IAR J-Link and IAR J-Trace

User Guide

JTAG Emulators for

ARM Cores

EDITION NOTICE

Second edition: May 2009

Part number: J-Link/J-TraceARM-2

Internal reference: ISUD

Contents

Preface

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

About this guide

This guide provides an overview over the major features of J-Link and J-Trace, gives you some background information about JTAG, ARM and Tracing in general and describes J-Link and J-Trace related software packages. Finally, the chapter *Support and FAQs* on page 107 helps to troubleshoot common problems.

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

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

TYPOGRAPHIC CONVENTIONS

This manual uses the following typographic conventions:

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

Table 1: Typographic conventions

Literature and references

To gain deeper understanding of technical details, see:

Reference	Title	Comments
[ETM]	Embedded Trace Macrocell™ Architecture Specification, ARM IHI 0014J	This document defines the ETM standard, including signal protocol and physical interface. It is publicly available from ARM (www.arm.com).

Table 2: Literature and references

Introduction

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

Overview about the J-Link product family

The J-Link product family includes different variations of J-Link, designed for different purposes / target devices. This section gives a short overview about the different products which are part of the J-Link product family.

J-LINK

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

J-TRACE

J-Trace is a JTAG emulator designed for ARM cores which includes trace (ETM) support. It connects via USB to a PC running Microsoft Windows 2000, Windows XP, Windows 2003 or Windows Vista. 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.

Common features of the J-Link product family

- USB 2.0 interface
- Any ARM7/ARM9 and Cortex-M3 core supported, including thumb mode
- Automatic core recognition
- Maximum JTAG speed 12 MHz
- Seamless integration into the IAR Embedded Workbench® IDE
- No power supply required, powered through USB
- Support for adaptive clocking
- All JTAG signals can be monitored, target voltage can be measured
- Support for multiple devices
- Fully plug and play compatible
- Standard 20-pin JTAG connector, standard 38-pin JTAG+Trace connector
- USB and 20-pin ribbon cable included
- Memory viewer (J-Mem) included
- TCP/IP server included, which allows using J-Trace via TCP/IP networks
- Flash programming software (J-Flash) available
- Full integration with the IAR C-SPY® debugger; advanced debugging features available from IAR C-SPY debugger.
- Adapter for 5V JTAG targets available
- Target power supply via pin 19 of the JTAG/SWD interface (up to 300 mA to target with overload protection)

Supported ARM Cores

J-Link / J-Trace has been tested with the following cores, but should work with any ARM7/ARM9 and Cortex-M3 core.

- ARM7TDMI (Rev 1)
- ARM7TDMI (Rev 3)
- ARM7TDMI-S (Rev 4)
- ARM720T
- ARM920T
- ARM922T
- ARM926EJ-S
- ARM946E-S
- ARM966E-S
- Cortex-M3

Requirements

Host System

To use J-Link or J-Trace you need a host system running Windows 2000, Windows XP, Windows 2003, or Windows Vista.

Target System

An ARM7/ARM9/ARM11 or Cortex-M3 target system is required. The system should have a standardized 20-pin connector as defined by ARM Ltd. for a simple JTAG connection. The individual pins are described in section *20-pin JTAG/SWD connector* on page 77.

To use tracing with J-Trace, you need a 38-pin connector on your target board as defined by ARM Ltd. and described under *38-pin Mictor JTAG and Trace connector* on page 80. The individual pins are described in section *Pinout* on page 82.

Licensing

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

Introduction

J-Link functionality can be enhanced by flash breakpoints (`FlashBP`). This feature do not come with J-Link and need additional licenses. This chapter describes the licensing options.

License types

FlashBP requires an additional license, there are three types of licenses:

Built-in License

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

Key-based license

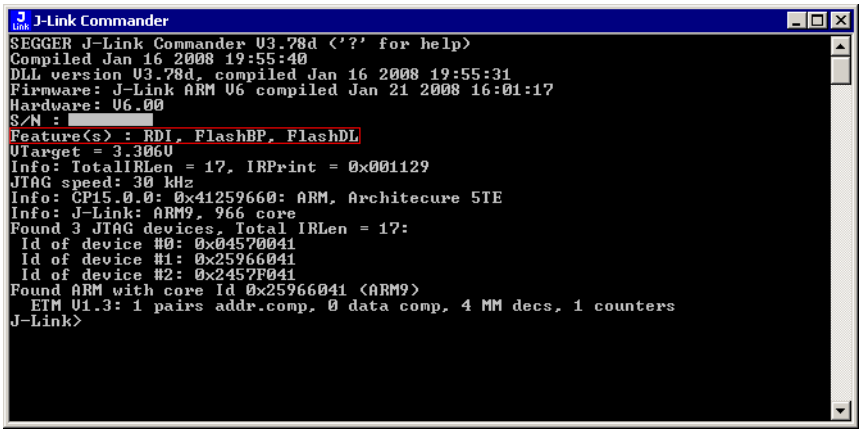
This type of license is used if you already have a J-Link probe, but want to enhance its functionality by using `J-Link FlashBP`. In addition to that, the key-based license is used for trial licenses. To enable this type of license you need to obtain a license key. This license key has to be added to the J-Link license management. How to enter a license key is described in detail in *Licensing* on page 57. Every license can be used on different PCs, but only with the J-Link the license is for. This means that if you want to use J-Link ARM FlashBP with other J-Links, every J-Link needs a license.

Device-based license

The device-based license comes with the J-Link software and is available for some devices. For a complete list of devices which have built-in licenses, please refer to *Device list* on page 14. The device-based license has to be activated via the debugger. How to activate a device-based license is described in detail in the section *Activating a device-based license* on page 14.

BUILT-IN LICENSE

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



KEY-BASED LICENSE

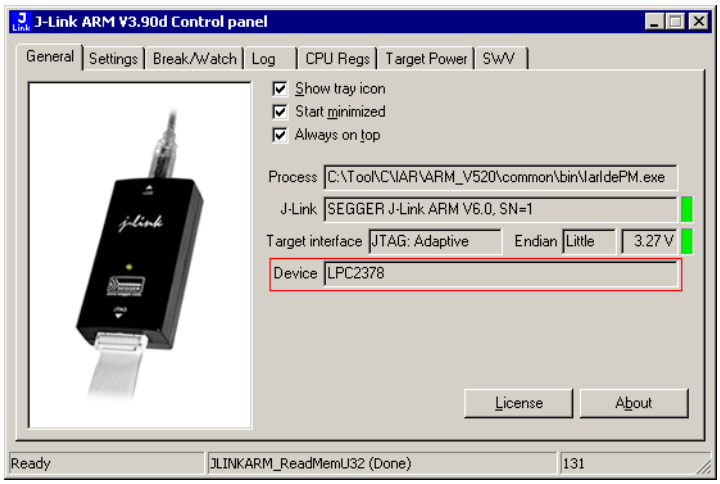
When using a key-based license, a license key is required in order to enable the J-Link features `J-Link FlashBP`. License keys can be added via the license manager. How to enter a license via the license manager is described in *Licensing* on page 57. Like the built-in license, the key-based license is only valid for one J-Link, so if another J-Link is used it needs a separate license.

DEVICE-BASED LICENSE

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

Activating a device-based license

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



Device list

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

Manufacturer	Name	Licenses
NXP	LPC2101	J-Link ARM FlashBP
NXP	LPC2102	J-Link ARM FlashBP
NXP	LPC2103	J-Link ARM FlashBP
NXP	LPC2104	J-Link ARM FlashBP
NXP	LPC2105	J-Link ARM FlashBP
NXP	LPC2106	J-Link ARM FlashBP
NXP	LPC2109	J-Link ARM FlashBP
NXP	LPC2114	J-Link ARM FlashBP
NXP	LPC2119	J-Link ARM FlashBP
NXP	LPC2124	J-Link ARM FlashBP
NXP	LPC2129	J-Link ARM FlashBP
NXP	LPC2131	J-Link ARM FlashBP
NXP	LPC2132	J-Link ARM FlashBP
NXP	LPC2134	J-Link ARM FlashBP
NXP	LPC2136	J-Link ARM FlashBP
NXP	LPC2138	J-Link ARM FlashBP
NXP	LPC2141	J-Link ARM FlashBP
NXP	LPC2142	J-Link ARM FlashBP

Table 3: Device list

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

Table 3: Device list (Continued)

Legal use of original J-Link software

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

Use of software

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

Products

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

J-LINK

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

Licenses

Comes with built-in licenses for J-Link ARM FlashBP for some devices. For a complete list of devices which are supported by the built-in licenses, please refer to *Device list* on page 14.

J-TRACE

J-Trace is a JTAG emulator designed for ARM cores which includes trace (ETM) support. It connects via USB to a PC running Microsoft Windows 2000, Windows XP, Windows 2003 or Windows Vista. 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.

J-Link OBs

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

Illegal Clones

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

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

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

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

Setup

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

Installing the J-Link ARM software and documentation

J-Link is shipped with a bundle of applications, corresponding manuals and some example projects, and the kernel mode J-Link USB driver. Some of the applications require an additional license.

Refer to chapter *J-Link and J-Trace related software* on page 21 for an overview about the J-Link software and documentation pack.

SETUP PROCEDURE

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

Note: Check for J-Link related downloads on our website:

http://www.iar.com/jlinkarm

1. Connect your computer and the J-Link debug probe using the USB cable. Do not connect the J-Link debug probe to the evaluation board, yet. The green LED on the front panel of the J-Link debug probe will blink for a few moments while Windows searches for a USB driver.
2. When you do this for the first time, Windows will start the Install wizard. Choose **Install from a specific location**.
3. When asked to locate the USB drivers, click the browse button and navigate to the directory `Program Files\IAR Systems\Embedded Workbench 5.0\Kickstart\arm\drivers\JLink`. This assumes that you already have installed the IAR Embedded Workbench IDE. If not, make sure to intall it.

   Note that Windows XP might display a warning that the driver is not certified by Microsoft. Ignore this warning and click **Continue**.
4. Click **Finish**. The green LED on the J-Link debug probe stops blinking. The installation is now ready.
5. Remove the USB cable that connects the computer and your J-Link.

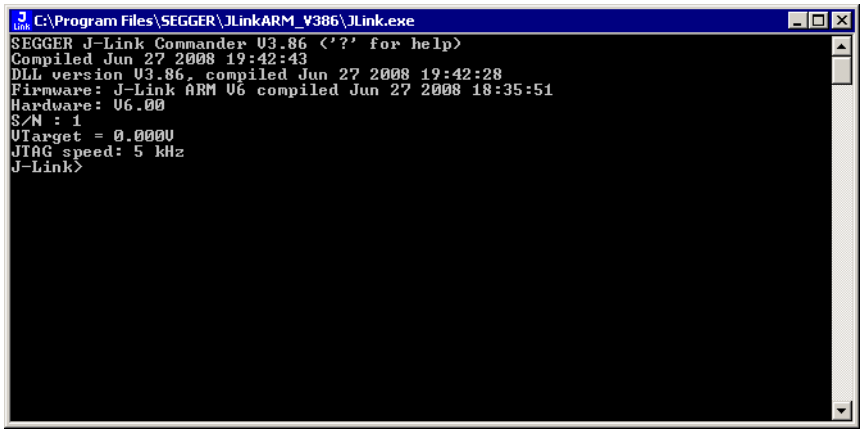
Setting up the USB interface

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

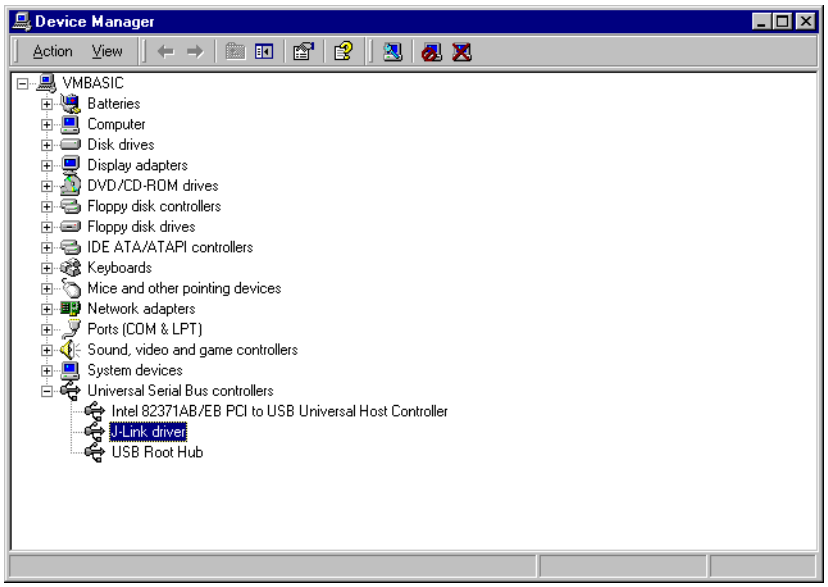
VERIFYING CORRECT DRIVER INSTALLATION

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

Start the provided sample application `JLink.exe` available in the `arm\bin` directory in your product installation, which should display the compilation time of the J-Link firmware, the serial number, a target voltage of 0.000V, a complementary error message, which says that the supply voltage is too low if no target is connected to J-Link / J-Trace, and the speed selection. The screenshot below shows an example.

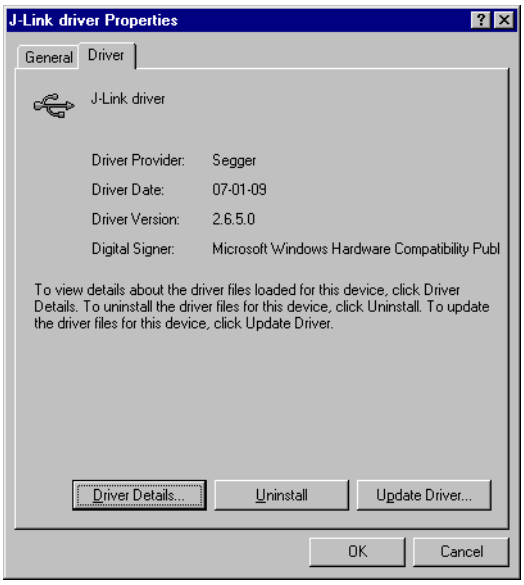


In addition you can verify the driver installation by consulting the Windows device manager. If the driver is installed and your J-Link / J-Trace is connected to your computer, the device manager should list the J-Link USB driver as a node below "Universal Serial Bus controllers" as shown in the following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



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

If you experience problems, refer to the chapter *Support and FAQs* on page 107 for help. You can select the **Driver** tab for detailed information about driver provider, version, date and digital signer.



Uninstalling the J-Link USB driver

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

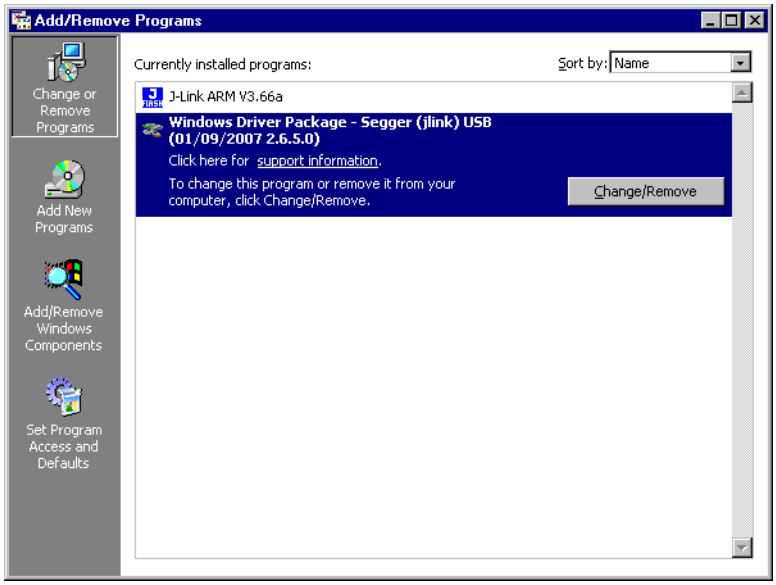
This might be the case when:

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

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

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

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



3 Confirm the uninstallation process.

J-Link and J-Trace related software

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

J-Link related software

J-LINK SOFTWARE AND DOCUMENTATION PACKAGE

IAR Embedded Workbench is bundled with applications for J-Link. Some of the applications require an additional license.

Software	Description
JLinkARM.dll	DLL for using J-Link / J-Trace.
JLink.exe	Free command-line tool with basic functionality for target analysis.
JLinkSTR91x	Free command-line tool to configure the ST STR91x cores. For more information please refer to *J-Link STR91x Commander (Command line tool)* on page 22
JLinkSTM32	Free command-line tool for STM32 devices. Can be used to disable the hardware watchdog and to unsecure STM32 devices (override read-protection).
J-Link TCP/IP Server	Free utility which provides the possibility to use J-Link / J-Trace remotely via TCP/IP.
J-Mem memory viewer	Free target memory viewer. Shows the memory content of a running target and allows editing as well.
J-Flash	Stand-alone flash programming application. This requires an additional license.
Flash breakpoints	Flash breakpoints provide the ability to set an unlimited number of software breakpoints in flash memory areas. This requires an additional license.

