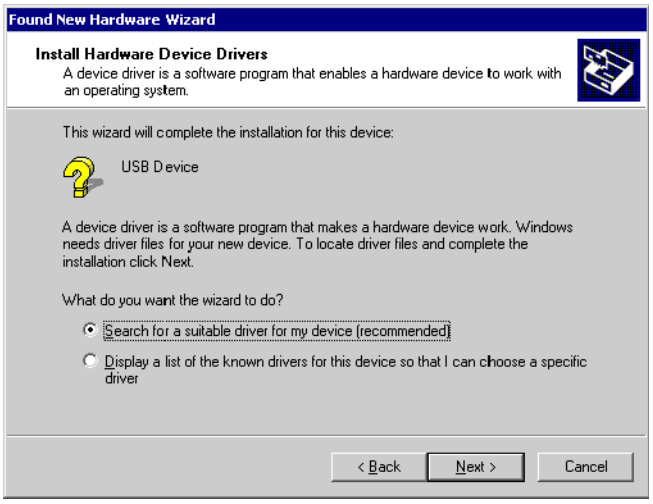
Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

The wizard starts the installation of the driver. First select the "`Search for a suitable driver for my device (recommended)`" option, then click on the "`Next >`" button.



Found New Hardware Wizard

Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

USB Device

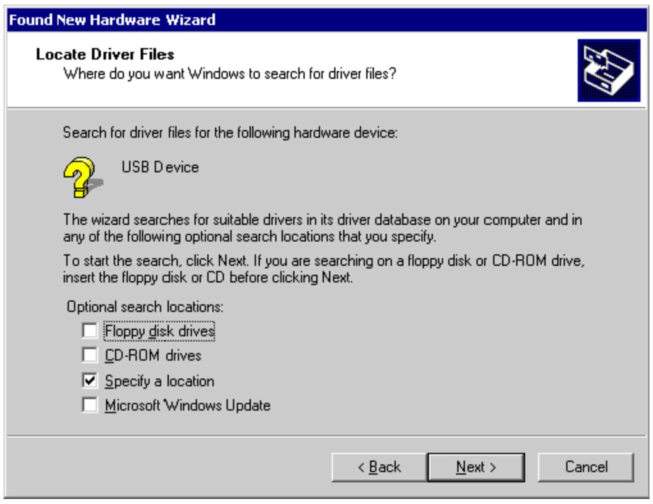
A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

(●) Search for a suitable driver for my device (recommended)

( ) Display a list of the known drivers for this device so that I can choose a specific driver

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



Found New Hardware Wizard

Locate Driver Files
Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

USB Device

The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

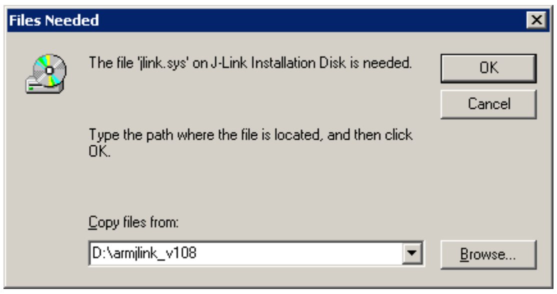
To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

- [ ] Floppy disk drives
- [ ] CD-ROM drives
- [x] Specify a location
- [ ] Microsoft Windows Update

< Back | Next > | Cancel

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 "`Next >`" button.



Files Needed

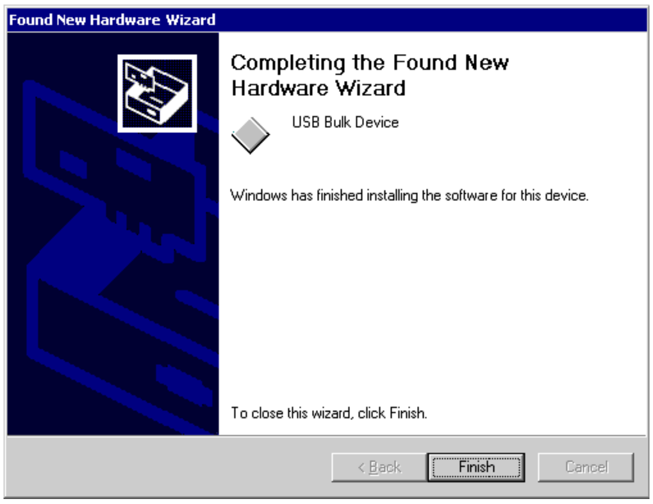
The file 'jlink.sys' on J-Link Installation Disk is needed.

Type the path where the file is located, and then click OK.

Copy files from: D:\armjlink_v108

OK | Cancel | Browse...

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



Found New Hardware Wizard

Completing the Found New Hardware Wizard

USB Bulk Device

Windows has finished installing the software for this device.

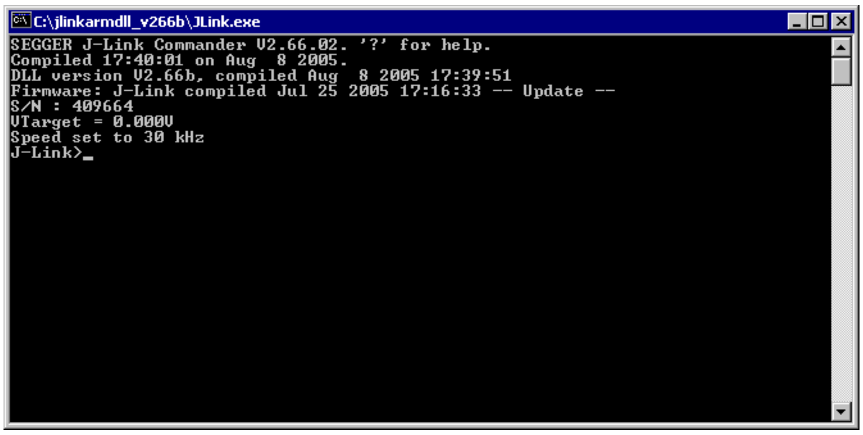
To close this wizard, click Finish.

< Back | Finish | Cancel

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

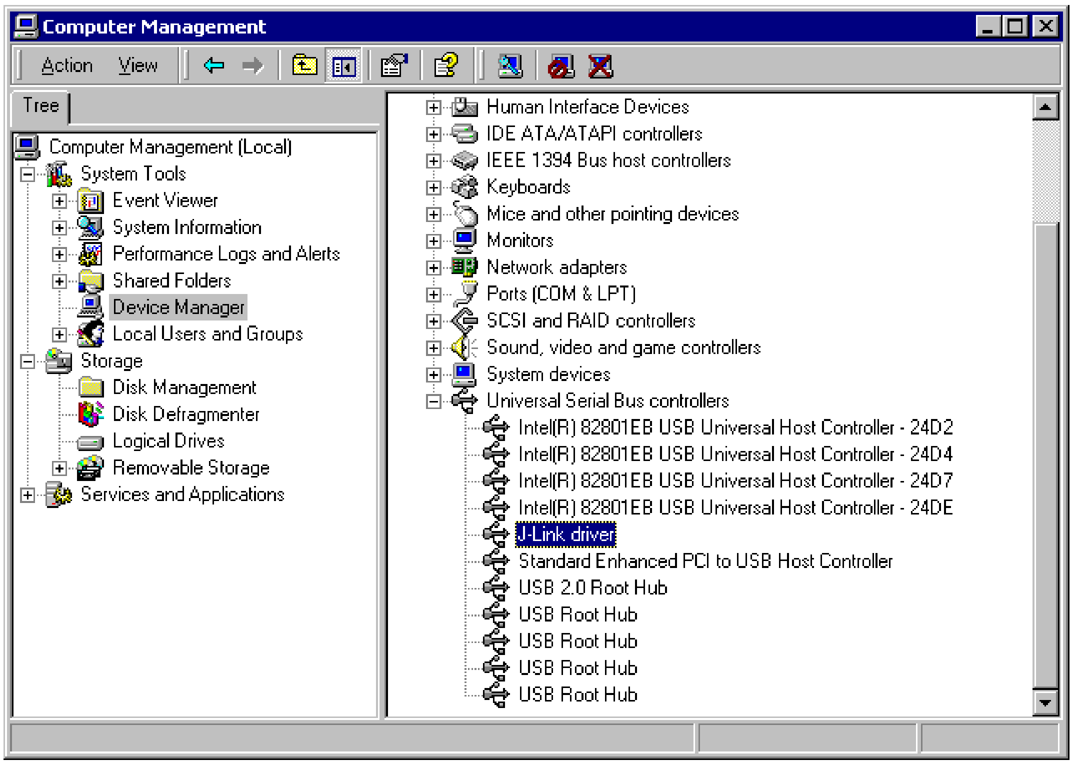
3.1.1 Verifying correct driver installation