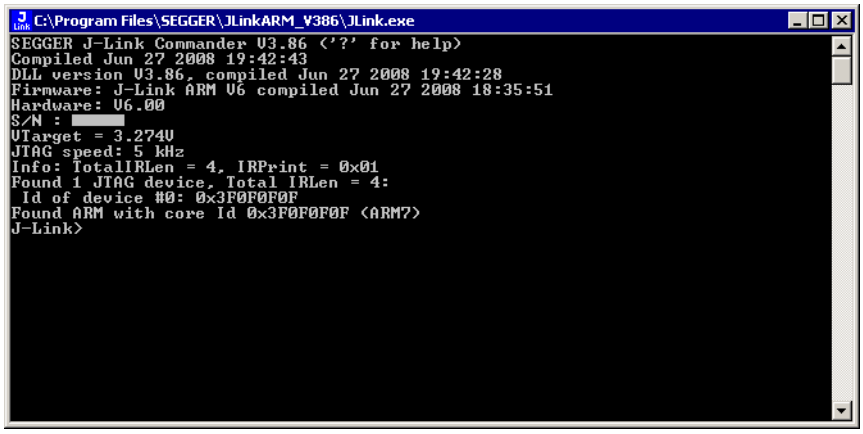
Table 4: J-Link / J-Trace related software

J-Link software and documentation package in detail

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

J-LINK COMMANDER (COMMAND LINE TOOL)

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

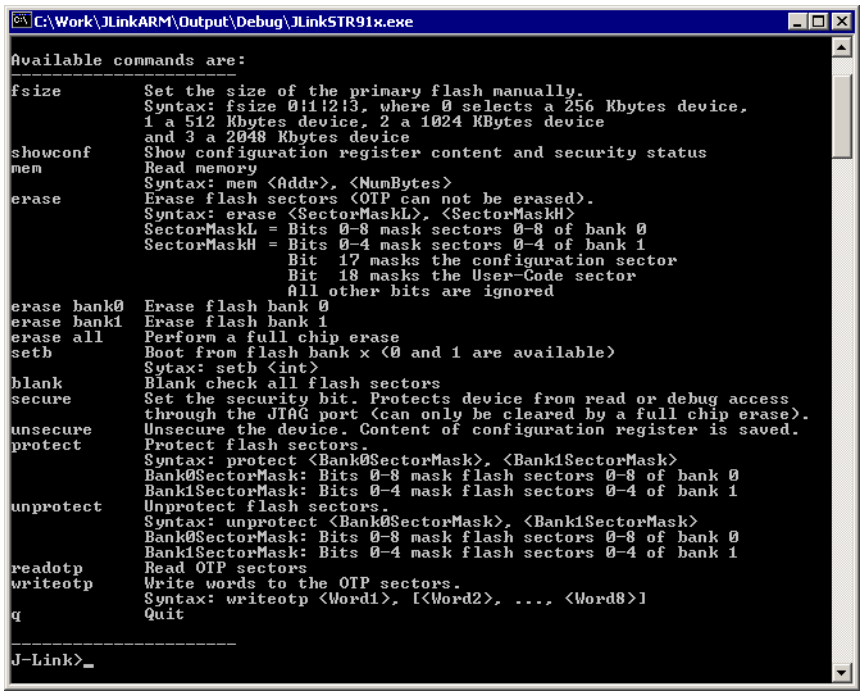


J-LINK STR91X COMMANDER (COMMAND LINE TOOL)

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

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

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



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

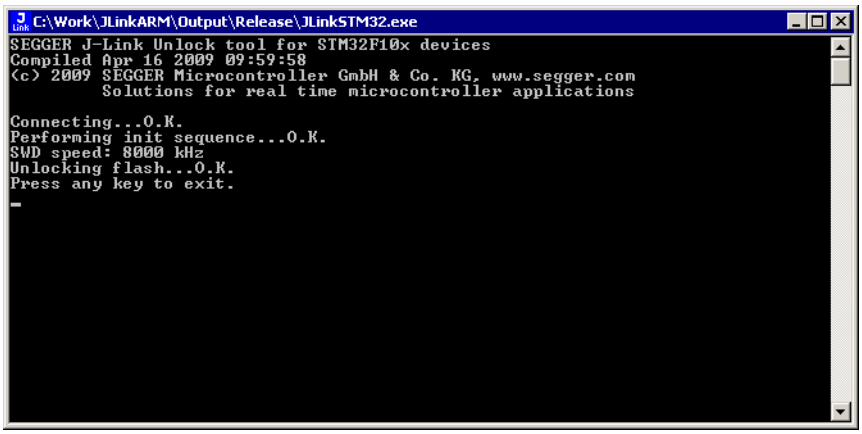
"While enabling the Turbo Mode, a dedicated test mode signal is set and controls the GPIOs in output. The IOs are maintained in this state until a next JTAG instruction is send." (ST Microelectronics)

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

J-LINK STM32 COMMANDER (COMMAND LINE TOOL)

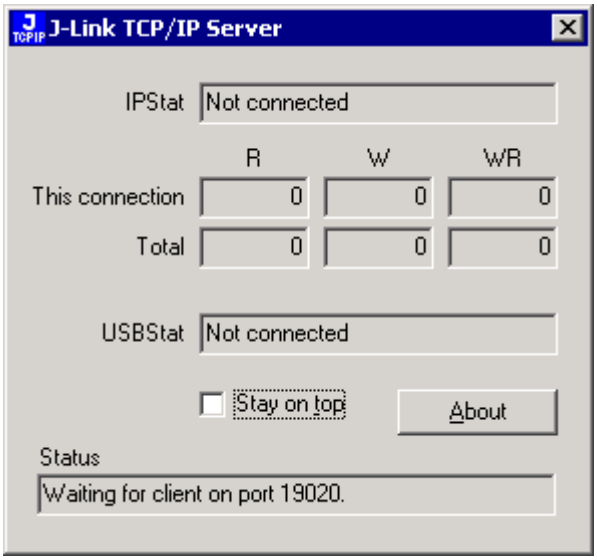
J-Link STM32 Commander (`JLinkSTM32.exe`) is a free command line tool which can be used to disable the hardware watchdog of STM32 devices which can be activated by programming the option bytes. Moreover the J-Link STM32 Commander unsecures a read-protected STM32 device by re-programming the option bytes.

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



J-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.



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

List of available commands

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

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

Table 5: Available commands

port

Syntax

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

Example

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

```
-port 19021
```

usb

Syntax

```
-usb <USBIndex>
```

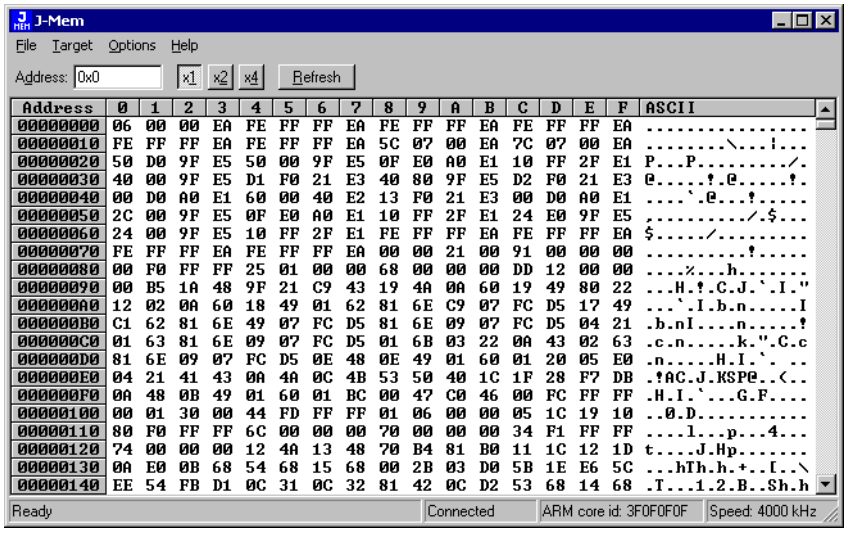
Example

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

```
-usb 2
```

J-MEM MEMORY VIEWER

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

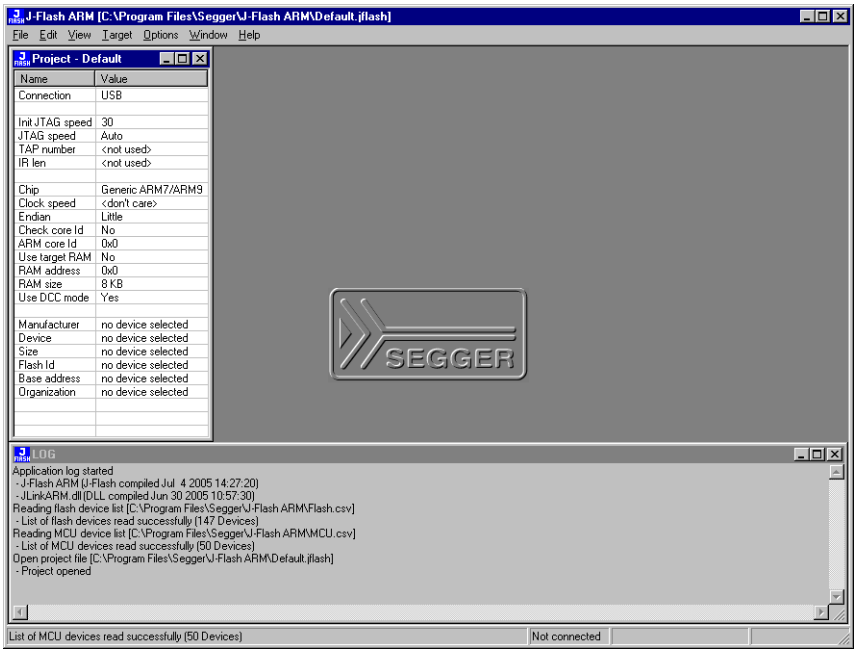


J-FLASH ARM (PROGRAM FLASH MEMORY VIA JTAG)

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

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

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



Features

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

Using the J-LinkARM.dll

WHAT IS THE JLINKARM.DLL?

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

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

UPDATING THE DLL

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

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

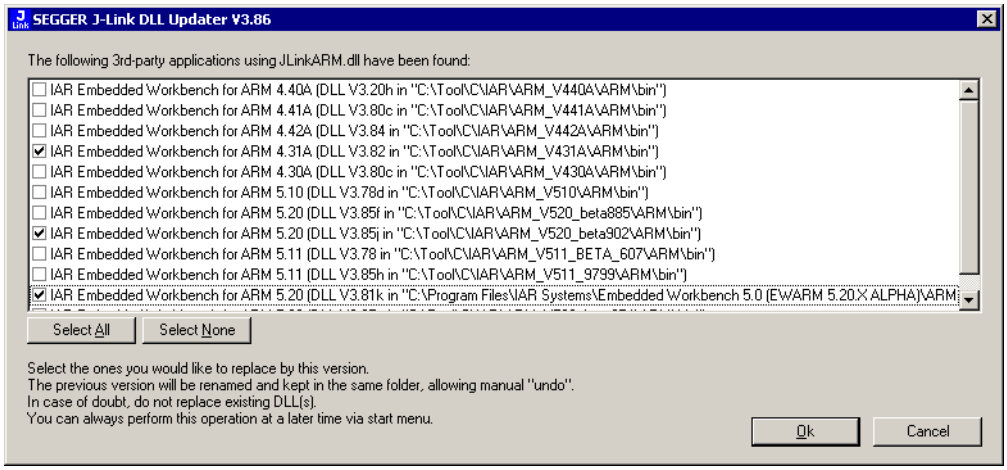
It's recommended to use the J-Link DLL updater to update the `JLinkARM.dll` in the IAR Embedded Workbench. The IAR Embedded Workbench IDE is a high-performance integrated development environment with an editor, compiler, linker, debugger. The compiler generates very efficient code and is widely used.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

Typically, the DLL is located in `C:\Program Files\IAR Systems\Embedded Workbench 5.0\arm\bin\`.

After updating the DLL, it is recommended to verify that the new DLL is loaded as described in *Determining which DLL is used by a program* on page 27.

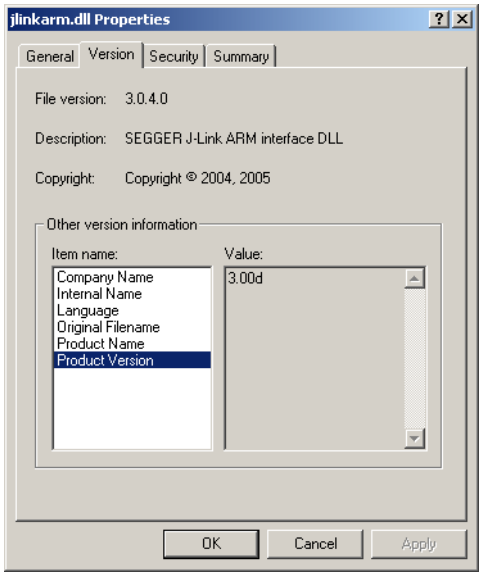
J-Link DLL updater

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



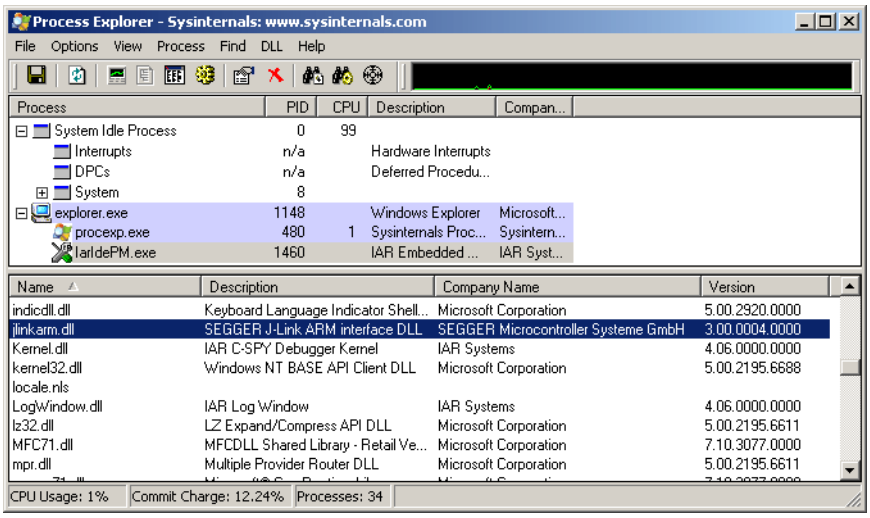
DETERMINING THE VERSION OF JLINKARM.DLL

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



DETERMINING WHICH DLL IS USED BY A PROGRAM

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



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

Working with J-Link and J-Trace

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

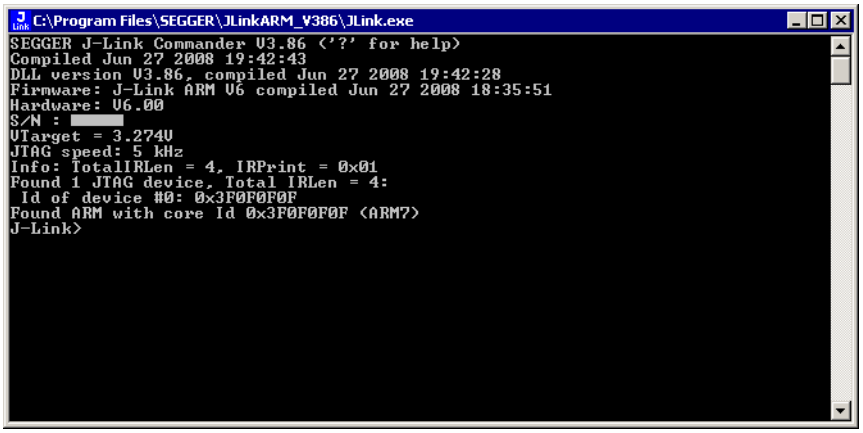
Connecting the target system

POWER-ON SEQUENCE

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

VERIFYING TARGET DEVICE CONNECTION

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



PROBLEMS

If you experience problems with any of the steps described above, read the chapter *Support and FAQs* on page 107 for troubleshooting tips. If you still do not find appropriate help there and your J-Link / J-Trace is an original IAR System product, you can contact support via e-mail. Provide the necessary information about your target processor, board etc. and we will try to solve your problem. A checklist of the required information together with the contact information can be found in chapter *Support and FAQs* on page 107 as well.

Indicators

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

MAIN INDICATOR

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

Single color indicator (J-Link V7 and earlier)

Indicator status	Meaning
GREEN, flashing at 10 Hz	Emulator enumerates.
GREEN, flickering	Emulator is in operation. Whenever the emulator is executing a command, the LED is switched off temporarily. Flickering speed depends on target interface speed. At low interface speeds, operations typically take longer and the "OFF" periods are typically longer than at fast speeds.
GREEN, constant	Emulator has enumerated and is in Idle mode.
GREEN, flashing at 1 Hz	Emulator has a fatal error. This should not normally happen.

Table 6: J-Link single color main indicator

Bi-color indicator (J-Link V8)

Indicator status	Meaning
GREEN, flashing at 10 Hz	Emulator enumerates.
GREEN, flickering	Emulator is in operation. Whenever the emulator is executing a command, the LED is switched off temporarily. Flickering speed depends on target interface speed. At low interface speeds, operations typically take longer and the "OFF" periods are typically longer than at fast speeds.
GREEN, constant	Emulator has enumerated and is in Idle mode.
ORANGE	Reset is active on target.
RED, flashing at 1 Hz	Emulator has a fatal error. This should not normally happen.

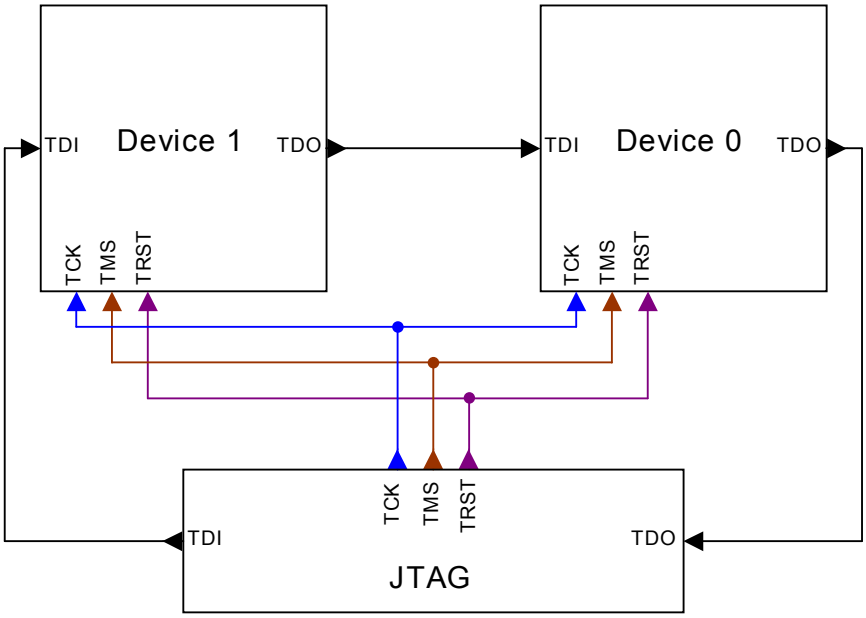
Table 7: J-Link single color LED main color indicator

JTAG interface

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

MULTIPLE DEVICES IN THE SCAN CHAIN

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



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

Configuration

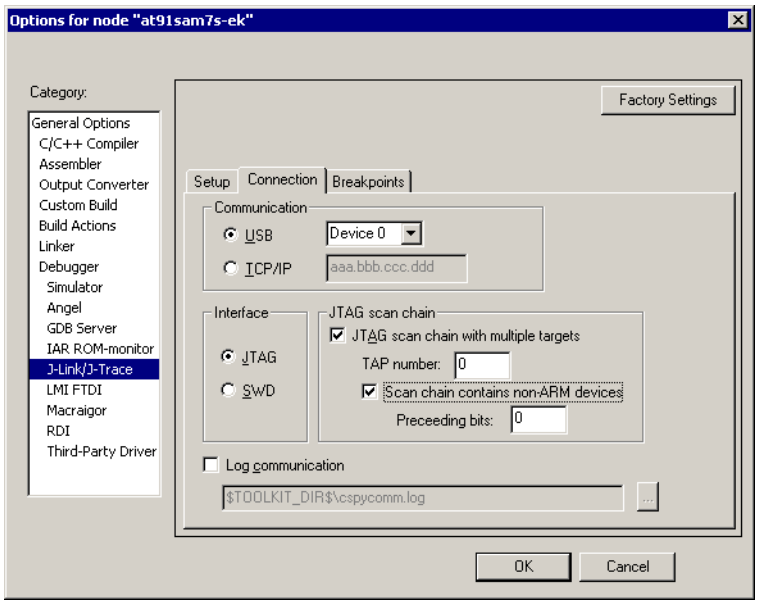
The configuration of the scan chain depends on the application used. Read *JTAG interface* on page 30 for further instructions and configuration examples.

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.

IAR J-Link configuration dialog box

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



DETERMINING VALUES FOR SCAN CHAIN CONFIGURATION

When do I need to configure the scan chain?

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

How do I configure the scan chain?

2 values need to be known:

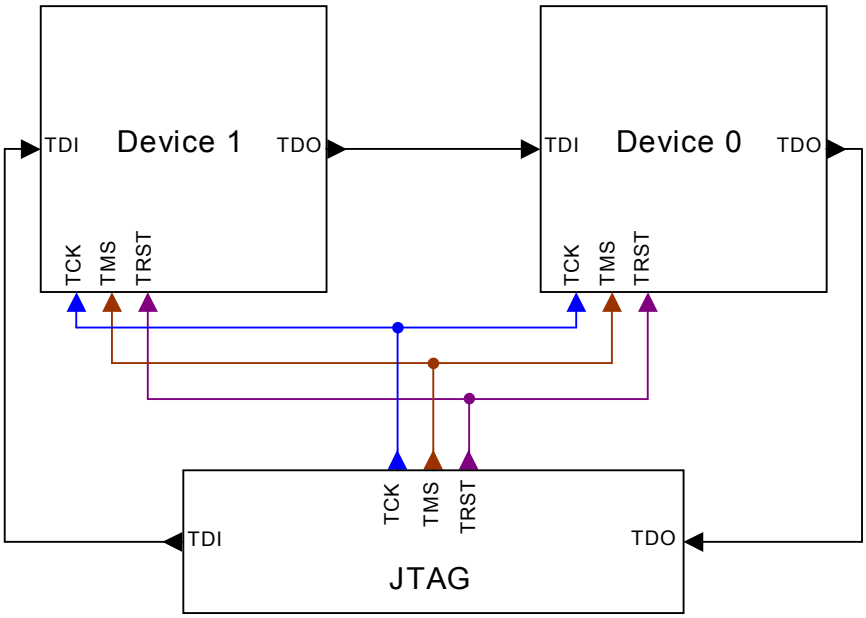
- The position of the target device in the scan chain
- The total number of bits in the instruction registers of the devices before the target device (IR len).

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

ARM7/ARM9 have an IR len of four.

Sample configurations

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



Examples

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

Device 0 Chip(IR len)	Device 1 Chip(IR len)	Device 2 Chip(IR len)	Position	IR len
ARM(4)	-	-	0	0
ARM(4)	Xilinx(8)	-	0	0
Xilinx(8)	ARM(4)	-	1	8
Xilinx(8)	Xilinx(8)	ARM(4)	2	16
ARM(4)	Xilinx(8)	ARM(4)	0	0
ARM(4)	Xilinx(8)	ARM(4)	2	12
Xilinx(8)	ARM(4)	Xilinx(8)	1	8

Table 8: Example scan chain configurations

The target device is marked in blue.

JTAG SPEED

There are basically three types of speed settings:

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

These are explained below.

Fixed JTAG speed

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

JTAG speeds of more than 10 MHz are not recommended.

Automatic JTAG speed

Selects the maximum JTAG speed handled by the TAP controller.

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

Adaptive clocking

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

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

SWD interface

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

SWO

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

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

Max. SWO speeds

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

from the emulator. Currently, the following are supported:

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

Table 9: J-Link supported SWO input speeds

Configuring SWO speeds

The max. SWO speed in practice is the max. speed which both, target and J-Link can handle. J-Link can handle the frequencies described in *SWO* on page 34 whereas the max. deviation between the target and the J-Link speed is about 3%.

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

Example1

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

Possible SWO output speeds are:

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

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

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

Permitted combinations are:

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

Table 10: Permitted SWO speed combinations

Example 2

Target CPU running at 10 MHz.

Possible SWO output speeds are:

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

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

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

Permitted combinations are:

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

Table 11: Permitted SWO speed combinations

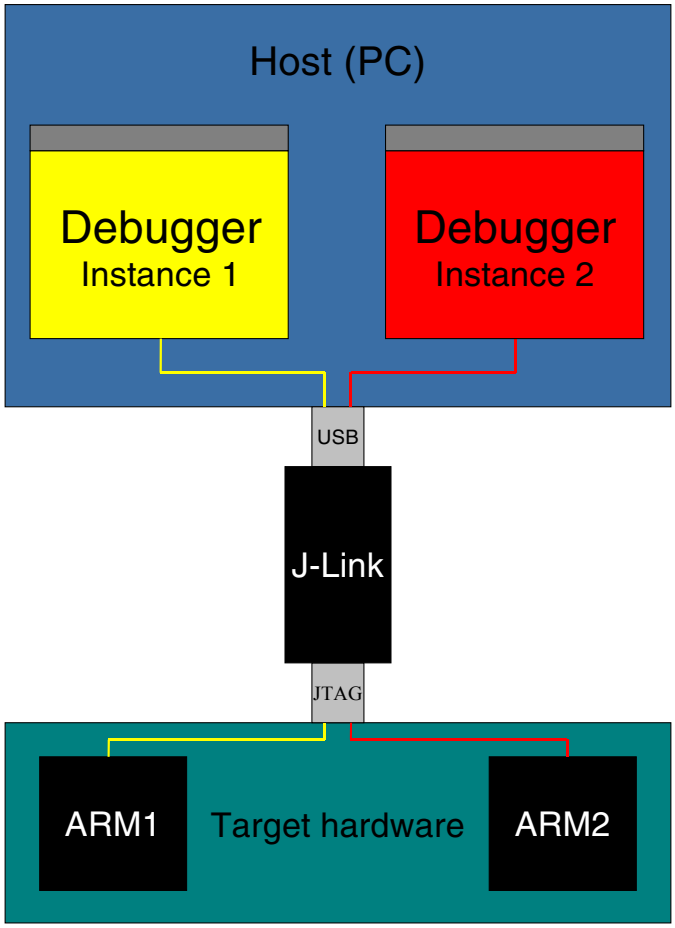
Multi-core debugging

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

HOW MULTI-CORE DEBUGGING WORKS

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

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



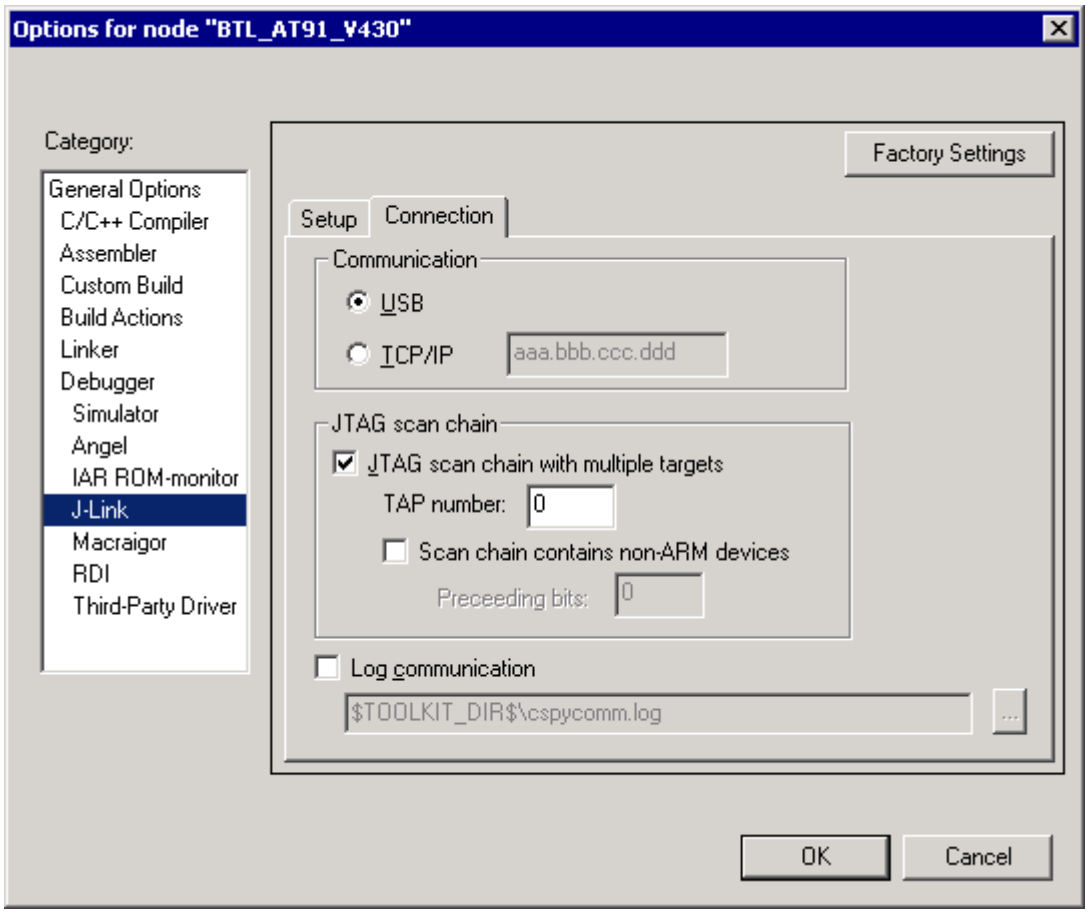
Both debuggers share the same physical connection. The core to debug is selected through the JTAG-settings as described below.

USING MULTI-CORE DEBUGGING IN DETAIL

1 Connect your target to J-Link / J-Trace.

2 Start your debugger in the IAR Embedded Workbench IDE for ARM.

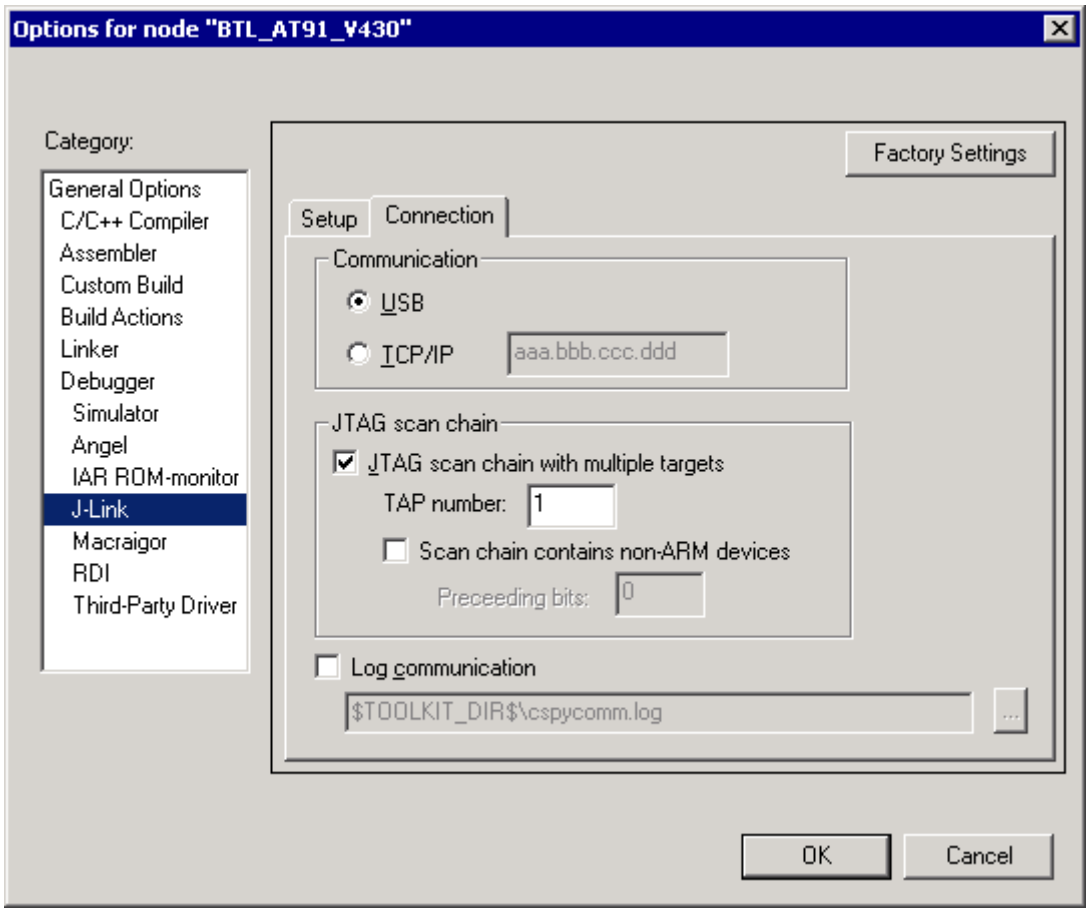
3 Choose **Project | Options** and configure your scan chain. The picture below shows the configuration for the first ARM core in the scan chain.