To verify the correct installation of the driver, disconnect and reconnect J-Link ARM to the USB port. During the enumeration process which takes about 2 seconds, the LED on J-Link ARM is flashing. After successful enumeration, the LED stays on permanently.
Start the provided sample application `JLink.exe`. `JLink.exe` should display the compilation time of the J-Link ARM firmware, the serial number, a target voltage of 0.000V if a target is not connected to J-Link ARM and the speed selection. The screenshot below should give you an idea.



```
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 : 409664
VTarget = 0.000V
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 ARM is connected to your computer, the device manager should list the J-Link ARM driver as a node below "Universal Serial Bus controllers" as show in the following screenshot:



```
Computer Management (Local)
  System Tools
    Event Viewer
    System Information
    Performance Logs and Alerts
    Shared Folders
    Device Manager
    Local Users and Groups
  Storage
    Disk Management
    Disk Defragmenter
    Logical Drives
    Removable Storage
  Services and Applications
```

```
Human Interface Devices
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
SCSI and RAID controllers
Sound, video and game controllers
System devices
Universal Serial Bus controllers
  Intel(R) 82801EB USB Universal Host Controller - 24D2
  Intel(R) 82801EB USB Universal Host Controller - 24D4
  Intel(R) 82801EB USB Universal Host Controller - 24D7
  Intel(R) 82801EB USB Universal Host Controller - 24DE
  J-Link driver
  Standard Enhanced PCI to USB Host Controller
  USB 2.0 Root Hub
  USB Root Hub
  USB Root Hub
  USB Root Hub
  USB Root Hub
```

A right-click on the driver will open a context menu which contains the item "Properties". If you select this item, a new dialog is opened and should report: "This device is working properly".

If you experience problems, please have a look at chapter “Support” on page 59 for help.

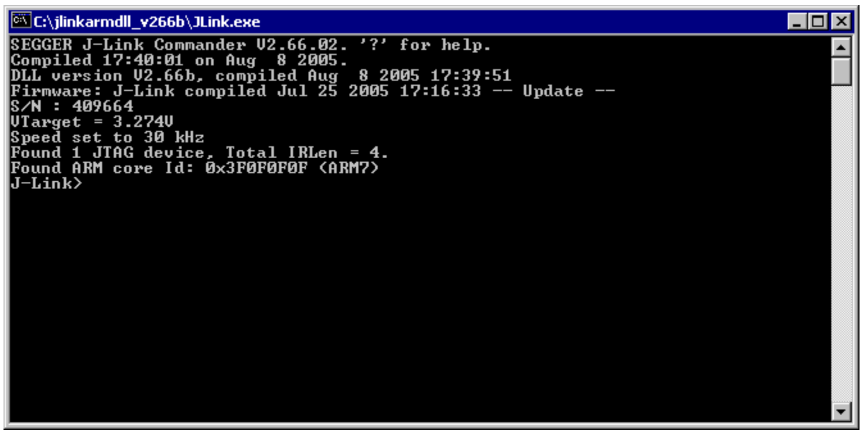
3.2 Connecting the target system

3.2.1 Power-on Sequence

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

3.2.2 Verifying target device connection

If the USB driver is working properly and your J-Link ARM is connected with the host system, you may connect J-Link ARM to your target hardware. Then start `JLink.exe` again which should now display the same J-Link ARM 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 ARM 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 : 409664
VTarget = 3.274V
Speed set to 30 kHz
Found 1 JTAG device, Total IRLen = 4.
Found ARM core Id: 0x3F0F0F0F (ARM7)
J-Link>
```

3.2.3 Problems

If you experience problems with any of the steps described above, it is recommended you read the chapter "Support" on page 59 for troubleshooting tips. If you still do not find appropriate help there and your J-Link ARM is an original Segger J-Link ARM (not an OEM version), 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" on page 59 as well.

3.3 Scan chain configuration

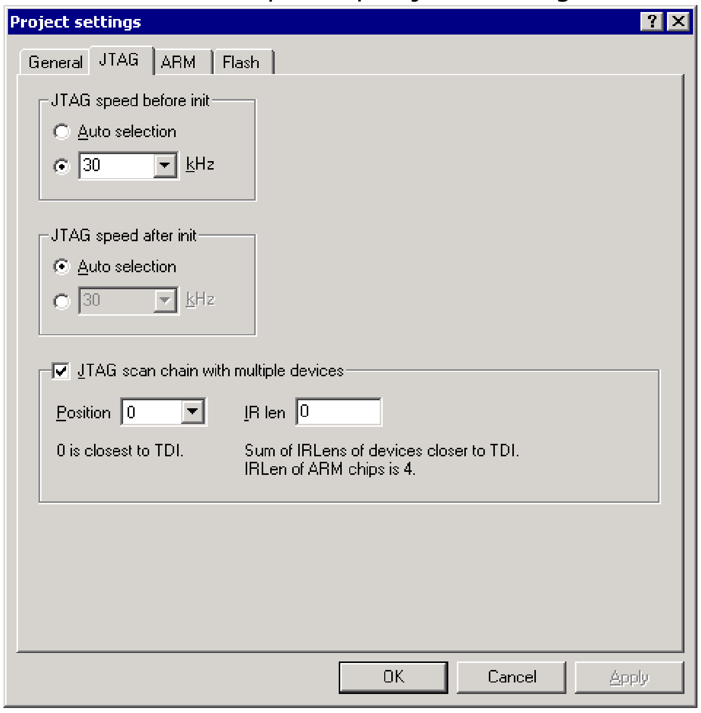
By default, only one ARM device is assumed to be in the JTAG scan chain. If you have multiple devices in the scan chain, you must properly configure it. To do so, one has to specify the exact position of the ARM device that should be addressed. Configuration of the scan is done by the target application.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

3.3.1 Sample configuration dialogs

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; some sample dialogs are shown below.

SEGGER J-Flash configuration dialog

This dialog can be found under "Options|Project settings"


Project settings

JTAG speed before init: Auto selection [✓ 30 kHz]

JTAG speed after init: [✓ Auto selection] 30 kHz

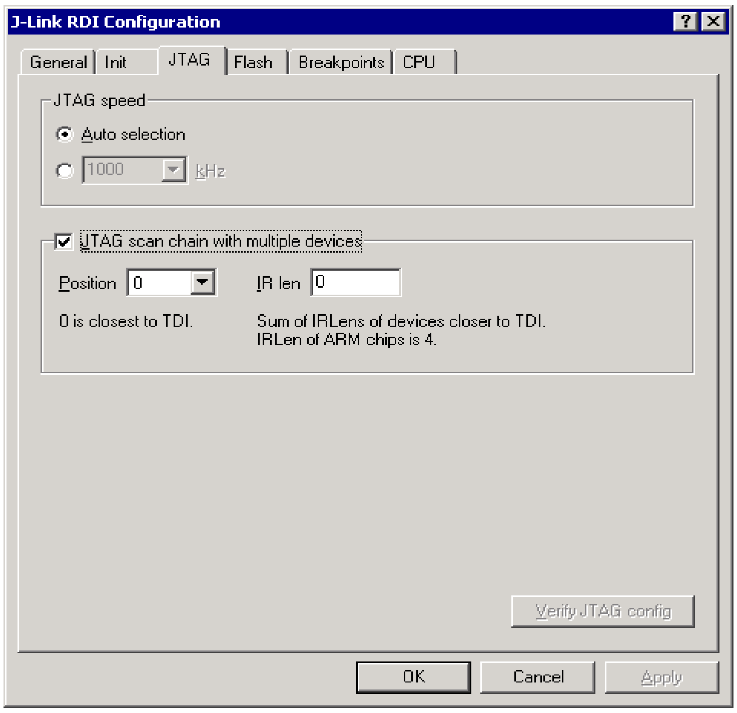
[✓ JTAG scan chain with multiple devices]

Position 0 — 0 is closest to TDI.

IR len 0 — Sum of IRLens of devices closer to TDI. IRLen of ARM chips is 4.

SEGGER J-Link RDI configuration dialog

This dialog can be found under "RDI|Configure" for example in IAR's Embedded Workbench.