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

Table 12: Multicore debugging

Cores to debug are marked in blue.

THINGS YOU SHOULD BE AWARE OF

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

JTAG speed

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

For example:

Core #1: 2MHz maximum JTAG speed

Core #2: 4MHz maximum JTAG speed

Scan chain: 2MHz maximum JTAG speed

Resetting the target

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

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

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.

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

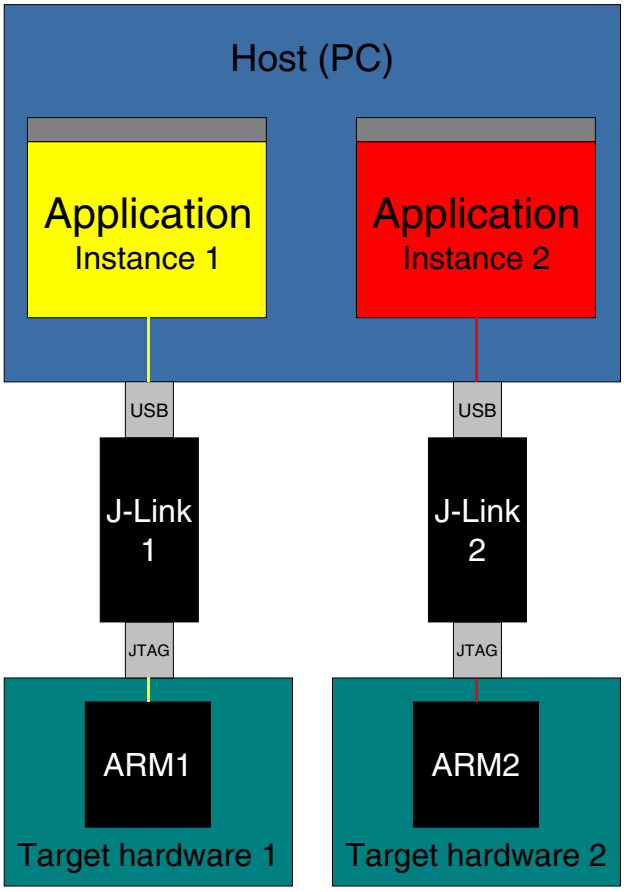
HOW DOES IT WORK?

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) 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.

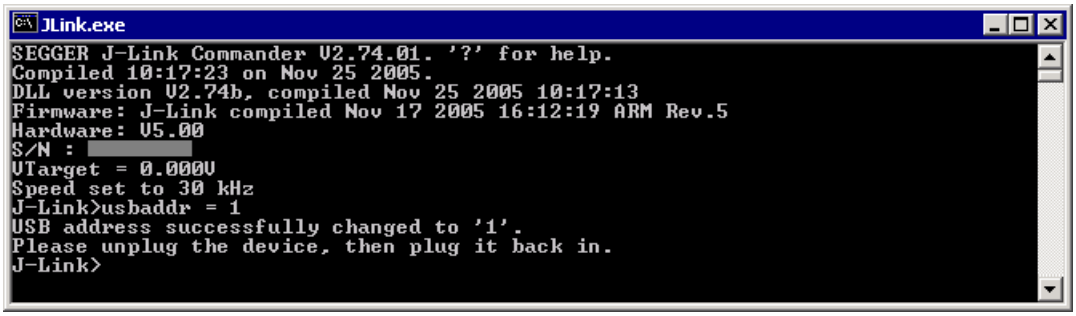
A different PID means that J-Link / J-Trace is identified as a different device, requiring a new driver. The driver for a new J-Link device will be installed automatically.

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



8 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.

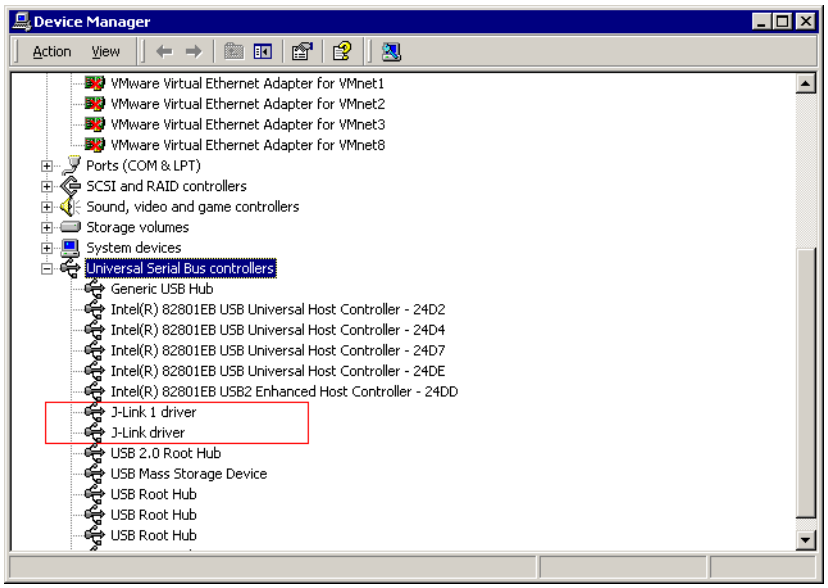
9 Type `usbaddr = 1` to set the J-Link / J-Trace #1.



10 Unplug J-Link / J-Trace and then plug it back in.

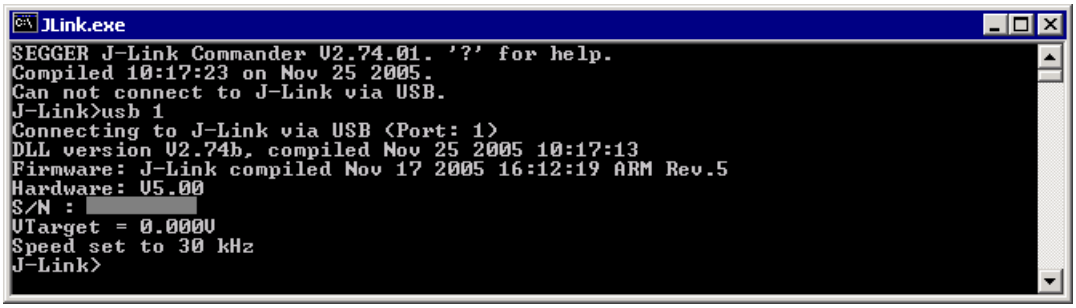
11 The system will recognize and automatically install a new J-Link / J-Trace.

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



CONNECTING TO A J-LINK / J-TRACE WITH NON DEFAULT USB-ADDRESS

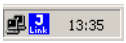
Restart `JLink.exe` and type `usb 1` to connect to J-Link / J-Trace #1.



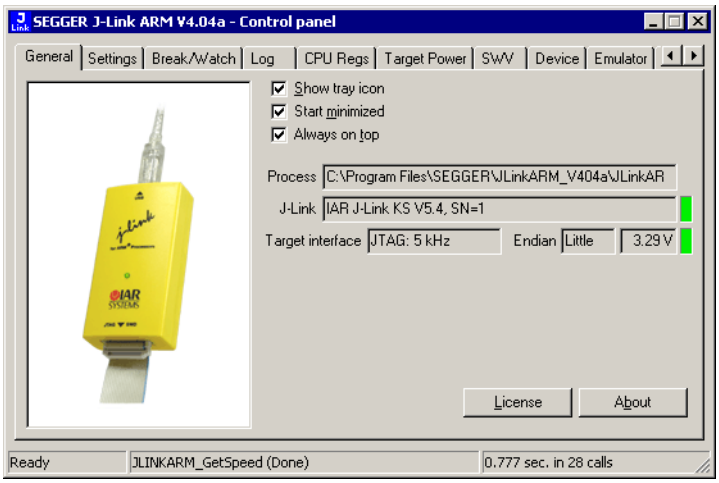
You may connect other J-Links / J-Traces to your PC and connect to them as well. To connect to an unconfigured J-Link / J-Trace (with default address "0"), restart `JLink.exe` or type `usb 0`.

J-Link control panel

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



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



TABS

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

General

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

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

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

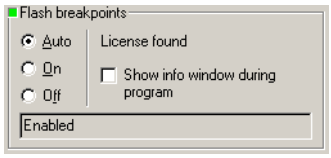
- **Process**: Shows the path of the file which loaded the DLL.
- **J-Link**: Shows the name of the connected J-Link, the hardware version and the Serial number. If no J-Link is connected it shows "not connected" and the color indicator is red.
- **Target interface**: Shows the selected target interface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

Settings

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

Section: Flash breakpoints:

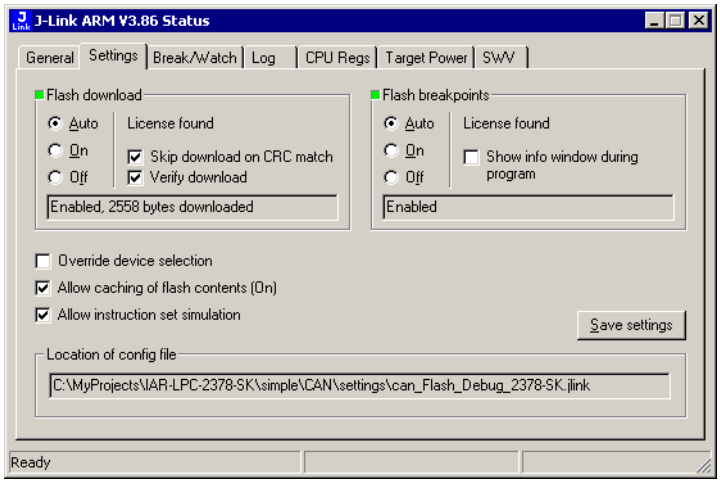
In this section, settings for the use of the `FlashBP` feature and related settings can be configured. When a license for `FlashBP` is found, the color indicator is green and "License found" appears right to the `FlashBP` usage settings.



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

FlashBP independent settings

These settings do not belong to the `FlashBP` settings section. They can be configured without any license needed.

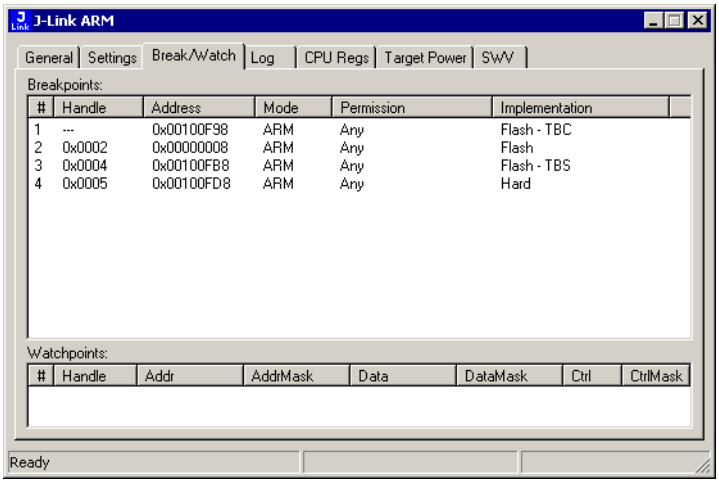


- **Override device selection**: If this checkbox is enabled, a dropdown list appears, which allows the user to set a device manually. This especially makes sense when J-Link can not identify the device name given by the debugger or if a particular device is not yet known to the debugger, but to the J-Link software.
- **Allow caching of flash contents**: If this checkbox is enabled, the flash contents are cached by J-Link to avoid reading data twice. This speeds up the transfer between debugger and target.

- **Allow instruction set simulation**: If this checkbox is enabled, ARM instructions will be simulated as far as possible. This speeds up single stepping, especially when `FlashBPs` are used.
- **Save settings**: When this button is pushed, the current settings in the **Settings** tab will be saved in a configuration file. This file is created by J-Link and will be created for each project and each project configuration (e.g. Debug_RAM, Debug_Flash).
- **Location of config file**: Shows the path where the configuration file is placed. This configuration file contains all the settings which can be configured in the **Settings** tab.

Break/Watch

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



Section: Breakpoints

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

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

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

Section: Watchpoints

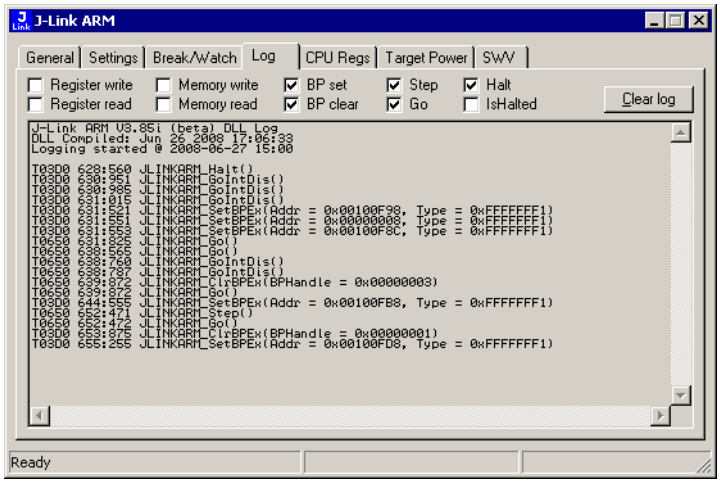
In this section, all watchpoints which are listed in the DLL internal breakpoint list are shown.

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

Log

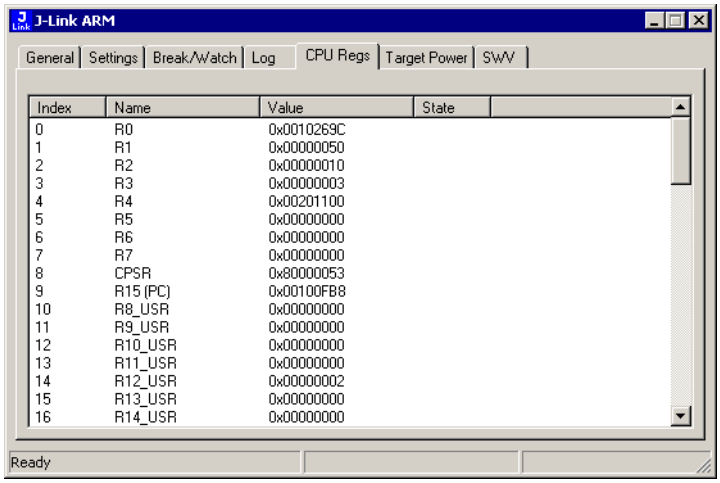
In this section the log output of the DLL is shown. The user can determine which function calls should be shown in the log window.

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



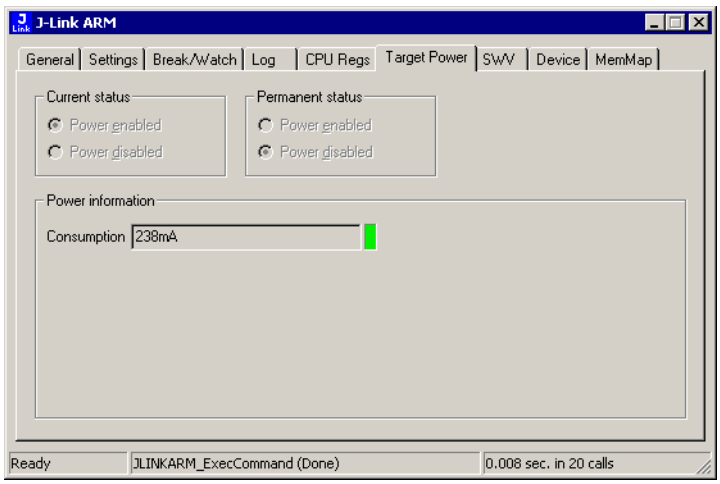
CPU Regs

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



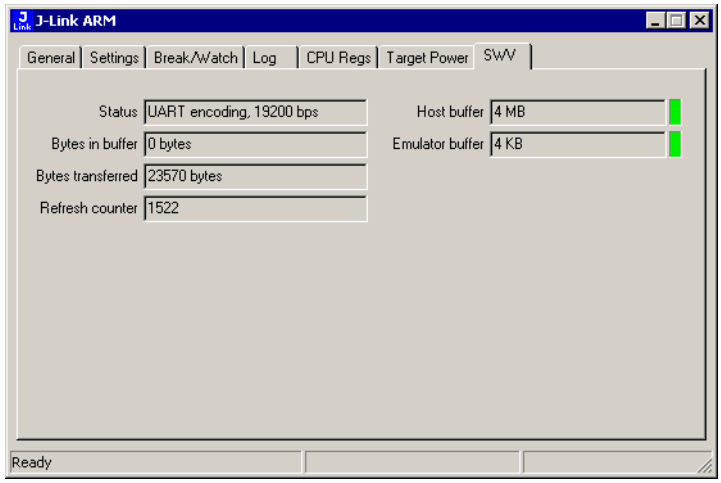
Target Power

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



SWV

In this section SWV information are shown.



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

Reset strategies

J-Link / J-Trace supports different reset strategies. This is necessary because there is no single way of resetting and halting an ARM core before it starts to execute instructions.

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.

RESET STRATEGIES IN DETAIL

Type 0: Hardware, halt after reset (normal)

The hardware reset pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU. This typically halts the CPU shortly after reset release; the CPU can in most systems execute some instructions before it is halted. The number of instructions executed depends primarily on the JTAG speed: the higher the JTAG speed, the faster the CPU can be halted.

Some CPUs can actually be halted before executing any instruction, because the start of the CPU is delayed after reset release. If a pause has been specified, J-Link waits for the specified time before trying to halt the CPU. This can be useful if a bootloader which resides in flash or ROM needs to be started after reset.

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

Type 1: Hardware, halt with BP@0

The hardware reset pin is used to reset the CPU. Before doing so, the ICE breaker is programmed to halt program execution at address 0; effectively, a breakpoint is set at address 0. If this strategy works, the CPU is actually halted before executing a single instruction.

This reset strategy does not work on all systems for two reasons:

- If nRESET and nTRST are coupled, either on the board or the CPU itself, reset clears the breakpoint, which means the CPU is not stopped after reset.
- Some MCUs contain a bootloader program (sometimes called kernel), which needs to be executed to enable JTAG access.

Type 2: Software, for Analog Devices ADuC7xxx MCUs

This reset strategy is a software strategy. The CPU is halted and performs a sequence which causes a peripheral reset. The following sequence is executed:

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

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

Type 3: No reset

No reset is performed. Nothing happens.

Type 4: Hardware, halt with WP

The hardware RESET pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU using a watchpoint. This typically halts the CPU shortly after reset release; the CPU can in most systems execute some instructions before it is halted.

The number of instructions executed depends primarily on the JTAG speed: the higher the JTAG speed, the faster the CPU can be halted. Some CPUs can actually be halted before executing any instruction, because the start of the CPU is delayed after reset release

Type 5: Hardware, halt with DBGRQ

The hardware RESET pin is used to reset the CPU. After reset release, J-Link continuously tries to halt the CPU using the DBGRQ. This typically halts the CPU shortly after reset release; the CPU can in most systems execute some instructions before it is halted.

The number of instructions executed depends primarily on the JTAG speed: the higher the JTAG speed, the faster the CPU can be halted. Some CPUs can actually be halted before executing any instruction, because the start of the CPU is delayed after reset release.

Type 6: Software

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

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

Type 7: Reserved

Reserved reset type.

Type 8: Software, for ATMEL AT91SAM7 MCUs

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

It is recommended to use this reset strategy. This special reset strategy resets the peripherals by writing to the RSTC_CR register. Resetting the peripherals puts all peripherals in the defined reset state. This includes memory mapping register, which means that after reset flash is mapped to address 0. It is also possible to achieve the same effect by writing 0x4 to the RSTC_CR register located at address 0xfffffd00.

Type 9: Hardware, for NXP LPC MCUs

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

CORTEX-M3 SPECIFIC RESET STRATEGIES

J-Link supports different specific reset strategies for the Cortex-M3 core. All of the following reset strategies are available in JTAG and in SWD mode. All three reset strategies halt the CPU after the reset.

Type 0: Normal

This reset strategy is the default strategy and should usually be used to reset the target. When using this strategy, J-Link sets the VC_CORERESET bit before reset, which causes the CPU to halt before execution of the first instruction. In addition to that the SYSRESETREQ bit and the VECTRESET bit are also set in order to guarantee that the core and the peripherals are reset even on targets where the CPU RESET pin is not connected to Pin 15 (RESET) of the JTAG/SWD connector.

Type 1: Core

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

Type 2: ResetPin

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

Using DCC for memory access

The ARM7/9 architecture requires cooperation of the CPU to access memory when the CPU is running (not in debug mode). This means that memory can not normally be accessed while the CPU is executing the application program. The normal way to read or write memory is to halt the CPU (put it into debug mode) before accessing memory. Even if the CPU is restarted after the memory access, the real time behavior is significantly affected; halting and restarting the CPU costs typically multiple milliseconds. For this reason, most debuggers do not even allow memory access if the CPU is running.
Fortunately, there is one other option: DCC (Direct communication channel) can be used to communicate with the CPU while it is executing the application program. All that is required is that the application program calls a DCC handler from time to time. This DCC handler typically requires less than 1 µs per call.

The DCC handler, as well as the optional DCC abort handler, is part of the J-Link software and can be found in the `Program Files\IAR Systems\Embedded Workbench 5.0\arm\src\debugger\dcc\` directory of the package.

WHAT IS REQUIRED?

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

- A target application program that regularly calls the DCC handler
- The supplied abort handler should be installed (optional)

Another application program that uses DCC is `JLink.exe`.

TARGET DCC HANDLER

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

TARGET DCC ABORT HANDLER

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

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

Command strings

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

LIST OF AVAILABLE COMMANDS

The table below lists and describes the available command strings.

Command	Description
device	Selects the target device.
DisableFlashBPs	Disables the `FlashPB` feature.
DisableFlashDL	Disables the `J-Link ARM FlashDL` feature.
EnableFlashBPs	Enables the `FlashPB` feature.
EnableFlashDL	Enables the `J-Link ARM FlashDL` feature.
map exclude	Ignore all memory accesses to specified area.
map indirectread	Specifies an area which should be read indirect.
map ram	Specifies location of target RAM.
map reset	Restores the default mapping, which means all memory accesses are permitted.
SetAllowSimulation	Enable/Disable instruction set simulation.
SetCheckModeAfterRead	Enable/Disable CPSR check after read operations.
SetResetPulseLen	Defines the length of the RESET pulse in milliseconds.
SetResetType	Selects the reset strategy
SetRestartOnClose	Specifies restart behavior on close.

Table 13: Available command line options

Command	Description
SetDbgPowerDownOnClose	Used to power-down the debug unit of the target CPU when the debug session is closed.
SetSysPowerDownOnIdle	Used to power-down the target CPU, when there are no transmissions between J-Link and target CPU, for a specified timeframe.
SupplyPower	Activates/Deactivates power supply over pin 19 of the JTAG connector.
SupplyPowerDefault	Activates/Deactivates power supply over pin 19 of the JTAG connector permanently.

Table 13: Available command line options

device

This command selects the target device.

Syntax

```
device = <DeviceID>
```

DeviceID has to be a valid device identifier. For a list of all available device identifiers please refer to chapter *Supported devices* on page 58.

Example

```
device = AT91SAM7S256
```

DisableFlashBPs

This command disables the FlashBP feature.

Syntax

```
DisableFlashBPs
```

DisableFlashDL

This command disables the J-Link ARM FlashDL feature.

Syntax

```
DisableFlashDL
```

EnableFlashBPs

This command enables the FlashBP feature.

Syntax

```
EnableFlashBPs
```

EnableFlashDL

This command enables the J-Link ARM FlashDL feature.

Syntax

```
EnableFlashDL
```

map exclude

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

Memory mapping

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

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

Syntax

```
map exclude <SAddr>-<EAddr>
```

Example

This is an example for the `map exclude` command in combination with an NXP LPC2148 MCU.

Memory map

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

The "problematic" memory areas are:

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

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

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

map indirectread

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

Typical applications

Refer to chapter *Fast GPIO bug* on page 71 for an example.

Syntax

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

Example

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

map ram

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

Typical applications

Refer to chapter *Fast GPIO bug* on page 71 for an example.

Syntax

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

Example

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

map reset

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

Typical applications

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

Syntax

```
map reset
```

Example

```
map reset
```

SetAllowSimulation

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

Syntax

```
SetAllowSimulation = 0 | 1
```

Example

```
SetAllowSimulation 1  // Enables instruction set simulation
```

SetCheckModeAfterRead

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

Typical applications

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

Syntax

```
SetCheckModeAfterRead = 0 | 1
```

Example

```
SetCheckModeAfterRead = 0
```

SetResetPulseLen

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

Syntax

```
SetResetPulseLen = <value>
```

Example

```
SetResetPulseLen = 50
```

SetResetType

This command changes the reset strategy.

Typical applications

Refer to chapter *Reset strategies* on page 45 for additional informations about the different reset strategies.

Value	Description
0	Hardware, halt after reset (normal).
1	Hardware, halt with BP@0.
2	Software, for Analog Devices ADuC7xxx MCUs.

Table 14: List of possible value for command SetResetType

Syntax

```
SetResetType = <value>
```

Example

```
SetResetType = 0
```

SetRestartOnClose

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

Syntax

```
SetRestartOnClose = 0 | 1
```

Example

```
SetRestartOnClose = 1
```

SetDbgPowerDownOnClose

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

Note: This command works only for Cortex-M3 devices

Typical applications

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

Syntax

SetDbgPowerDownOnClose = <value>

Example

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

SetSysPowerDownOnIdle

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

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

Typical applications

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

Syntax

```
SetSysPowerDownOnIdle = <value>
```

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

Example

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

SupplyPower

This command activates power supply over pin 19 of the JTAG connector. J-Link have the V5 supply over pin 19 activated by default.

Typical applications

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

Syntax

```
SupplyPower = 0 | 1
```

Example

```
SupplyPower = 1
```

SupplyPowerDefault

This command activates power supply over pin 19 of the JTAG connector permanently. J-Link have the V5 supply over pin 19 activated by default.

Typical applications

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

Syntax

```
SupplyPowerDefault = 0 | 1
```

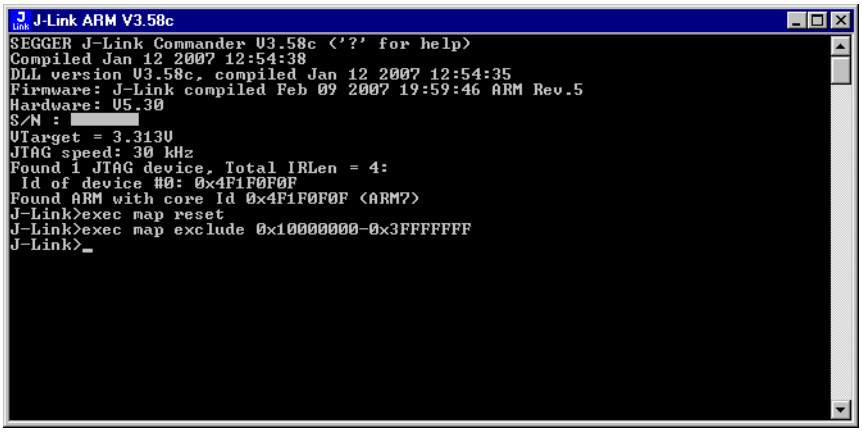
Example

```
SupplyPowerDefault = 1
```

USING COMMAND STRINGS

J-Link Commander

The J-Link command strings can be tested with the J-Link Commander. Use the comamand `exec` supplemented by one of the command strings.

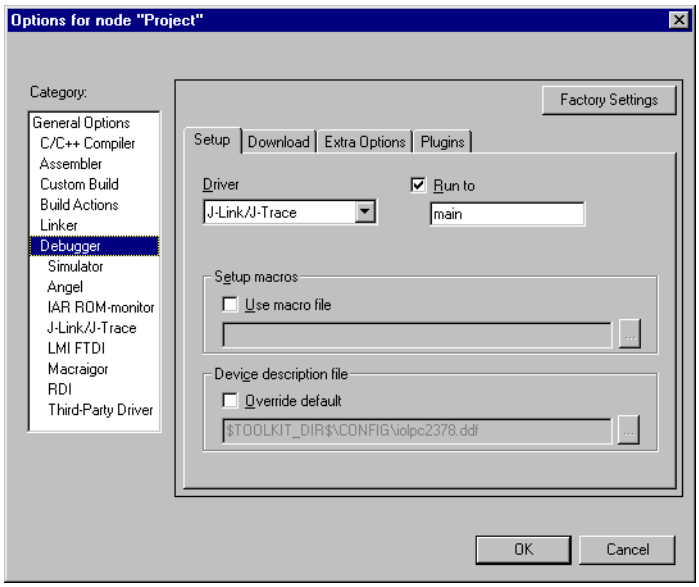


Example

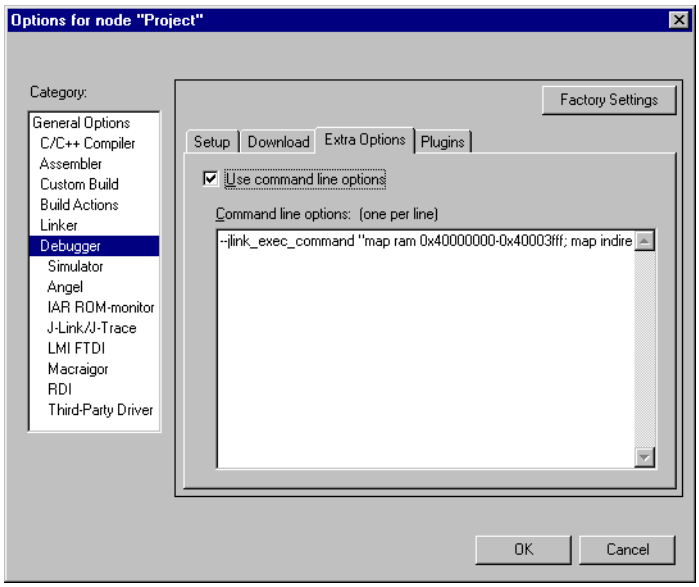
```
exec SupplyPower = 1
exec map reset
exec map exclude 0x10000000-0x3FFFFFFF
```

IAR Embedded Workbench

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



On the **Extra Options** page, select **Use command line options**.
Enter `--jlink_exec_command "<CommandLineOption>"` in the textfield. If more than one command should be used separate the commands with semicolon.



Switching off CPU clock during debug

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

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

With these cores, the TAP controller uses the clock signal provided by the emulator, which means the TAP controller and ICE-Breaker continue to be accessible even if the CPU has no clock.
Therefore, switching off CPU clock during debug is normally possible if the CPU clock is periodically (typically using a regular timer interrupt) switched on every few ms for at least a few us. In this case, the CPU will stop at the first instruction in the ISR (typically at address 0x18).

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

With these cores, the clock input of the TAP controller is connected to the output of a three-stage synchronizer, which is fed by clock signal provided by the emulator, which means that the TAP controller and ICE-Breaker are not accessible if the CPU has no clock.
If the RTCK signal is provided, the adaptive clocking function can be used to synchronize the JTAG clock (provided by the emulator) to the processor clock. This way, the JTAG clock is stopped if the CPU clock is switched off.

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

Cache handling

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

CACHE COHERENCY

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

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

CACHE CLEAN AREA

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

CACHE HANDLING OF ARM7 CORES

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

CACHE HANDLING OF ARM9 CORES

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

When entering debug state

J-Link / J-Trace performs the following:

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

When leaving debug state

J-Link / J-Trace performs the following:

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

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

Flash download and flash breakpoints

This chapter describes how flash breakpoints work. In addition, the chapter contains a list of supported microcontrollers.

Introduction

The `JLinkARM.dll` is able to use the flash breakpoints features, but it requires an additional license.

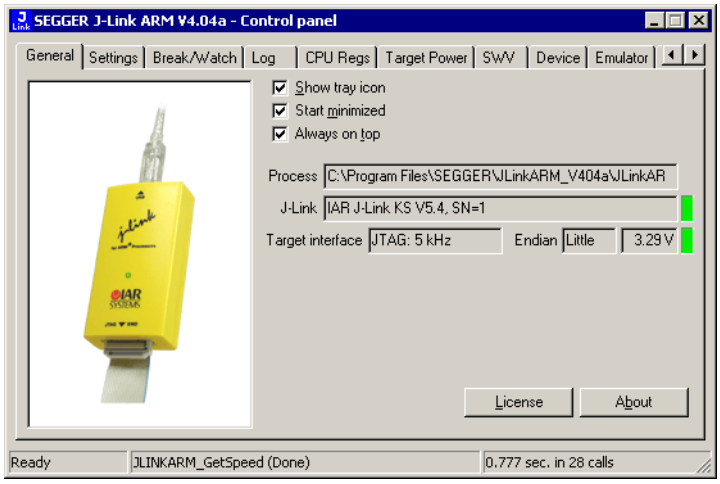
Licensing

The standard J-Link does not come with a built-in license. You will need to obtain a license for every J-Link. For more information about the different license types, please refer to *License types* on page 13.
For a complete list of devices which are supported by the device-based licenses, please refer to *Device list* on page 14.