J-Link RDI Configuration

General | Init | JTAG | Flash | Breakpoints | CPU

JTAG speed

[✓ Auto selection]

1000 kHz

[✓ JTAG scan chain with multiple devices]

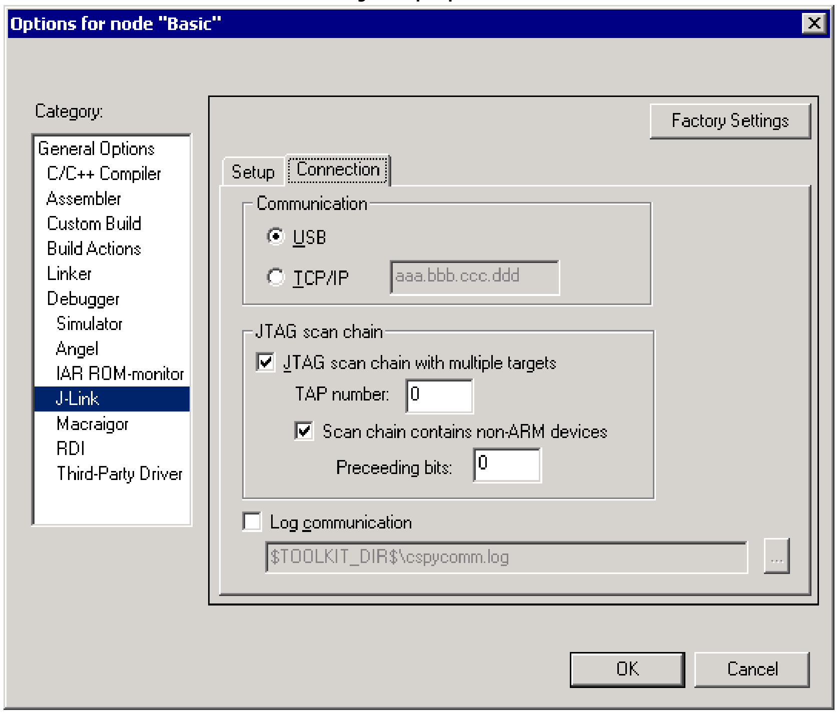
Position: 0   IR len: 0

0 is closest to TDI.

Sum of IRLens of devices closer to TDI.
IRLen of ARM chips is 4.

IAR J-Link configuration dialog

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



Options for node "Basic"

Category:

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

Setup | Connection

Communication

[✓ USB]

TCP/IP: aaa.bbb.ccc.ddd

JTAG scan chain

[✓ JTAG scan chain with multiple targets]

TAP number: 0

[✓ Scan chain contains non-ARM devices]

Preceeding bits: 0

Log communication: $TOOLKIT_DIR$\cspycomm.log

3.3.2 Determining values for scan chain configuration

When do I need to configure the scan chain?

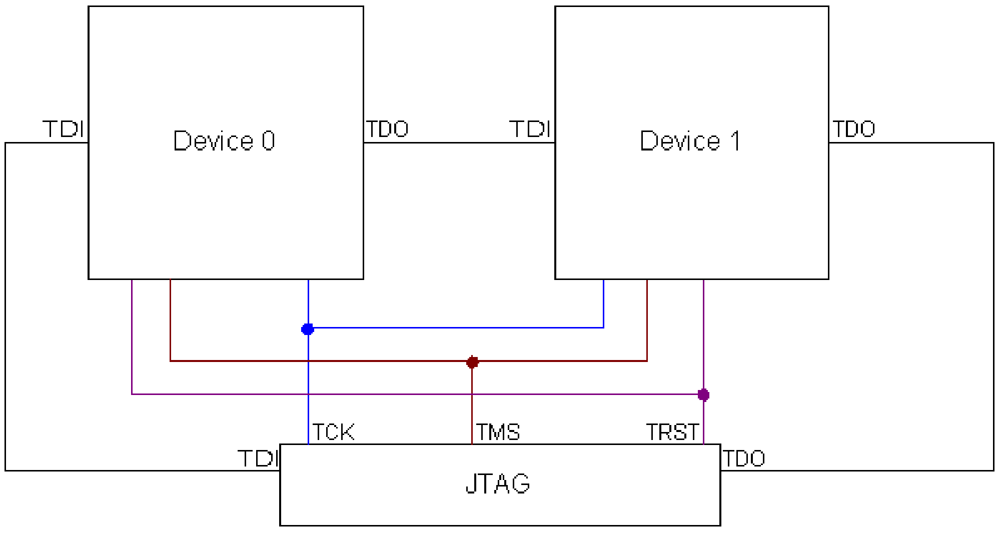
If only one device is connected to the scan chain, the default configuration can be used. In other cases, J-Link ARM may succeed in automatically recognizing the devices on the scan chain, but whether this is possible depends on the devices present on the scan chain.

How do I configure the scan chain ?

2 values need to be known:
The position of the target device in the scan chain and 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.

3.4 JTAG Speed

There are basically three types of speed settings:

- Fixed JTAG speed
- Automatic JTAG speed
- Adaptive clocking

These are explained below.

3.4.1 Fixed JTAG speed

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

3.4.2 Automatic JTAG speed

Selects the maximum JTAG speed handled by the TAP controller.

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

3.4.3 Adaptive clocking

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

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 4

J-Link related software

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

Software	Description
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 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.
J-Link ARM Developer Pack	The J-Link ARM Developer Pack is needed if you want to write your own program with J-Link.
J-Link ARM Flash DLL	An enhanced version of the JLinkARM.DLL, which contains additional API functions for flash programming.
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.
JTAGLoad	Command line tool that opens a svf file and sends the data in it via J-Link to the target.

4.1 Free software

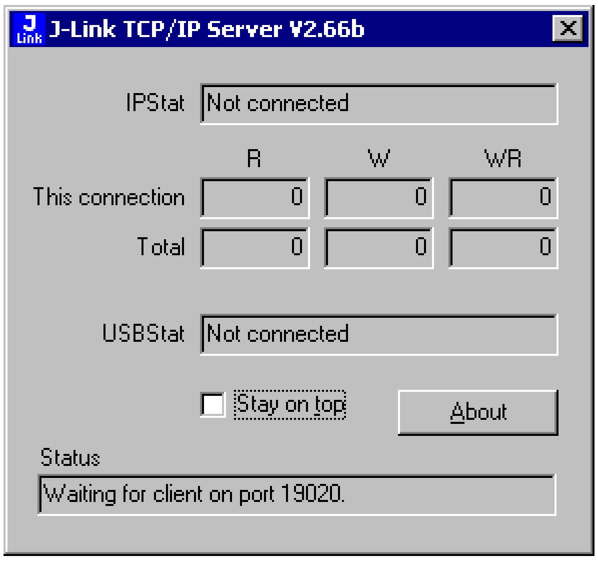
This kind of J-Link related software is shipped together with J-Link ARM and may also be downloaded from www.segger.com. No additional license is required to use this software.

4.1.1 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.

4.1.2 J-Link TCP/IP Server (Remote J-Link use)

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


J-Link TCP/IP Server V2.66b

IPStat: Not connected

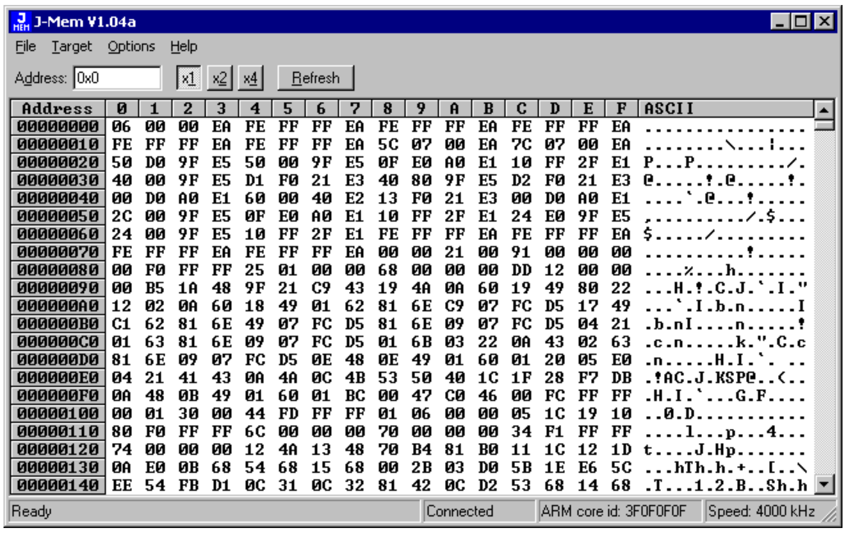
	R	W	WR
This connection	0	0	0
Total	0	0	0

USBStat: Not connected

Status: Waiting for client on port 19020.

4.1.3 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 acces 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.