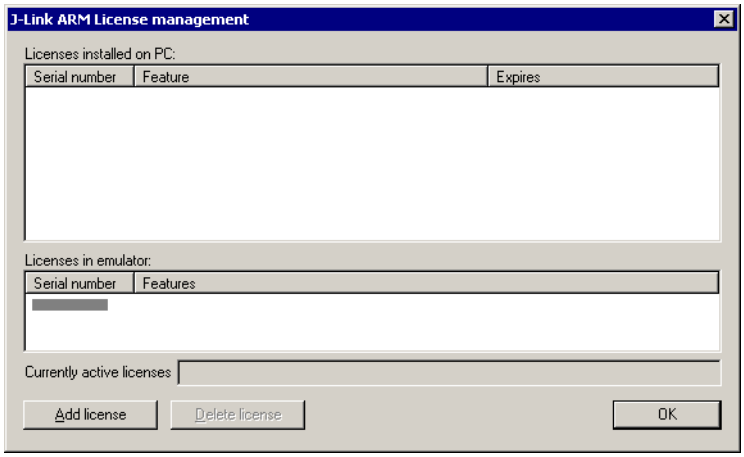
Entering a license

The easiest way to enter a license is the following:

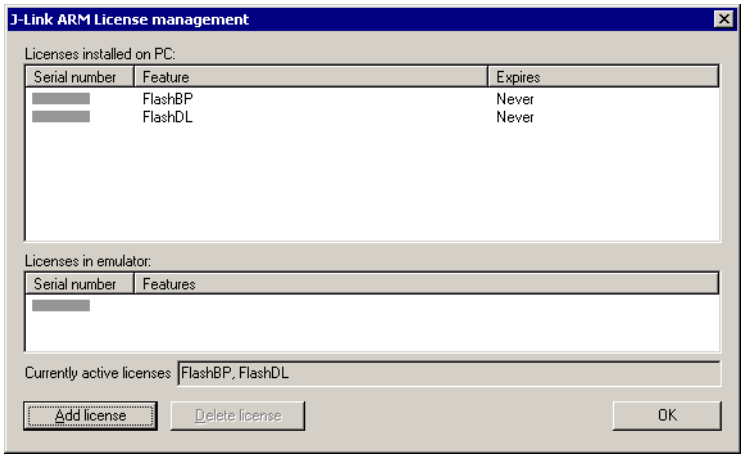
Open the J-Link control panel window, go to the **General** tab and choose **License**.



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



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



Supported devices

The following table lists the microcontrollers for which flash breakpoints are available.

Note: Only the devices listed below are currently supported with the flash breakpoint feature. This feature works on the internal flash of the devices only. You need to make sure that the device you are using is supported.

The device is selected by its device identifier.

Manufacturer	Device ID	Devices
Analog Devices	ADuC7020x62	ADuC7020x62
Analog Devices	ADuC7021x32	ADuC7021x32
Analog Devices	ADuC7021x62	ADuC7021x62
Analog Devices	ADuC7022x32	ADuC7022x32
Analog Devices	ADuC7022x62	ADuC7022x62
Analog Devices	ADuC7024x62	ADuC7024x62
Analog Devices	ADuC7025x32	ADuC7025x32
Analog Devices	ADuC7025x62	ADuC7025x62
Analog Devices	ADuC7026x62	ADuC7026x62
Analog Devices	ADuC7027x62	ADuC7027x62
Analog Devices	ADuC7028x62	ADuC7028x62
Analog Devices	ADuC7030	ADuC7030
Analog Devices	ADuC7031	ADuC7031
Analog Devices	ADuC7032	ADuC7032
Analog Devices	ADuC7033	ADuC7033
Analog Devices	ADuC7038	ADuC7038
Analog Devices	ADuC7060	ADuC7060
Analog Devices	ADuC7128	ADuC7128
Analog Devices	ADuC7129	ADuC7129
Analog Devices	ADuC7229x126	ADuC7229x126
Atmel	AT91FR40162	AT91FR40162
Atmel	AT91SAM7A3	AT91SAM7A3
Atmel	AT91SAM7S32	AT91SAM7S32
Atmel	AT91SAM7S321	AT91SAM7S321
Atmel	AT91SAM7S64	AT91SAM7S64
Atmel	AT91SAM7S128	AT91SAM7S128

Table 15: Supported microcontrollers

Manufacturer	Device ID	Devices
Atmel	AT91SAM7S256	AT91SAM7S256
Atmel	AT91SAM7S512	AT91SAM7S512
Atmel	AT91SAM7SE32	AT91SAM7SE32
Atmel	AT91SAM7SE256	AT91SAM7SE256
Atmel	AT91SAM7SE512	AT91SAM7SE512
Atmel	AT91SAM7X128	AT91SAM7X128
Atmel	AT91SAM7X256	AT91SAM7X256
Atmel	AT91SAM7X512	AT91SAM7X512
Atmel	AT91SAM7XC128	AT91SAM7XC128
Atmel	AT91SAM7XC256	AT91SAM7XC256
Atmel	AT91SAM7XC512	AT91SAM7XC512
Freescale*	MAC7101	MAC7101
Freescale*	MAC7106	MAC7106
Freescale*	MAC7111	MAC7111
Freescale*	MAC7112	MAC7112
Freescale*	MAC7116	MAC7116
Freescale*	MAC7121	MAC7121
Freescale*	MAC7122	MAC7122
Freescale*	MAC7126	MAC7126
Freescale*	MAC7131	MAC7131
Freescale*	MAC7136	MAC7136
Freescale*	MAC7141	MAC7141
Freescale*	MAC7142	MAC7142
Luminary	LM3S101	LM3S101
Luminary	LM3S102	LM3S102
Luminary	LM3S301	LM3S301
Luminary	LM3S310	LM3S310
Luminary	LM3S315	LM3S315
Luminary	LM3S316	LM3S316
Luminary	LM3S317	LM3S317
Luminary	LM3S328	LM3S328
Luminary	LM3S601	LM3S601
Luminary	LM3S610	LM3S610
Luminary	LM3S611	LM3S611
Luminary	LM3S612	LM3S612
Luminary	LM3S613	LM3S613
Luminary	LM3S615	LM3S615
Luminary	LM3S617	LM3S617
Luminary	LM3S618	LM3S618
Luminary	LM3S628	LM3S628
Luminary	LM3S801	LM3S801
Luminary	LM3S811	LM3S811
Luminary	LM3S812	LM3S812
Luminary	LM3S815	LM3S815
Luminary	LM3S817	LM3S817
Luminary	LM3S818	LM3S818
Luminary	LM3S828	LM3S828

Table 15: Supported microcontrollers (Continued)

Manufacturer	Device ID	Devices
Luminary	LM3S2110	LM3S2110
Luminary	LM3S2139	LM3S2139
Luminary	LM3S2410	LM3S2410
Luminary	LM3S2412	LM3S2412
Luminary	LM3S2432	LM3S2432
Luminary	LM3S2533	LM3S2533
Luminary	LM3S2620	LM3S2620
Luminary	LM3S2637	LM3S2637
Luminary	LM3S2651	LM3S2651
Luminary	LM3S2730	LM3S2730
Luminary	LM3S2739	LM3S2739
Luminary	LM3S2939	LM3S2939
Luminary	LM3S2948	LM3S2948
Luminary	LM3S2950	LM3S2950
Luminary	LM3S2965	LM3S2965
Luminary	LM3S6100	LM3S6100
Luminary	LM3S6110	LM3S6110
Luminary	LM3S6420	LM3S6420
Luminary	LM3S6422	LM3S6422
Luminary	LM3S6432	LM3S6432
Luminary	LM3S6610	LM3S6610
Luminary	LM3S6633	LM3S6633
Luminary	LM3S6637	LM3S6637
Luminary	LM3S6730	LM3S6730
Luminary	LM3S6918	LM3S6918
Luminary	LM3S6938	LM3S6938
Luminary	LM3S6952	LM3S6952
Luminary	LM3S6965	LM3S6965
NXP	LPC2101	LPC2101
NXP	LPC2102	LPC2102
NXP	LPC2103	LPC2103
NXP	LPC2104	LPC2104
NXP	LPC2105	LPC2105
NXP	LPC2106	LPC2106
NXP	LPC2109	LPC2109
NXP	LPC2114	LPC2114
NXP	LPC2119	LPC2119
NXP	LPC2124	LPC2124
NXP	LPC2129	LPC2129
NXP	LPC2131	LPC2131
NXP	LPC2132	LPC2132
NXP	LPC2134	LPC2134
NXP	LPC2136	LPC2136
NXP	LPC2138	LPC2138
NXP	LPC2141	LPC2141
NXP	LPC2142	LPC2142
NXP	LPC2144	LPC2144

Table 15: Supported microcontrollers (Continued)

Manufacturer	Device ID	Devices
NXP	LPC2146	LPC2146
NXP	LPC2148	LPC2148
NXP	LPC2194	LPC2194
NXP	LPC2212	LPC2212
NXP	LPC2214	LPC2214
NXP	LPC2292	LPC2292
NXP	LPC2294	LPC2294
NXP	LPC2364	LPC2364
NXP	LPC2366	LPC2366
NXP	LPC2368	LPC2368
NXP	LPC2378	LPC2378
NXP	LPC2387	LPC2387
NXP	LPC2388	LPC2388
NXP	LPC2468	LPC2468
NXP	LPC2478	LPC2478
OKI	ML67Q4002	ML67Q4002
OKI	ML67Q4003	ML67Q4003
OKI	ML67Q4050	ML67Q4050
OKI	ML67Q4051	ML67Q4051
OKI	ML67Q4060	ML67Q4060
OKI	ML67Q4061	ML67Q4061
Samsung*	S3F445HX	S3F445HX
ST	STM32F101C6	STM32F101C6
ST	STM32F101C8	STM32F101C8
ST	STM32F101CB	STM32F101CB
ST	STM32F101R6	STM32F101R6
ST	STM32F101R8	STM32F101R8
ST	STM32F101RB	STM32F101RB
ST	STM32F101RC	STM32F101RC
ST	STM32F101RD	STM32F101RD
ST	STM32F101RE	STM32F101RE
ST	STM32F101T6	STM32F101T6
ST	STM32F101T8	STM32F101T8
ST	STM32F101V8	STM32F101V8
ST	STM32F101VB	STM32F101VB
ST	STM32F101VC	STM32F101VC
ST	STM32F101VD	STM32F101VD
ST	STM32F101VE	STM32F101VE
ST	STM32F101ZC	STM32F101ZC
ST	STM32F101ZD	STM32F101ZD
ST	STM32F101ZE	STM32F101ZE
ST	STM32F102C6	STM32F102C6
ST	STM32F102C8	STM32F102C8
ST	STM32F102CB	STM32F102CB
ST	STM32F103C6	STM32F103C6
ST	STM32F103C8	STM32F103C8
ST	STM32F103R6	STM32F103R6

Table 15: Supported microcontrollers (Continued)

Manufacturer	Device ID	Devices
ST	STM32F103R8	STM32F103R8
ST	STM32F103RB	STM32F103RB
ST	STM32F103RC	STM32F103RC
ST	STM32F103RD	STM32F103RD
ST	STM32F103RE	STM32F103RE
ST	STM32F103T6	STM32F103T6
ST	STM32F103T8	STM32F103T8
ST	STM32F103V8	STM32F103V8
ST	STM32F103VB	STM32F103VB
ST	STM32F103VC	STM32F103VC
ST	STM32F103VD	STM32F103VD
ST	STM32F103VE	STM32F103VE
ST	STM32F103ZC	STM32F103ZC
ST	STM32F103ZD	STM32F103ZD
ST	STM32F103ZE	STM32F103ZE
ST	STR710FZ1	STR710FZ1
ST	STR710FZ2	STR710FZ2
ST	STR711FR0	STR711FR0
ST	STR711FR1	STR711FR1
ST	STR711FR2	STR711FR2
ST	STR712FR0	STR712FR0
ST	STR712FR1	STR712FR1
ST	STR712FR2	STR712FR2
ST	STR715FR0	STR715FR0
ST	STR730FZ1	STR730FZ1
ST	STR730FZ2	STR730FZ2
ST	STR731FV0	STR731FV0
ST	STR731FV1	STR731FV1
ST	STR731FV2	STR731FV2
ST	STR735FZ1	STR735FZ1
ST	STR735FZ2	STR735FZ2
ST	STR736FV0	STR736FV0
ST	STR736FV1	STR736FV1
ST	STR736FV2	STR736FV2
ST	STR750FV0	STR750FV0
ST	STR750FV1	STR750FV1
ST	STR750FV2	STR750FV2
ST	STR751FR0	STR751FR0
ST	STR751FR1	STR751FR1
ST	STR751FR2	STR751FR2
ST	STR752FR0	STR752FR0
ST	STR752FR1	STR752FR1
ST	STR752FR2	STR752FR2
ST	STR755FR0	STR755FR0
ST	STR755FR1	STR755FR1
ST	STR755FR2	STR755FR2
ST	STR755FV0	STR755FV0

Table 15: Supported microcontrollers (Continued)

Manufacturer	Device ID	Devices
ST	STR755FV1	STR755FV1
ST	STR755FV2	STR755FV2
ST	STR910FAM32	STR910FAM32
ST	STR910FAW32	STR910FAW32
ST	STR910FAZ32	STR910FAZ32
ST	STR911FAM42	STR911FAM42
ST	STR911FAM44	STR911FAM44
ST	STR911FAM46	STR911FAM46
ST	STR911FAM47	STR911FAM47
ST	STR911FAW42	STR911FAW42
ST	STR911FAW44	STR911FAW44
ST	STR911FAW46	STR911FAW46
ST	STR911FAW47	STR911FAW47
ST	STR911FM32	STR911FM32
ST	STR911FM42	STR911FM42
ST	STR911FM44	STR911FM44
ST	STR911FW32	STR911FW32
ST	STR911FW42	STR911FW42
ST	STR911FW44	STR911FW44
ST	STR912FAW32	STR912FAW32
ST	STR912FAW42	STR912FAW42
ST	STR912FAW44	STR912FAW44
ST	STR912FAW46	STR912FAW46
ST	STR912FAW47	STR912FAW47
ST	STR912FAZ42	STR912FAZ42
ST	STR912FAZ44	STR912FAZ44
ST	STR912FAZ46	STR912FAZ46
ST	STR912FAZ47	STR912FAZ47
ST	STR912FM32	STR912FM32
ST	STR912FM42	STR912FM42
ST	STR912FM44	STR912FM44
ST	STR912FW32	STR912FW32
ST	STR912FW42	STR912FW42
ST	STR912FW44	STR912FW44
TI	TMS470R1A64	TMS470R1A64
TI	TMS470R1A128	TMS470R1A128
TI	TMS470R1A256	TMS470R1A256
TI	TMS470R1A288	TMS470R1A288
TI	TMS470R1A384	TMS470R1A384
TI	TMS470R1B512	TMS470R1B512
TI	TMS470R1B768	TMS470R1B768
TI	TMS470R1B1M	TMS470R1B1M
TI	TMS470R1VF288	TMS470R1VF288
TI	TMS470R1VF688	TMS470R1VF688
TI	TMS470R1VF689	TMS470R1VF689

Table 15: Supported microcontrollers (Continued)

*Not available for RDI, J-Link GDB Server

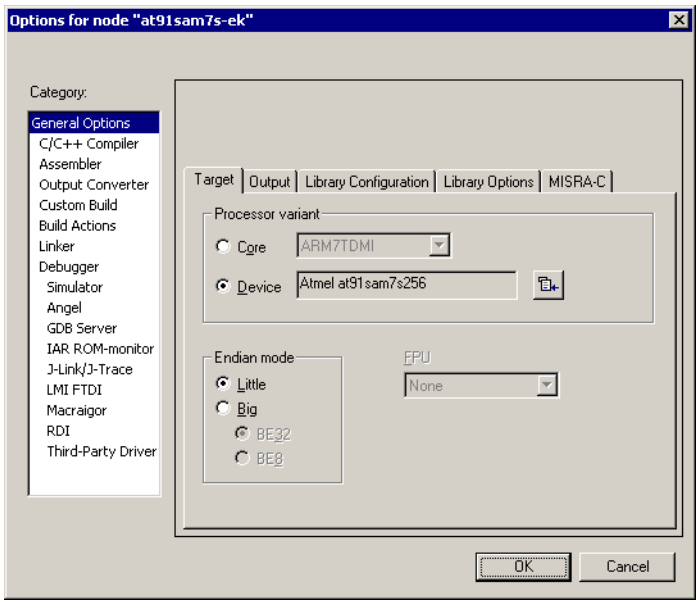
Using flash breakpoints

J-Link ARM FlashBP can be used by IAR Embedded Workbench.