Address: 0x0

```
Address   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  ASCII
00000000 06 00 00 EA FE FF FF EA FE FF FF EA FE FF FF EA ................
00000010 FE FF FF EA FE FF FF EA 5C 07 00 EA 7C 07 00 EA ........\...¦...
00000020 50 D0 9F E5 50 00 9F E5 0F E0 A0 E1 10 FF 2F E1 P...P........./.
00000030 40 00 9F E5 D1 F0 21 E3 40 80 9F E5 D2 F0 21 E3 @.....!.@.....!.
00000040 00 D0 A0 E1 60 00 40 E2 13 F0 21 E3 00 D0 A0 E1 ....`.@...!.....
00000050 2C 00 9F E5 0F E0 A0 E1 10 FF 2F E1 24 E0 9F E5 ,........./.$...
00000060 24 00 9F E5 10 FF 2F E1 FE FF FF EA FE FF FF EA $...../.........
00000070 FE FF FF EA FE FF FF EA 00 00 21 00 91 00 00 00 ..........!.....
00000080 00 F0 FF FF 25 01 00 00 68 00 00 00 DD 12 00 00 ....%...h.......
00000090 00 B5 1A 48 9F 21 C9 43 19 4A 0A 60 19 49 80 22 ...H.!.C.J.`.I."
000000A0 12 02 0A 60 18 49 01 62 81 6E C9 07 FC D5 17 49 ...`.I.b.n.....I
000000B0 C1 62 81 6E 49 07 FC D5 81 6E 09 07 FC D5 04 21 .b.nI....n.....!
000000C0 01 63 81 6E 09 07 FC D5 01 6B 03 22 0A 43 02 63 .c.n.....k.".C.c
000000D0 81 6E 09 07 FC D5 0E 48 0E 49 01 60 01 20 05 E0 .n.....H.I.`. ..
000000E0 04 21 41 43 0A 4A 0C 4B 53 50 40 1C 1F 28 F7 DB .!AC.J.KSP@..(..
000000F0 0A 48 0B 49 01 60 01 BC 00 47 C0 46 00 FC FF FF .H.I.`...G.F....
00000100 00 01 30 00 44 FD FF FF 01 06 00 00 05 1C 19 10 ..0.D...........
00000110 80 F0 FF FF 6C 00 00 00 70 00 00 00 34 F1 FF FF ....l...p...4...
00000120 74 00 00 00 12 4A 13 48 70 B4 81 B0 11 1C 12 1D t....J.Hp.......
00000130 0A E0 0B 68 54 68 15 68 00 2B 03 D0 5B 1E E6 5C ...hTh.h.+..[..\
00000140 EE 54 FB D1 0C 31 0C 32 81 42 0C D2 53 68 14 68 .T...1.2.B..Sh.h
```

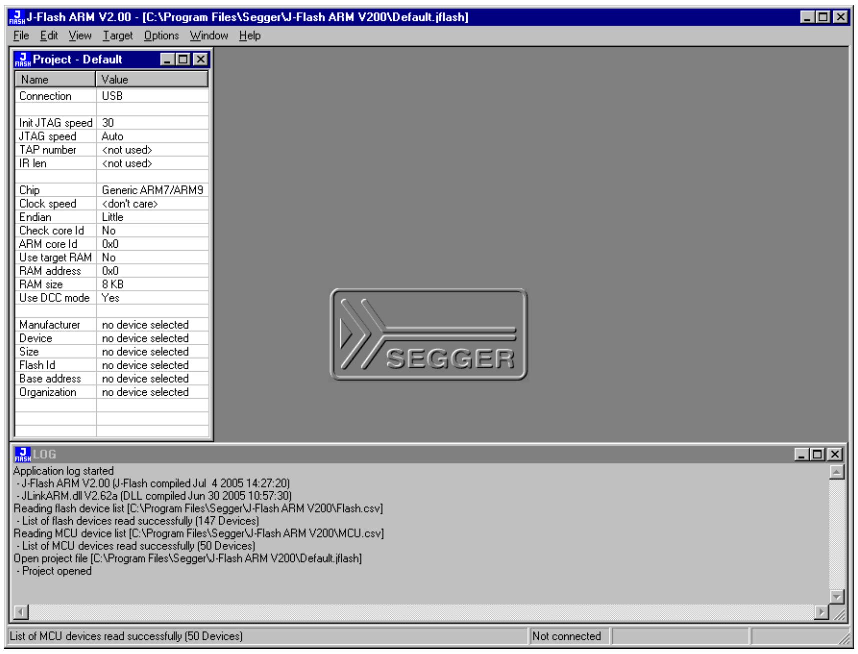
Ready | Connected | ARM core id: 3F0F0F0F | Speed: 4000 kHz

4.2 Additional software

The software described in this section requires separate licenses from Segger. You can however download the software from www.segger.com. Evaluation licenses are offered as well. For further information go to our website or contact us directly.

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

J-Flash ARM is a PC 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, CHECKSUM, 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 ARM you are required to obtain a license key from us.


J-Flash ARM V2.00 - [C:\Program Files\Segger\J-Flash ARM V200\Default.jflash]
File Edit View Target Options Window Help
Project - Default
SEGGER
LOG

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

```
Application log started
 - J-Flash ARM V2.00 (J-Flash compiled Jul  4 2005 14:27:20)
 - JLinkARM.dll V2.62a (DLL compiled Jun 30 2005 10:57:30)
Reading flash device list [C:\Program Files\Segger\J-Flash ARM V200\Flash.csv]
 - List of flash devices read successfully (147 Devices)
Reading MCU device list [C:\Program Files\Segger\J-Flash ARM V200\MCU.csv]
 - List of MCU devices read successfully (50 Devices)
Open project file [C:\Program Files\Segger\J-Flash ARM V200\Default.jflash]
 - Project opened
```

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

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 kByte/sec (dep. on flash device).
- Very high speed blank check: App. 16 Mybte /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.

4.2.2 J-Link ARM Developer Pack

The J-Link ARM Developer Pack is needed if you want to write your own program with J-Link ARM. 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 ARM available through its 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. 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, typedefs 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.)

4.2.3 J-Link ARM Flash DLL

This is an enhanced version of the JLinkARM.DLL which contains additional API functions for flash programming. The add. 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.

4.2.4 RDI Support

The J-Link RDI software is an RDI interface for J-Link ARM. It makes it possible to use J-Link ARM 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 a RDI license in addition to its own license.

4.2.4.1 Flash download

The 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 where RAM.

4.2.4.2 Flash breakpoints

The J-Link RDI software contains an add. feature, called Flash break points (short FlashBPs). This feature requires an add. 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 RAMcode specially 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 ARM 9 cores have 2 breakpoint units (called "watchpoint units" in ARM's documentation), allowing 2 hardware breakpoints to be set. Hardware breakpoints do not require modification of the program code. Software breakpoints are different: The debugger modifies the program and replaces the breakpointed instruction with a special value. Add. soft BPs do not require add. 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 allows you to set an unlimited number of breakpoints even if your application program is not located in RAM, but in Flash memory.This is a scenario which was very rare before ARM-microcontrollers hit the market. This new technology makes very powerful, yet inexpensive ARM microcontrollers available for systems, which required external RAM before. The downside of this new technology is that it is not possible to debug larger programs on these Micros in RAM, since the RAM is not big enough to hold program and data (typically, these chips contain about 4 times as much flash as RAM), and therefore with standard debuggers, only 2 breakpoints can be set. The 2 breakpoint limit makes debugging very tough; a lot of times the debugger requires 2 breakpoints to simply step over a line of code. With software breakpoints in Flash, this limitation is gone.

How does this work?

Basically its very simple:The J-Link RDI software reprograms a sector of the flash to set or clear a breakpoint.

What performance can I expect?

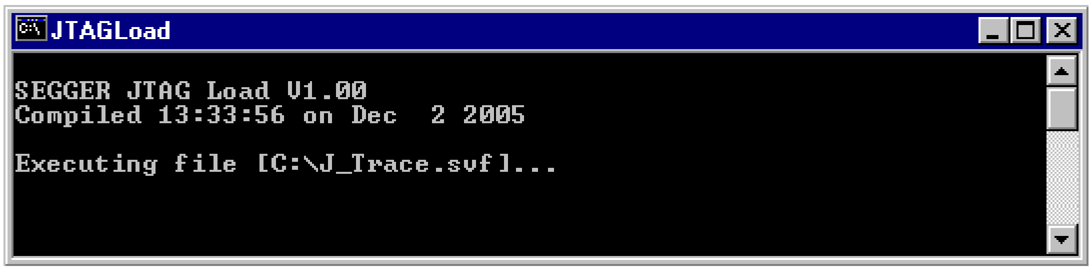
A RAM code, specially designed for this purpose, sets and clears flash breakpoints extremely fast; on micros with fast flash the difference between breakpoints in RAM and Flash is hardly noticeable.

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 then one breakpoint is located in the same flash sector, which allows programming multiple breakpoints by programming just a single sector. The contents of program memory are cached, avoiding time consuming reading of the flash sectors. A smart combination of software and hardware breakpoints allows us to use hardware breakpoints a lot of times, especially when the debugger is source level-stepping, avoiding reprogramming 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.

4.2.5 JTAGLoad (Command line tool)

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



```
SEGGER JTAG Load V1.00
Compiled 13:33:56 on Dec  2 2005

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

Chapter 5

Working with J-Link

This chapter describes the features of J-Link.

5.1 Reset strategies

J-Link ARM 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.

5.1.1 Reset strategies in detail

5.1.1.1 Hardware, halt after reset (normal)

The hardware reset pin is used to reset the CPU. After reset release, J-Link ARM 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.

5.1.1.2 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.

5.1.1.3 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 recommended reset sequence for Analog Devices ADuC7xxx MCUs and works with these chips only.

5.2 Cache handling

Most ARM systems with external memory have at least one cache. Typically, ARM 7 systems with external memory come with a unified cache, wich is used for both code and data. Most ARM 9 systems with external memory come with separate caches for the instruction bus (I-Cache) and data bus (D-Cache) due to the hardware architecture.

5.2.1 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.

5.2.2 Cache clean area

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

5.2.3 Cache handling of ARM 7 cores

Since ARM 7 cores have a unified cache, there is no need to handle the caches during debug.

5.2.4 Cache handling of ARM 9 cores

ARM 9 cores with cache require J-Link ARM to handle the caches during debug. If the processor enters debug state with caches enabled, J-Link does the following:

When entering debug state

J-Link ARM does 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 ARM does the following:
— it restores the stored write behavior for the D-Cache.
— it invalidates the D-Cache.

Note that the implementation of the cache handling is different for different cores. However, the cache is handled correctly for all supported ARM 9 cores.

5.3 Connecting multiple J-Links to your PC

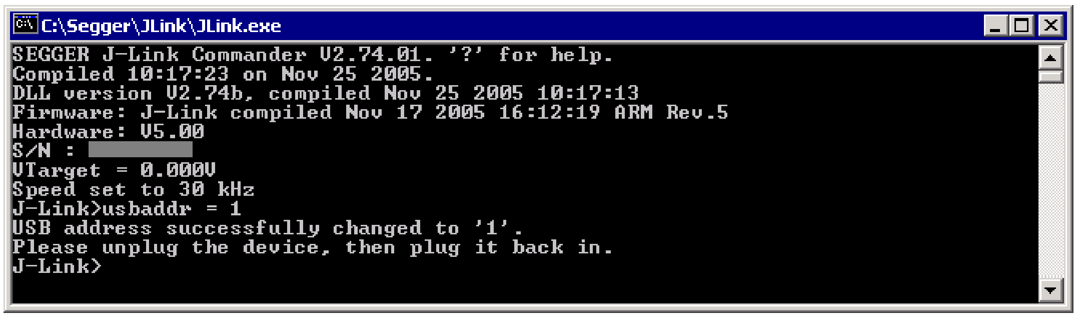
You can connect up to 4 J-Links to your PC. In this case all J-Links must have different USB-Addresses. The default USB-Address is "0".
In order to do this, 3 J-Links must be configured as described below. Every J-Link need its own USB-Driver which can be downloaded from www.segger.com.
This feature is supported by J-Link Rev. 5.0 and up.

5.3.1 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 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 #1 is 101.
The product id (PID) for J-Link #2 is 102 and so on.
A different PID means that J-Link is identified as a different device, requiring a new driver.

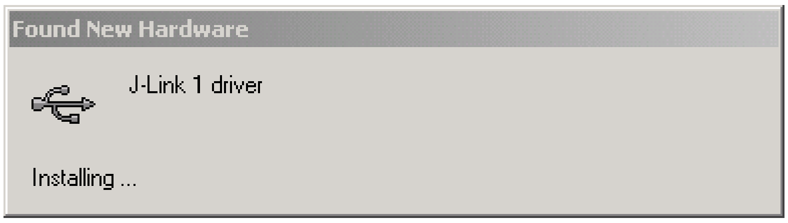
5.3.2 Configuring multiple J-Links

1. Start "JLink.exe" to view your hardware version. Your J-Link needs to be V5.0 or up to continue.
2. Type "usbaddr = 1" to set the J-Link #1.



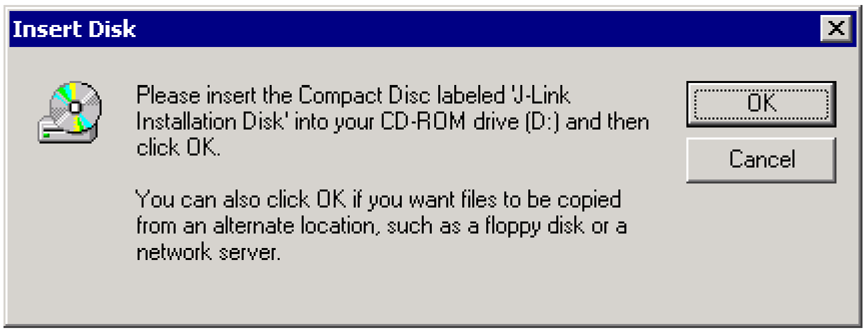
```
SEGGER J-Link Commander V2.74.01. '?' for help.
Compiled 10:17:23 on Nov 25 2005.
DLL version V2.74b, compiled Nov 25 2005 10:17:13
Firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>usbaddr = 1
USB address successfully changed to '1'.
Please unplug the device, then plug it back in.
J-Link>
```

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



J-Link 1 driver

Installing ...

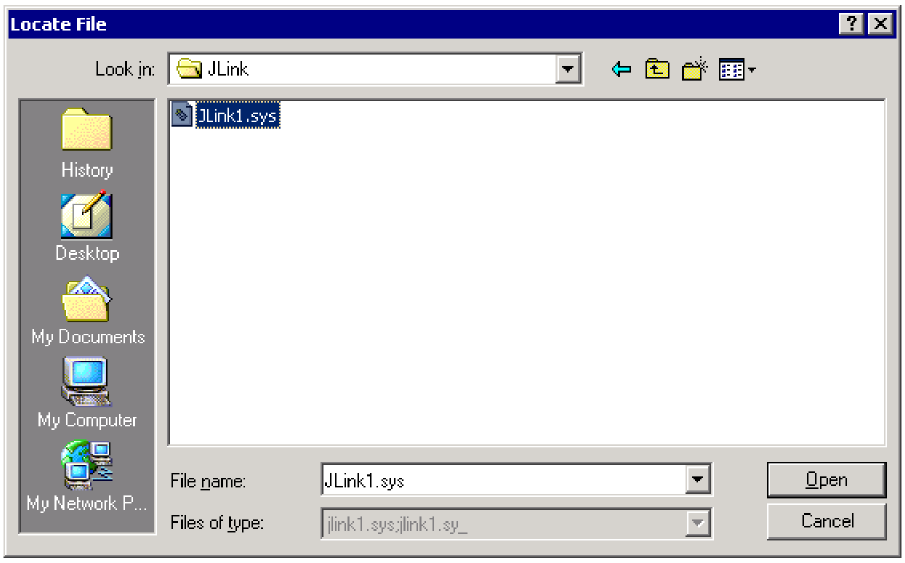


Please insert the Compact Disc labeled 'J-Link Installation Disk' into your CD-ROM drive (D:) and then click OK.

You can also click OK if you want files to be copied from an alternate location, such as a floppy disk or a network server.

5. Press "OK" and browse to the J-Link driver for your new J-Link. For your second

J-Link this would be "JLink1.sys", for your third J-Link this would be "JLink2.sys".



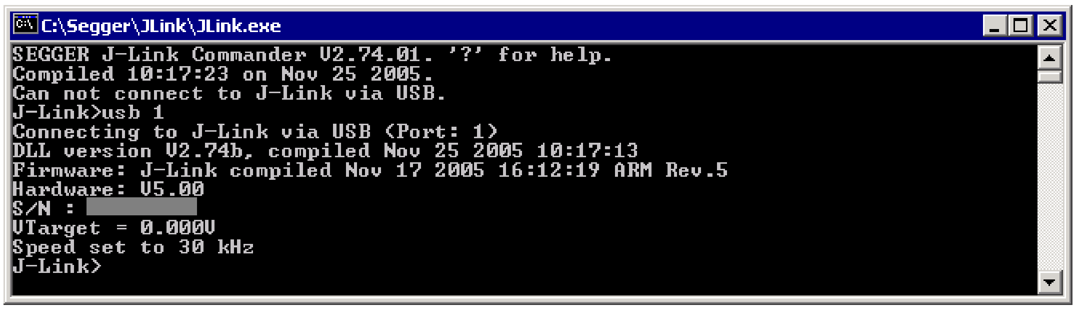
Look in: JLink

File name: JLink1.sys

Files of type: jlink1.sys;jlink1.sy_

5.3.3 Connecting to a J-Link with non default USB-Address

Restart "JLink.exe" and type "usb 1" to connect to J-Link #1.