IAR EMBEDDED WORKBENCH

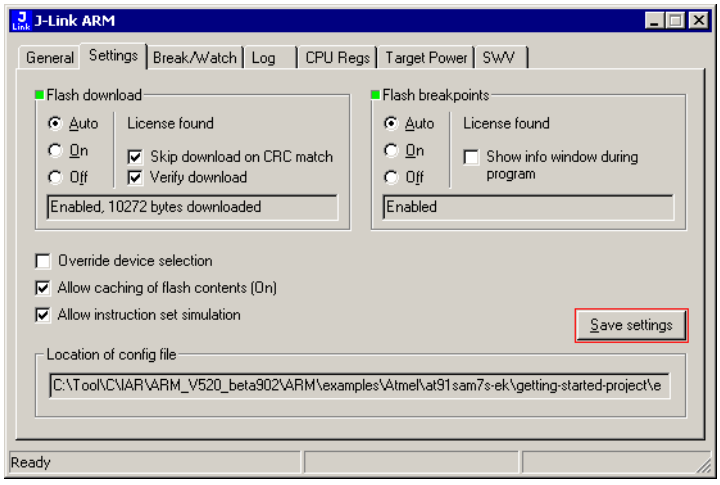
To use J-Link FlashBP with IAR Embedded Workbench is quite simple:

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



If you use the IAR project for the first time, the use of FlashBPs is set to **Auto**, which is the default value. For more information about different configurations for FlashBPs, refer to *Settings* on page 42.

Now you can start the debug session. If you run this project for the first time a settings file is created in which the configuration of FlashBPs is saved. This settings file is created for every project configuration (for example, Debug_RAM, Debug_FLASH), so you can save different J-Link FlashBP configurations for different project configurations. When the debug session starts, you should see the selected target in the **Device** tab of the J-Link status window. When the debug session is running you can modify the settings regarding FlashBPs, in the **Settings** tab and save them to the settings file.



Currently changes in this tab, will take effect next time the debug session is started.

Device specifics

This chapter gives additional information about specific devices.

Analog Devices

J-Link has been tested with the following MCUs from Analog Devices, but should work with any ARM7/9 and Cortex-M3 device:

- ADuC7020x62
- ADuC7021x32
- ADuC7021x62
- ADuC7021x62
- ADuC7022x32
- ADuC7022x62
- ADuC7024x62
- ADuC7025x32
- ADuC7025x62
- ADuC7026x62
- ADuC7027x62
- ADuC7030
- ADuC7031
- ADuC7032
- ADuC7033
- ADuC7060
- ADuC7128
- ADuC7129
- ADuC7229x126

ADUC7XXX

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

Software reset

A special reset strategy has been made available for Analog Devices ADuC7xxx MCUs. This special reset strategy is a software reset. "Software reset" means basically no reset, just changing the CPU registers such as PC and CPSR.

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

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

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

This information is applicable to the following devices:

- Analog ADuC7020x62
- Analog ADuC7021x32

- Analog ADuC7021x62
- Analog ADuC7022x32
- Analog ADuC7022x62
- Analog ADuC7024x62
- Analog ADuC7025x32
- Analog ADuC7025x62
- Analog ADuC7026x62
- Analog ADuC7027x62
- Analog ADuC7030
- Analog ADuC7031
- Analog ADuC7032
- Analog ADuC7033
- Analog ADuC7128
- Analog ADuC7129
- Analog ADuC7229x126

ATMEL

J-Link has been tested with the following ATMEL devices, but should work with any ARM7/9 and Cortex-M3 device:

- AT91SAM7A3
- AT91SAM7S32
- AT91SAM7S321
- AT91SAM7S64
- AT91SAM7S128
- AT91SAM7S256
- AT91SAM7S512
- AT91SAM7SE32
- AT91SAM7SE256
- AT91SAM7SE512
- AT91SAM7X128
- AT91SAM7X256
- AT91SAM7X512
- AT91SAM7XC128
- AT91SAM7XC256
- AT91SAM7XC512
- AT91RM9200
- AT91SAM9260
- AT91SAM9261
- AT91SAM9262
- AT91SAM9263

AT91SAM7

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

Reset strategy

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

It is recommended to use this reset strategy. This special reset strategy resets the peripherals by writing to the RSTC_CR register. Reseting the peripherals puts all peripherals in the defined reset state. This includes memory mapping register, which means that after reset flash is mapped to address 0. It is also possible to achieve the same effect by writing 0x4 to the RSTC_CR register located at address 0xfffffd00.

This information is applicable to the following devices:

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

Memory mapping

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

To achieve a defined mapping, there are two options:

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

Clearly 1 is the easiest solution and is recommended.

This information is applicable to the following devices:

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

Recommended init sequence

In order to work with an ATMEL AT91SAM7 device, it has to be initialized. The following paragraph describes the steps of an initialization sequence.