```
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: V5.00
S/N : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>
```

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

5.4 Multi core debugging

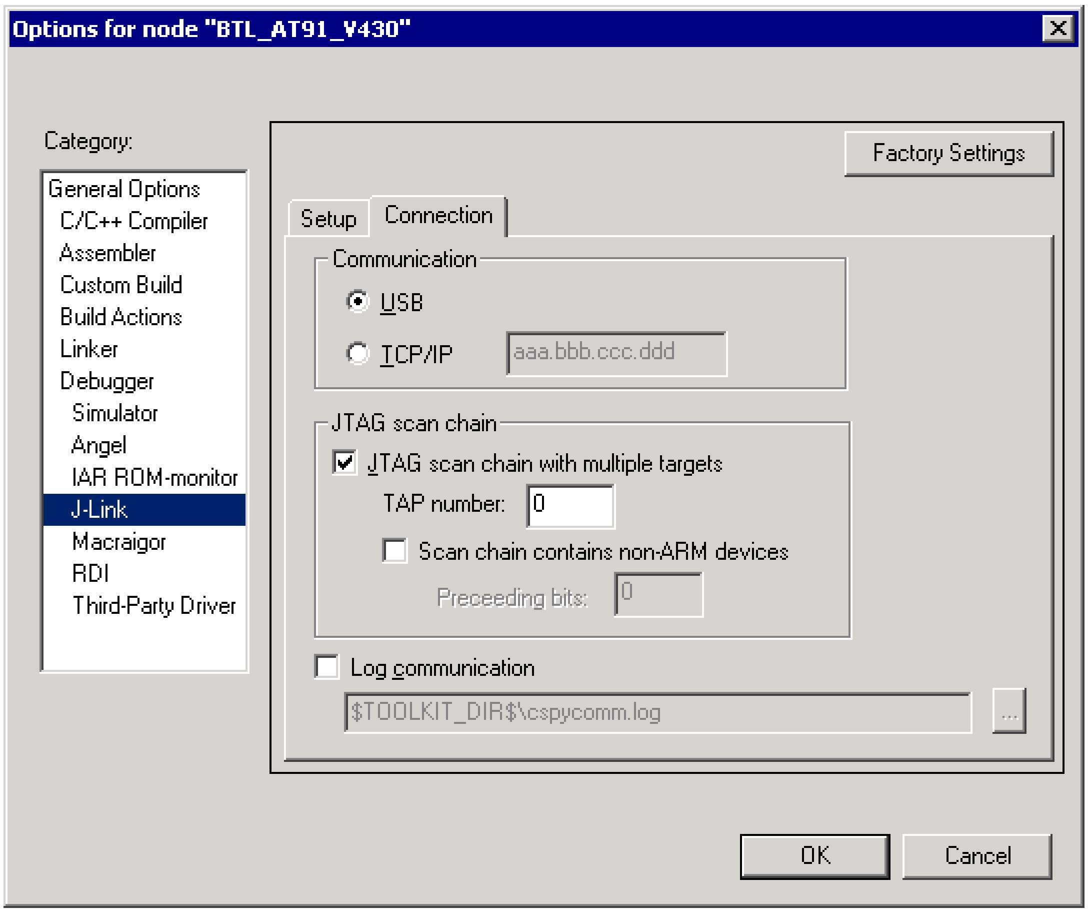
J-Link is able to debug multiple cores on one target system connected to the same scan chain. Configuring and using this feature is described below.

5.4.1 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 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 to debug more than one core on a target at the same time. The core to debug is selected thru the JTAG-settings as described below.

5.4.2 Using multi core debugging in detail

1. Connect your target to J-Link.
2. Start your debugger, for example IAR's Embedded Workbench for ARM.
3. From the menu choose "Project | Options" and configure your scan chain. The picture below shows the configuration for the first ARM core on your target.


Options for node "BTL_AT91_V430"

Category: J-Link

Connection tab:

Communication: [✓ USB] TCP/IP aaa.bbb.ccc.ddd

JTAG scan chain: [✓] JTAG scan chain with multiple targets

TAP number: 0

[ ] Scan chain contains non-ARM devices

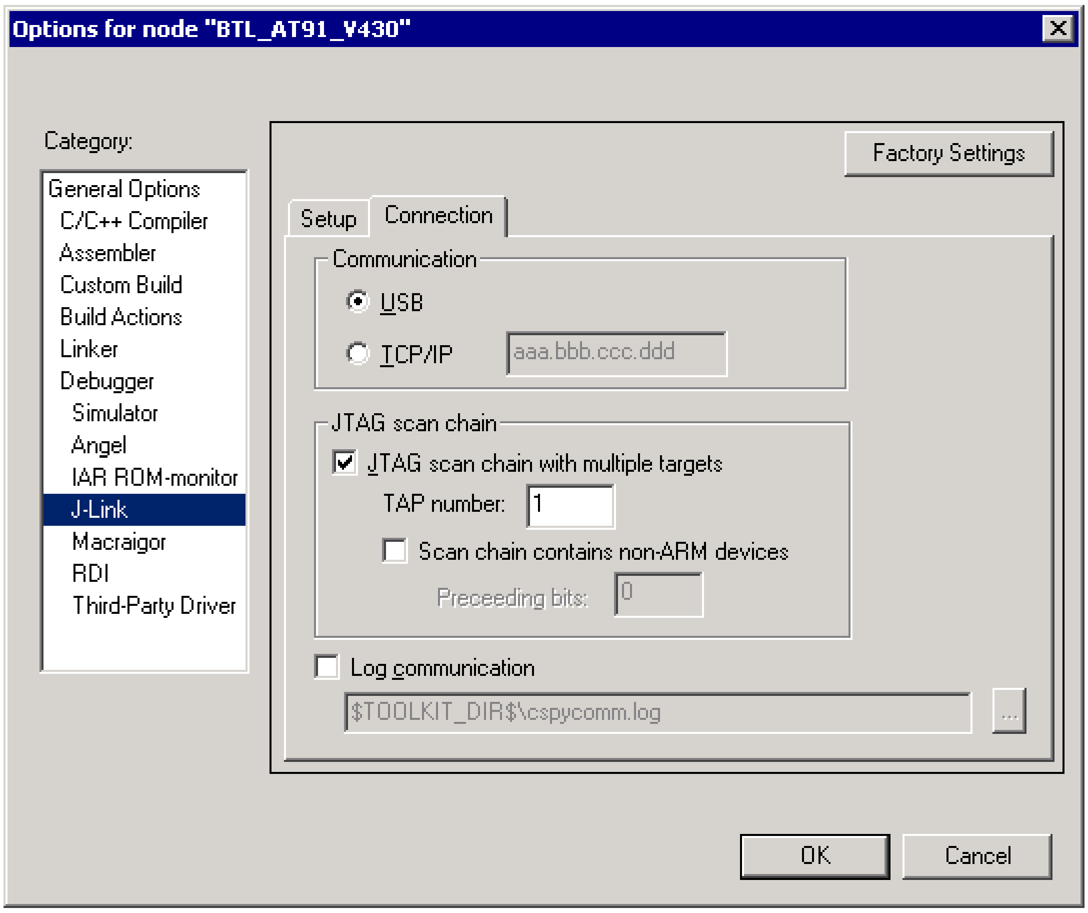
Preceeding bits: 0

[ ] Log communication

$TOOLKIT_DIR$\cspycomm.log

4. Start debugging the first core.

5. Start another debugger, for example another instance of IAR's Embedded Workbench for ARM.
6. From the menu choose "Project | Options" and configure your second scan chain. The picture below shows the configuration for the second ARM core on your target.



Communication: USB (selected); TCP/IP: aaa.bbb.ccc.ddd

JTAG scan chain: [✓] JTAG scan chain with multiple targets; TAP number: 1; [ ] Scan chain contains non-ARM devices; Preceeding bits: 0

[ ] Log communication: $TOOLKIT_DIR$\cspycomm.log

7. Start debugging your second core.

Example:

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

Cores to debug are marked in blue.

5.4.3 Things you should be aware of

Multi core debugging is more difficult than single core debugging. You should be aware of the following pitfalls:

5.4.3.1 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.
Example:

Core #1:	2MHz maximum JTAG speed
Core #2:	4MHz maximum JTAG speed

Scan chain: 2MHz maximum JTAG speed

5.4.3.2 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.

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 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.
TRST	Input (optional)	The optional test reset (TRST) 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

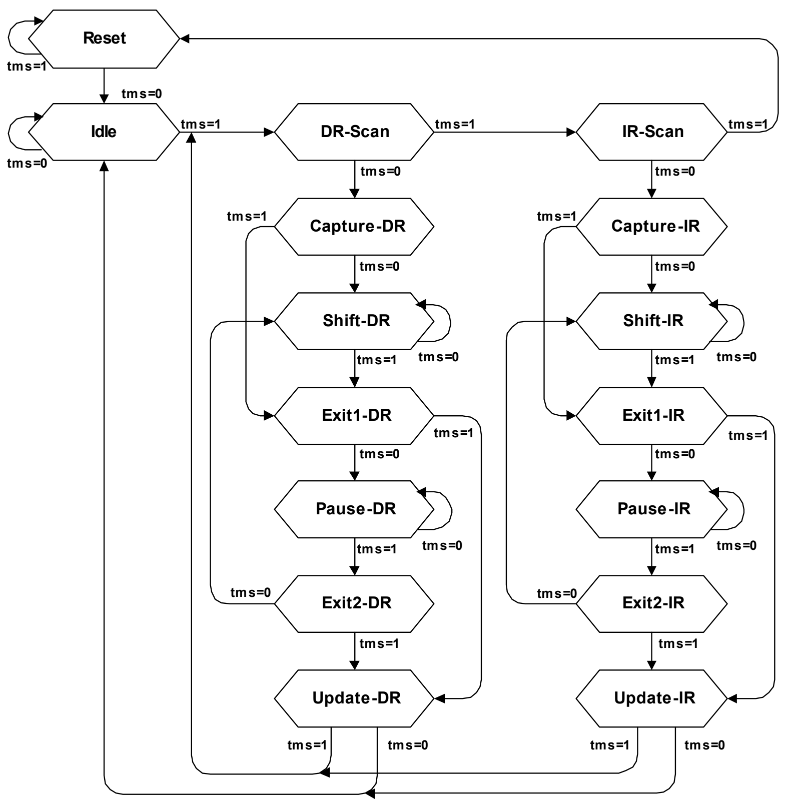
A test data register which allows the testing of board interconnections, access to input and output of components when testing their system logic and so on.

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.



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

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 approx. 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 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)

6.4 Flash programming

J-Link 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 work ?

This requires extra code. This extra code typically downloads a program into the RAM of the target system, which is able to erase and program the flash. This program is called Ram code and "knows" how to program the flash; it contains an implementation of the flash programming algorithm for the particular flash. Different flash chips have different programming algorithms; the programming algorithm also depends on other things such as endianess of the target system and organization of the flash memory (e.g. 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 PC (via DCC, JTAG and J-Link), 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 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 since 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. 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 add. 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; please 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. In order 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 a workbench / 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 the target system.

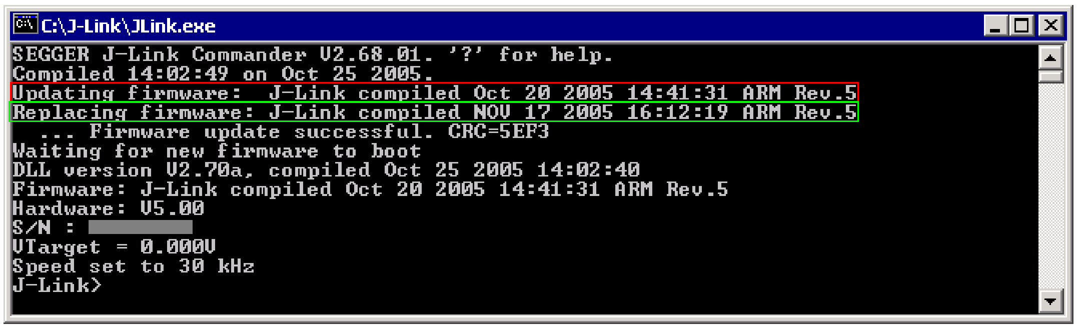
6.5 J-Link firmware

The heart of J-Link is a microcontroller. The firmware is the software executed by the microcontroller inside of the J-Link. The J-Link 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, `JLinkARM.dll` checks if its embedded firmware is newer than the one used in the J-Link. It will then update the firmware automatically. This process takes less than 3 seconds and does not require reboot.

It is recommended that you always use the latest version of `JLinkARM.dll`.



```
SEGGER J-Link Commander V2.68.01. '?' for help.
Compiled 14:02:49 on Oct 25 2005.
Updating firmware:  J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Replacing firmware: J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5
  ... Firmware update successful. CRC=5EF3
Waiting for new firmware to boot
DLL version V2.70a, compiled Oct 25 2005 14:02:40
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>
```

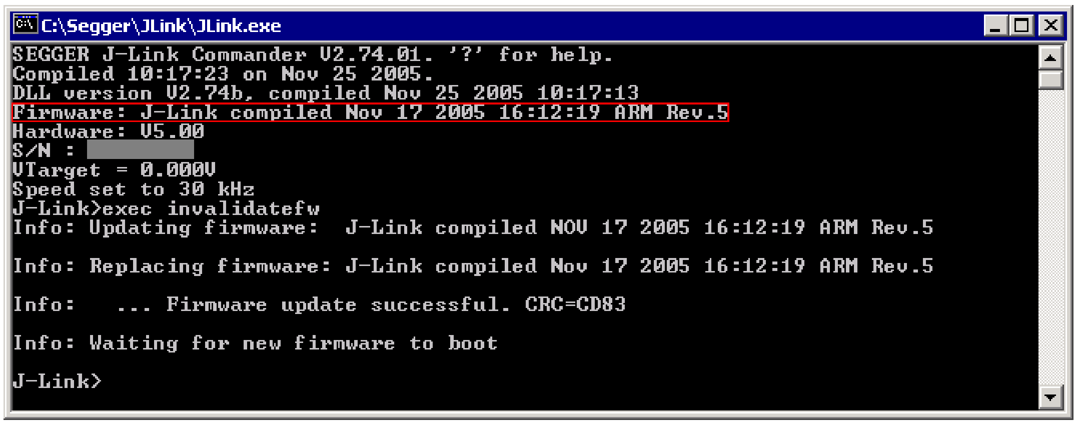
The red box identifies the new firmware.
The green box identifies the old firmware which has been replaced.

6.5.2 Invalidating the firmware

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

Downdating J-Link 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.

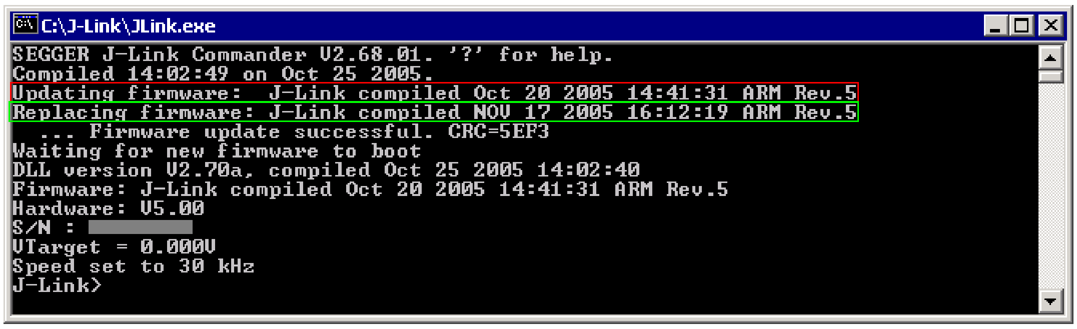
To downdate J-Link, you need to invalidate the current J-Link firmware, using the command "exec InvalidateFW".



```
SEGGER J-Link Commander V2.74.01. '?' for help.
Compiled 10:17:23 on Nov 25 2005.
DLL version V2.74b, compiled Nov 25 2005 10:17:13
Firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>exec invalidatefw
Info: Updating firmware:  J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5

Info: Replacing firmware: J-Link compiled Nov 17 2005 16:12:19 ARM Rev.5

Info:   ... Firmware update successful. CRC=CD83

Info: Waiting for new firmware to boot

J-Link>
```

The red box contains information about the formerly used J-Link ARM firmware version.

Use an application (for example JLink.exe) which uses the desired version of `JLinkARM.dll`. This automatically replaces the invalidated firmware with its embedded firmware.



```
SEGGER J-Link Commander V2.68.01. '?' for help.
Compiled 14:02:49 on Oct 25 2005.
Updating firmware:  J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Replacing firmware: J-Link compiled NOV 17 2005 16:12:19 ARM Rev.5
  ... Firmware update successful. CRC=5EF3
Waiting for new firmware to boot
DLL version V2.70a, compiled Oct 25 2005 14:02:40
Firmware: J-Link compiled Oct 20 2005 14:41:31 ARM Rev.5
Hardware: V5.00
S/N : [redacted]
VTarget = 0.000V
Speed set to 30 kHz
J-Link>
```

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

Chapter 7

FAQs

You can find in this chapter a collection of frequently asked questions (FAQs) together with answers.

7.1 FAQs

Q: Which CPUs are supported?
A: J-Link should work with any ARM7 / ARM9 core. For a list of supported cores see section “Supported ARM Cores” on page 12.

Q: What is the maximum JTAG speed supported by J-Link?
A: J-Links maximum supported JTAG speed is 8MHz.

Q: What is the maximum download speed?
A: The maximum download speed is currently about 150 kByte/sec 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?
A: Yes, you can access all individual ICE registers via J-Link.

Q: I want to write my own application and use J-Link. Is this possible?
A: Yes. We offer a dedicated devloper pack. See section “J-Link ARM Developer Pack” on page 33 for further information.

Q: Can I use J-Link 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.

Q: Can J-Link 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 as well as the supply voltage and can be useful to detect hardware problems on the target system.

Q: J-Link 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 is a suitable solution for the majority of development tasks as well as for production purposes. Some features that are available for J-Link, 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 - in contrast to most other JTAG emulators for ARM cores - not require the user to specify a cache clean area?
A: J-Link handles cache cleaning directly thru JTAG commands. Unlike other emulators, it does not have to download code to the target system. This makes setting up J-Link easier. Therefore, a cache clean area is not required.

Chapter 8

Support

This chapter contains troubleshooting tips together with solutions for common problems which might occur when using J-Link. There are several steps you can take before contacting support. Performing these steps can solve many problems and often eliminates the need for assistance.

8.1 Troubleshooting

8.1.1 General procedure

If you experience problems with a J-Link, you should follow the steps below to solve these problems:

1. Close all running applications on your host system.
2. Disconnect the J-Link device from USB.
3. Power-off target.
4. Re-connect J-Link 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 device from USB.
9. Power-off target.
10. Re-connect J-Link with host system (attach USB cable).
11. Power-on target.
12. Start `JLink.exe`.
13. If `JLink.exe` reports the J-Link serial number and the target processor's core ID, the J-Link 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 with a logic analyzer or oscilloscope. Follow the instructions in section 8.2.
15. If your problem persists and you own an original Segger J-Link (not an OEM version), see section "Contacting support" on page 62.

8.1.2 Typical problem scenarios

J-Link LED is off

Meaning:
The USB connection does not work.
Remedy:
Check the USB connection. Try to re-initialize J-Link by disconnecting and reconnecting it. Make sure that the connectors are firmly attached. Check the cable connections on your J-Link and the computer. If this does not solve the problem, please check if your cable is defective. If the USB cable is ok, try a different PC.

J-Link LED is flashing at a high frequency

Meaning:
J-Link could not be enumerated by the USB controller.
Most likely reasons:
a.) Another program is already using J-Link.
b.) The J-Link USB driver does not work correctly.
Remedy:
a.) Close all running applications and try to reinitialize J-Link 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 19.

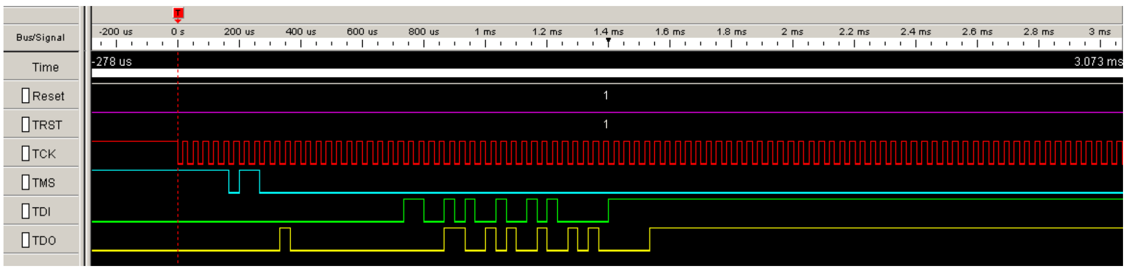
J-Link does not get any connection to the target

Most likely reasons:
a.) The JTAG cable is defective
b.) The target hardware is defective
Remedy:
Please 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 and the target device.

8.2.1 Start sequence



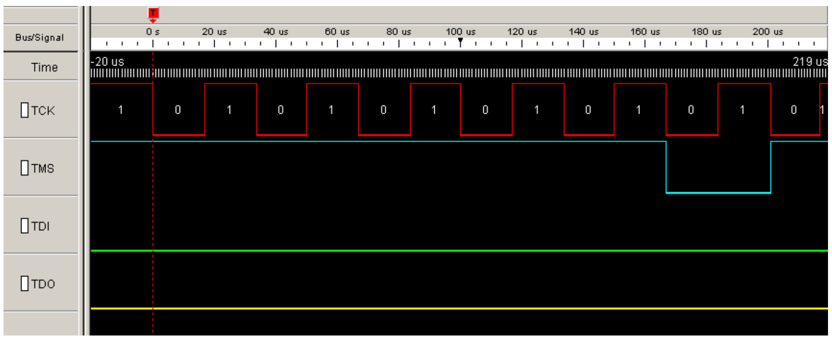
This is the signal sequence output by J-Link at start of JLink.exe. It should be used as reference when tracing potential J-Link related hardware problems.
The sequence consists of the following sections:

- 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.

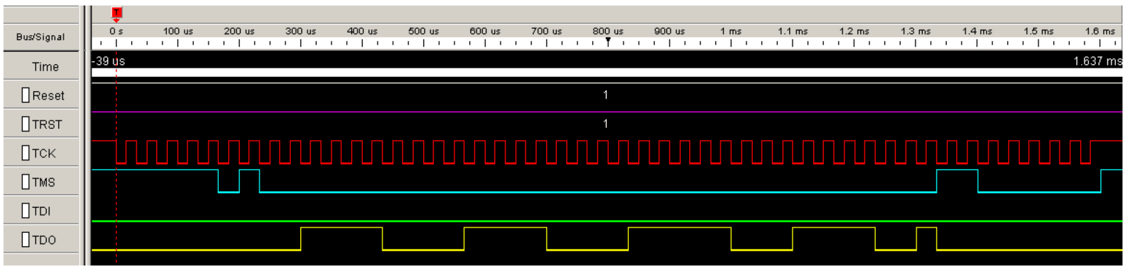
J-Link checks the output of the device (output on TDO) for the signature to measure the IR length. For ARM7 / ARM9 chips, the IR length is 4, which means TDO shifts out the data shifted in on TDI with 4 clock cycles delay. If you compare the screenshot with your own measurements, the signals of TCK,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

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 ID sequence



This screenshot shows the signal sequence when issuing an identify command in `JLink.exe`. The sequence consists of the following sections:

- 5 clocks: TMS high. Brings TAP controller into RESET state.
- 1 clock: TMS low: Brings TAP controller into IDLE state.
- 1 clock: TMS high: Brings TAP controller into DR-SCAN state.
- 2 clock: TMS low: Brings TAP controller into DR-SHIFT state.
- 31 clocks: TMS low:Capture DR: Read the core ID.
- 2 clocks: TMS high: Brings TAP controller into UPDATE-DR state.

Please note that the actual core ID reported via TDO depends on the particular core used.

8.2.3 Troubleshooting

If your measurements of TCK, TMS and TDI (the signals output by J-Link) 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 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 60. You may also try your J-Link 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.

If you need to contact support, please send the following information to support@segger.com:

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

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

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 and it waits for the returned clock before generating the next clock pulse. The technique allows the J-Link 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 programer-specified subroutine. Used by ARM to handle semihosting.

TAP Controller

Logic on a device which allows access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1.

Target

The actual processor (real silicon or simulated) on which the application program isrunning.

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 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 Interface Unit.

Test Access Port (TAP)

The port used to access a device's TAP Controller. Comprises TCK, TMS, TDI, TDO and nTRST (optional).

Transistor-transistor logic (TTL)

A type of logic design in which two bipolar transistors drive the logic output to one or zero. LSI and VLSI logic often used TTL with HIGH logic level approaching +5V and LOW approaching 0V.

Watchpoint

A location within the image that will be monitored and that will cause execution to stop when it changes.

Word

A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Index