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

Example

```
/*********************************************************************
*
*       _Init()
*/
_Init() {
  __emulatorSpeed(30000);                              // Set JTAG speed to 30 kHz
  __writeMemory32(0xA5000004,0xFFFFFD00,"Memory");     // Perform peripheral reset
  __sleep(20000);
  __writeMemory32(0x00008000,0xFFFFFD44,"Memory");     // Disable Watchdog
  __sleep(20000);
  __writeMemory32(0x00000601,0xFFFFFC20,"Memory");     // PLL
  __sleep(20000);
  __writeMemory32(0x10191c05,0xFFFFFC2C,"Memory");     // PLL
  __sleep(20000);
  __writeMemory32(0x00000007,0xFFFFFC30,"Memory");     // PLL
  __sleep(20000);
  __writeMemory32(0x002f0100,0xFFFFFF60,"Memory");     // Set 1 wait state for
  __sleep(20000);                                      // flash (2 cycles)
  __emulatorSpeed(12000000);                           // Use full JTAG speed
}

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

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

AT91SAM9

These devices are based on ARM926EJ-S core. All devices of this family are supported by J-Link.

JTAG settings

We recommend using adaptive clocking.

This information is applicable to the following devices:

- AT91RM9200
- AT91SAM9260
- AT91SAM9261
- AT91SAM9262
- AT91SAM9263

Freescale

J-Link has been tested with the following Freescale devices, but should work with any ARM7/9 and Cortex-M3 device:

- MAC7101
- MAC7106
- MAC7111
- MAC7112
- MAC7116
- MAC7121
- MAC7122

- MAC7126
- MAC7131
- MAC7136
- MAC7141
- MAC7142

MAC71X

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

Luminary Micro

J-Link has been tested with the following Luminary Micro devices, but should work with any ARM7/9 and Cortex-M3 device:

- LM3S101
- LM3S102
- LM3S301
- LM3S310
- LM3S315
- LM3S316
- LM3S317
- LM3S328
- LM3S601
- LM3S610
- LM3S611
- LM3S612
- LM3S613
- LM3S615
- LM3S617
- LM3S618
- LM3S628
- LM3S801
- LM3S811
- LM3S812
- LM3S815
- LM3S817
- LM3S818
- LM3S828
- LM3S2110
- LM3S2139
- LM3S2410
- LM3S2412
- LM3S2432
- LM3S2533
- LM3S2620
- LM3S2637
- LM3S2651
- LM3S2730
- LM3S2739

- LM3S2939
- LM3S2948
- LM3S2950
- LM3S2965
- LM3S6100
- LM3S6110
- LM3S6420
- LM3S6422
- LM3S6432
- LM3S6610
- LM3S6633
- LM3S6637
- LM3S6730
- LM3S6938
- LM3S6952
- LM3S6965

STELLARIS LM3S100 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S300 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S600 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S800 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S2000 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S6100 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S6400 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S6700 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

STELLARIS LM3S6900 SERIES

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

NXP

J-Link has been tested with the following NXP devices, but should work with any ARM7/9 and Cortex-M3 device:

- LPC2101
- LPC2102
- LPC2103
- LPC2104
- LPC2105
- LPC2106
- LPC2109
- LPC2114
- LPC2119
- LPC2124
- LPC2129
- LPC2131
- LPC2132
- LPC2134
- LPC2136
- LPC2138
- LPC2141
- LPC2142
- LPC2144
- LPC2146
- LPC2148
- LPC2194
- LPC2212
- LPC2214
- LPC2292
- LPC2294
- LPC2364
- LPC2366
- LPC2368
- LPC2378
- LPC2468
- LPC2478
- PCF87750
- SJA2010
- SJA2510

LPC

Fast GPIO bug

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

Solution / Workaround

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

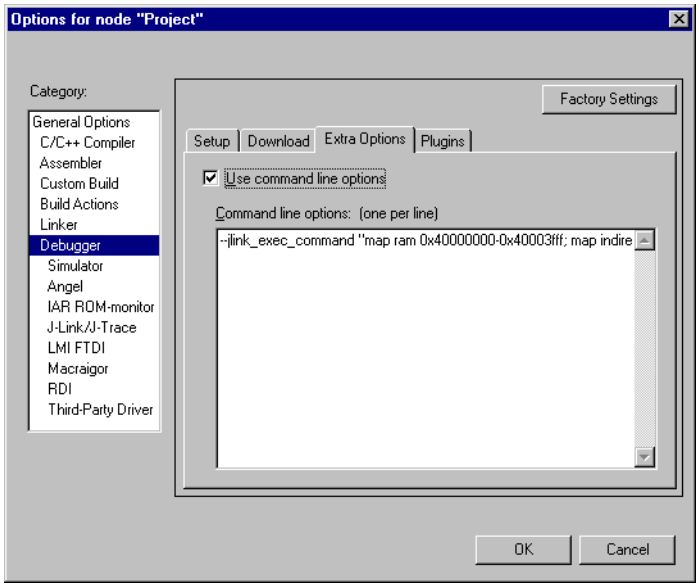
Define a 256 byte aligned area in RAM of the LPC target device with the J-Link command `map ram` and define afterwards the memory area which should be read indirect with the command `map indirectread` to use the indirectly reading feature of J-Link. Note that the data in the defined RAM area is saved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

This information is applicable to the following devices:

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

Example

J-Link command line options can be used with the C-SPY debugger of the IAR Embedded Workbench. Open the **Project options** dialog and select **Debugger**. Select **Use command line options** in the **Extra Options** tap and enter in the textfield `--jlink_exec_command "map ram 0x40000000-0x40003fff; map indirectread 0x3fffc000-0x3fffcfff; map exclude 0x3fffd000-0x3fffffff;"` as shown in the screenshot below.



With these additional commands are the values of the fast GPIO registers in the C-SPY debugger correct and can be used for debugging. For more information about J-Link command line options, refer to *Command strings* on page 48.

OKI

J-Link has been tested with the following OKI devices, but should work with any ARM7/9 and Cortex-M3 device:

- ML67Q4002
- ML67Q4003
- ML67Q4050
- ML67Q4051
- ML67Q4060

- ML67Q4061

ML67Q40X

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

ST Microelectronics

J-Link has been tested with the following ST Microelectronics devices, but should work with any ARM7/9 and Cortex-M3 device:

- STR710FZ1
- STR710FZ2
- STR711FR0
- STR711FR1
- STR711FR2
- STR712FR0
- STR712FR1
- STR712FR2
- STR715FR0
- STR730FZ1
- STR730FZ2
- STR731FV0
- STR731FV1
- STR731FV2
- STR735FZ1
- STR735FZ2
- STR736FV0
- STR736FV1
- STR736FV2
- STR750FV0
- STR750FV1
- STR750FV2
- STR751FR0
- STR751FR1
- STR751FR2
- STR752FR0
- STR752FR1
- STR752FR2
- STR755FR0
- STR755FR1
- STR755FR2
- STR755FV0
- STR755FV1
- STR755FV2
- STR911FM32
- STR911FM44
- STR911FW32
- STR911FW44
- STR912FM32

- STR912FM44
- STR912FW32
- STR912FW44
- STM32F101C6
- STM32F101C8
- STM32F101R6
- STM32F101R8
- STM32F101RB
- STM32F101V8
- STM32F101VB
- STM32F103C6
- STM32F103C8
- STM32F103R6
- STM32F103R8
- STM32F103RB
- STM32F103V8
- STM32F103VB

STR 71X

These devices are ARM7TDMI based.
All devices of this family are supported by J-Link.

STR 73X

These devices are ARM7TDMI based.
All devices of this family are supported by J-Link.

STR 75X

These devices are ARM7TDMI-S based.
All devices of this family are supported by J-Link.

STR91X

These device are ARM966E-S based.
All devices of this family are supported by J-Link.

Flash erasing

The devices have 3 TAP controllers built-in. When starting `J-Link.exe`, it reports 3 JTAG devices. A special tool, J-Link STR9 Commander (`JLinkSTR91x.exe`) is available to directly access the flash controller of the device. This tool can be used to erase the flash of the controller even if a program is in flash which causes the ARM core to stall. For more information about the J-Link STR9 Commander, please refer to *J-Link STR91x Commander (Command line tool)* on page 22.

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

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

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

STM32

These device are Cortex-M3 based.
All devices of this family are supported by J-Link.

Option byte programming

we suggest to perform the programming of the option bytes directly from the target application. J-Link (or an additional software tool like J-Flash) does not support programming of the option bytes.

Read-protection

The user area internal flash of the STM32 devices can be protected against read by untrusted code. In order to unsecure a read-protected STM32 device, SEGGER offers a free command line tool which overrides the read-protection of a STM32 device. For more information about the J-Link STM32 Commander, please refer to *J-Link STM32 Commander (Command line tool)* on page 23.

Note:J-Flash ARM supports securing and unsecuring a STM32 device, too.

Hardware watchdog

The hardware watchdog of a STM32 device can be enabled by programming the option bytes. If the hardware watchdog is enabled the device reset periodically if the watchdog timer is not refreshed and reaches 0. If the hardware watchdog is enabled by an application which is located in flash and which does not refresh the watchdog timer, the device can not be debugged anymore.

Disabling the hardware watchdog

In order to disable the hardware watchdog the option bytes have to be re-programmed. SEGGER offers a free command line tool which reprograms the option bytes in order to disable the hardware watchdog. For more information about the STM32 commander, please refer to *J-Link STM32 Commander (Command line tool)* on page 23.

Note: In order to re-program the option bytes they have to be erased first. Erasing the option bytes will read-protect the flash of the STM32. The STM32 commander will also override the read-protection of the STM32 device after disabling the watchdog. Please also note that unsecuring a read-protected device will cause a mass erase of the flash memory.

Texas Instruments

J-Link has been tested with the following Texas Intruments devices, but should work with any ARM7/9 and Cortex-M3 device:

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

TMS470

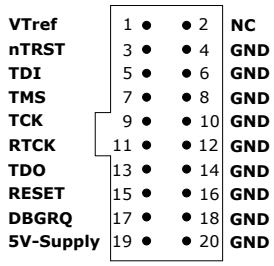
All devices of this family are supported by J-Link.

Hardware

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

20-pin JTAG/SWD connector

PINOUT FOR JTAG



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

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

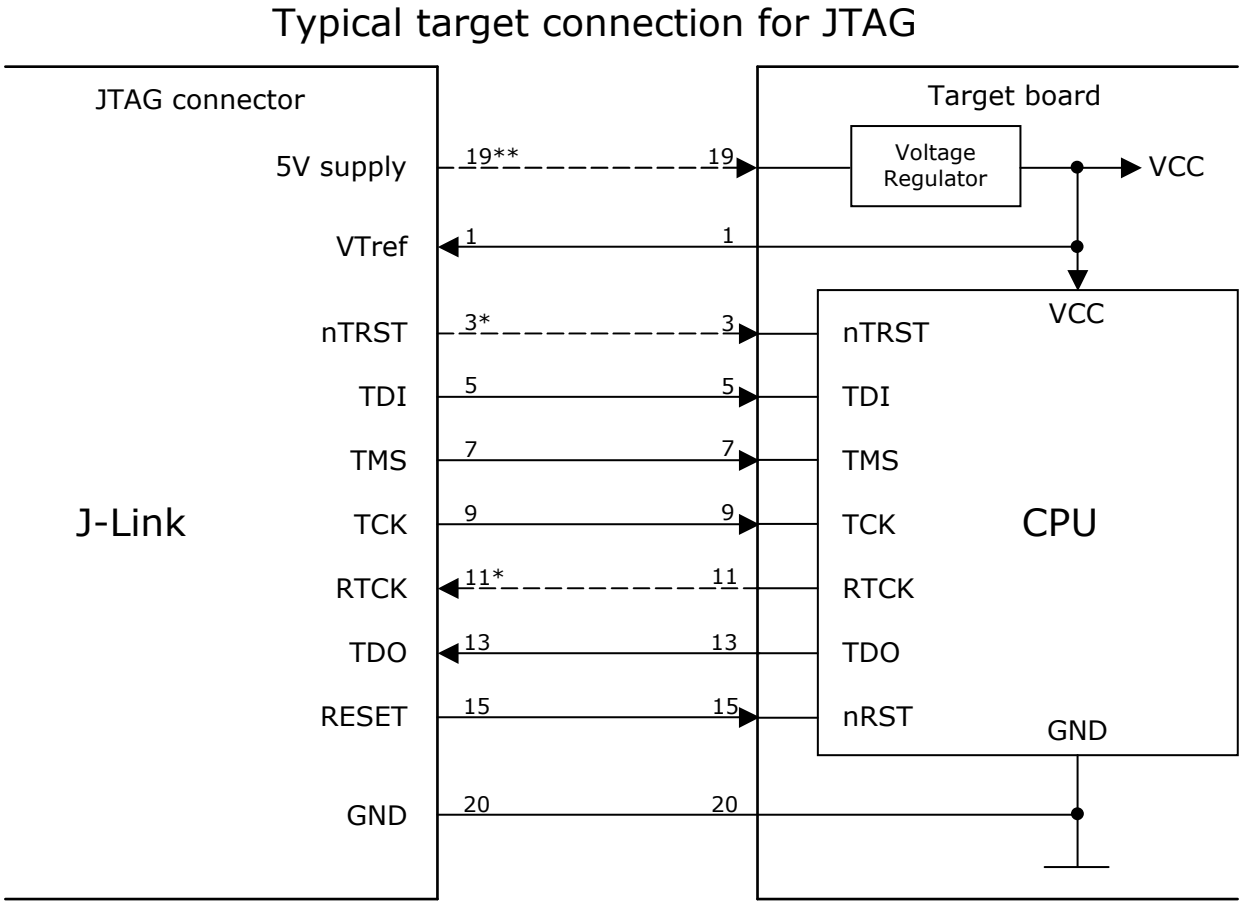
PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	Not connected	NC	This pin is not connected in J-Link.
3	nTRST	Output	JTAG Reset. Output from J-Link to the Reset signal of the target JTAG port. Typically connected to nTRST of the target CPU. This pin is normally pulled HIGH on the target to avoid unintentional resets when there is no connection.
5	TDI	Output	JTAG data input of the target CPU. It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU.
7	TMS	Output	JTAG mode set input of the target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU.
9	TCK	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.
11	RTCK	Input	Return test clock signal from the target. Some targets must synchronize the JTAG inputs to internal clocks. To assist in meeting this requirement, you can use a returned, and retimed, TCK to dynamically control the TCK rate. J-Link supports adaptive clocking, which waits for TCK changes to be echoed correctly before making further changes. Connect to RTCK if available, otherwise to GND.
13	TDO	Input	JTAG data output from the target CPU. Typically connected to TDO of the target CPU.
15	RESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
17	DBGRQ	NC	This pin is not connected in J-Link. It is reserved for compatibility with other equipment to be used as a debug request signal to the target system. Typically connected to DBGRQ if available, otherwise left open.
19	5V-Supply	Output	This pin can be used to supply power to the target hardware. Older J-Links may not be able to supply power on this pin. For more information about how to enable/disable the power supply, please refer to *Target power supply* on page 78.

Table 16: J-Link / J-Trace pinout

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

Target board design

We strongly advise following the recommendations given by the chip manufacturer. These recommendations are normally in line with the recommendations given in the table *Pinout for JTAG* on page 77. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.



* NTRST and RTCK may not be available on some CPUs.

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

Pull-up/pull-down resistors

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

Target power supply

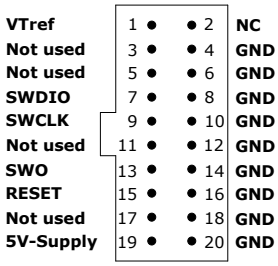
Pin 19 of the connector can be used to supply power to the target hardware. Supply voltage is 5 V, maximum current is 300 mA. The output current is monitored and protected against overload and short-circuit.

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

Command	Explanation
`power on`	Switch target power on
`power off`	Switch target power off
`power on perm`	Set target power supply default to on
`power off perm`	Set target power supply default to off

Table 17: Command List

PINOUT FOR SWD



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

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

PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	Not connected	NC	This pin is not connected in J-Link.
3	Not Used	NC	This pin is not used by J-Link. If the device is accessed via JTAG, this pin can be connected to nTRST, otherwise leave open.
5	Not used	NC	This pin is not used by J-Link. If the device is accessed via JTAG, this pin can be connected to TDI, otherwise leave open.
7	SWDIO	I/O	Single bi-directional data pin. A pull-up resistor is required. ARM recommends 100 kOhms.
9	SWCLK	Output	Clock signal to target CPU. It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TCK of the target CPU.
11	Not used	NC	This pin is not used by J-Link when operating in SWD mode. If the device is also accessed via JTAG, this pin can be connected to RTCK, otherwise leave open.
13	SWO	Output	Serial Wire Output trace port. (Optional, not required for SWD communication.)
15	RESET	I/O	Target CPU reset signal. Typically, connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
17	Not used	NC	This pin is not connected in J-Link.
19	5V-Supply	Output	This pin can be used to supply power to the target hardware. Older J-Links may not be able to supply power on this pin. For more information about how to enable/disable the power supply, please refer to *Target power supply* on page 80.

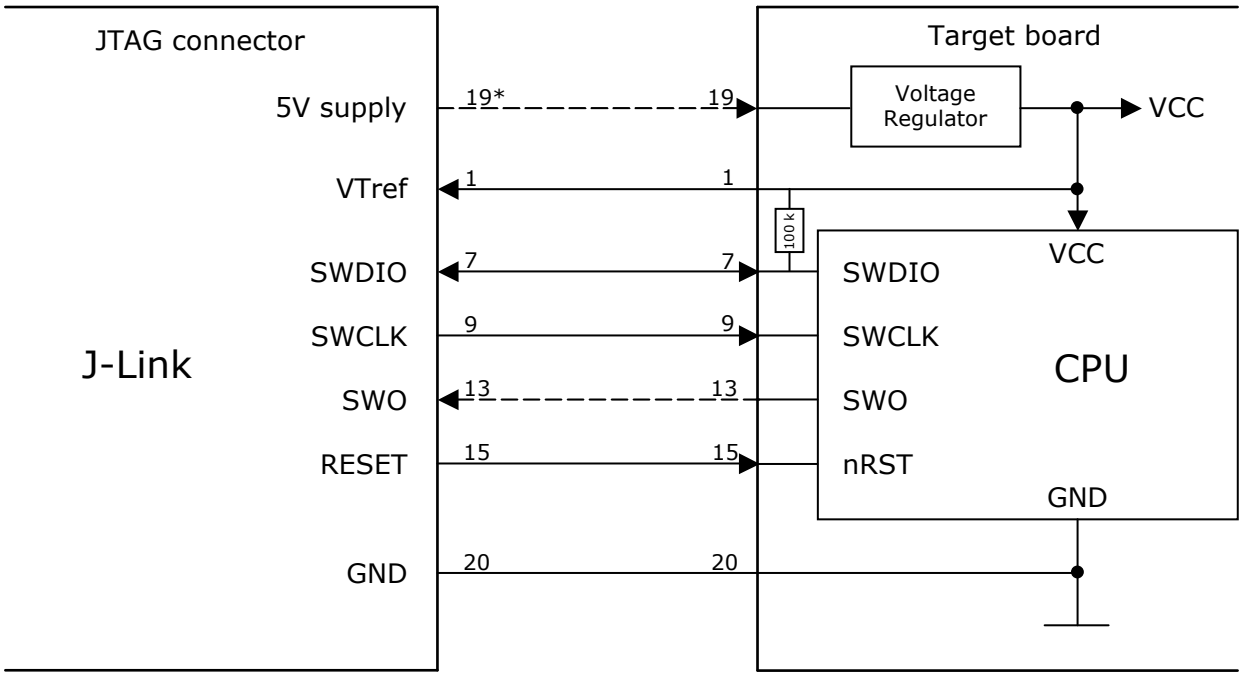
Table 18: J-Link / J-Trace SWD pinout

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

Target board design

We strongly advise following the recommendations given by the chip manufacturer. These recommendations are normally in line with the recommendations given in the table *Pinout for SWD* on page 79. In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

Typical target connection for SWD



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

Pull-up/pull-down resistors

A pull-up resistor is required on SWDIO on the target board. ARM recommends 100 kOhms.
In case of doubt you should follow the recommendations given by the semiconductor manufacturer.

Target power supply

Pin 19 of the connector can be used to supply power to the target hardware. Supply voltage is 5V, max. current is 300mA. The output current is monitored and protected against overload and short-circuit.

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

Command	Description
`power on`	Switch target power on
`power off`	Switch target power off
`power on perm`	Set target power supply default to on
`power off perm`	Set target power supply default to off

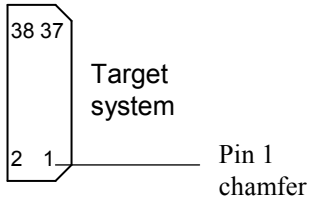
Table 19: Command List

38-pin Mictor JTAG and Trace connector

J-Trace provides a JTAG+Trace connector. This connector is a 38-pin mictor plug. It connects to the target via a 1-1 cable.

The connector on the target board should be "TYCO type 5767054-1" or a compatible receptacle. J-Trace supports 4, 8, and 16-bit data port widths with the high density target connector described below.

Target board trace connector

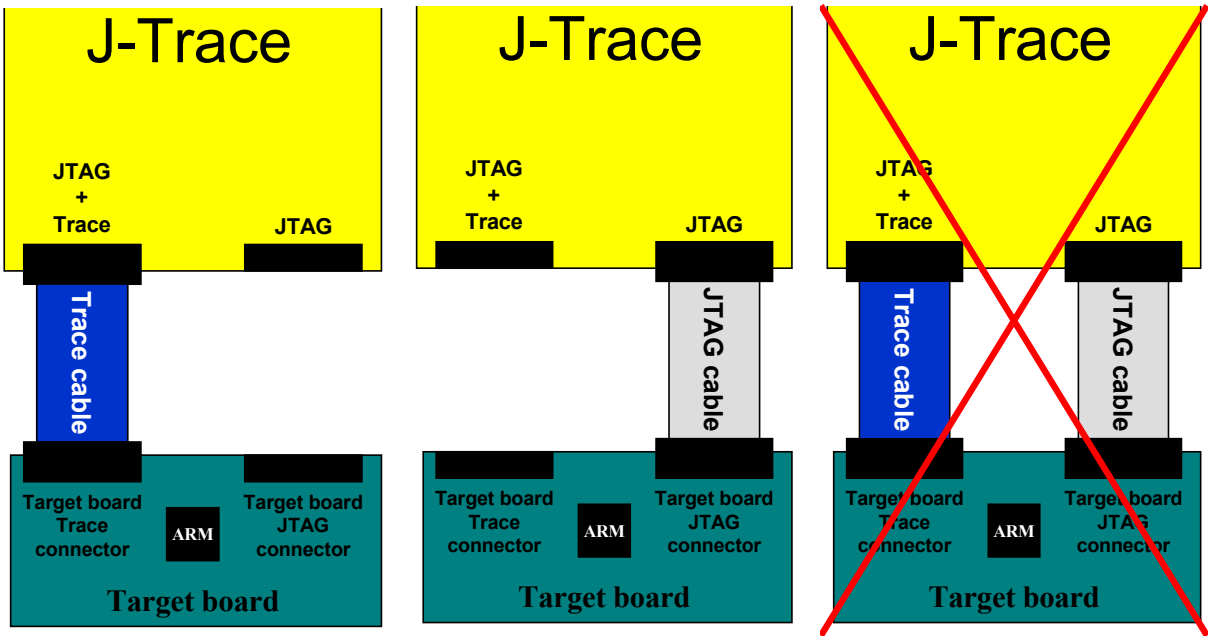


J-Trace can capture the state of signals PIPESTAT[2:0], TRACESYNC and TRACEPKT[n:0] at each rising edge of each TRACECLK or on each alternate rising or falling edge.

CONNECTING THE TARGET BOARD

J-Trace connects to the target board via a 38-pin trace cable. This cable has a receptacle on the one side, and a plug on the other side. Alternatively J-Trace can be connected with a 20-pin JTAG cable.

Warning: Never connect trace cable and JTAG cable at the same time because this may harm your J-Trace and/or your target.



PINOUT

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

PIN	SIGNAL	Description
1	NC	No connected.
2	NC	No connected.
3	NC	No connected.
4	NC	No connected.
5	GND	Signal ground.
6	TRACECLK	Clocks trace data on rising edge or both edges.
7	DBGRQ	Debug request.
8	DBGACK	Debug acknowledge from the test chip, high when in debug state.
9	RESET	Open-collector output from the run control to the target system reset.
10	EXTTRIG	Optional external trigger signal to the Embedded trace Macrocell (ETM). Not used. Leave open on target system.
11	TDO	Test data output from target JTAG port.
12	VTRef	Signal level reference. It is normally fed from Vdd of the target board and must not have a series resistor.
13	RTCK	Return test clock from the target JTAG port.
14	VSupply	Supply voltage. It is normally fed from Vdd of the target board and must not have a series resistor.
15	TCK	Test clock to the run control unit from the JTAG port.
16	Trace signal 12	Trace signal. For more information, refer to *Assignment of trace information pins between ETM architecture versions* on page 83.
17	TMS	Test mode select from run control to the JTAG port.
18	Trace signal 11	Trace signal. For more information, refer to *Assignment of trace information pins between ETM architecture versions* on page 83.
19	TDI	Test data input from run control to the JTAG port.
20	Trace signal 10	Trace signal. For more information, refer to *Assignment of trace information pins between ETM architecture versions* on page 83.
21	nTRST	Active-low JTAG reset
22	Trace signal 9	Trace signals. For more information, refer to *Assignment of trace information pins between ETM architecture versions* on page 83.
23	Trace signal 20	
24	Trace signal 8	
25	Trace signal 19	
26	Trace signal 7	
27	Trace signal 18	
28	Trace signal 6	
29	Trace signal 17	
30	Trace signal 5	
31	Trace signal 16	
32	Trace signal 4	
33	Trace signal 15	
34	Trace signal 3	
35	Trace signal 14	
36	Trace signal 2	
37	Trace signal 13	
38	Trace signal 1	

Table 20: JTAG+Trace connector pinout

ASSIGNMENT OF TRACE INFORMATION PINS BETWEEN ETM ARCHITECTURE VERSIONS

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

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

Table 21: Assignment of trace information pins between ETM architecture versions

TRACE SIGNALS

Data transfer is synchronized by TRACECLK.

Signal levels

The maximum capacitance presented by J-Trace at the trace port connector,

including the connector and interfacing logic, is less than 6pF. The trace port lines have a matched impedance of 50.

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

Clock frequency

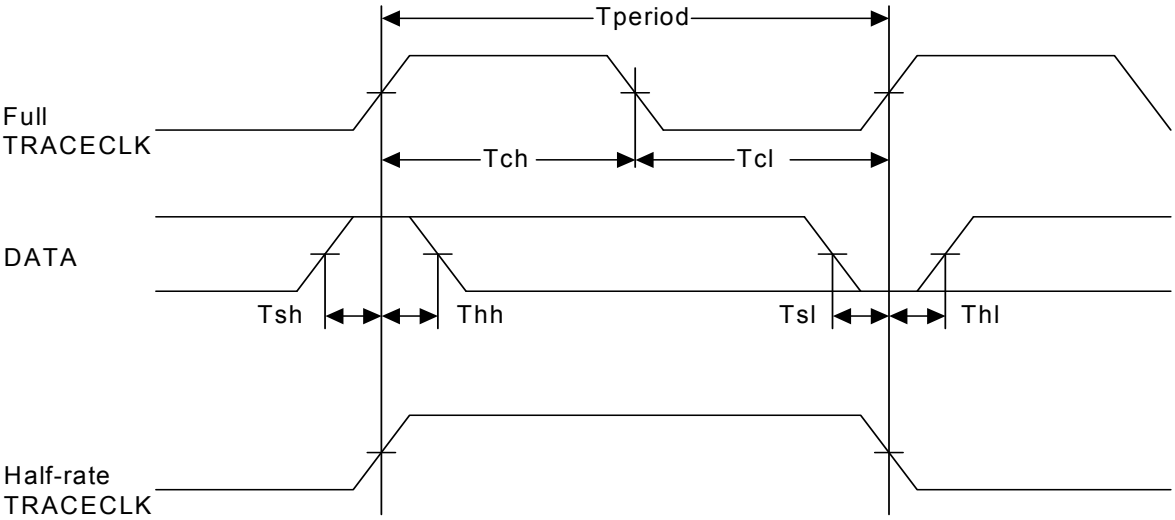
For capturing trace port signals synchronous to TRACECLK, J-Trace supports

a TRACECLK frequency of up to 200MHz. The following table shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACECLK.

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

Table 22: Clock frequency

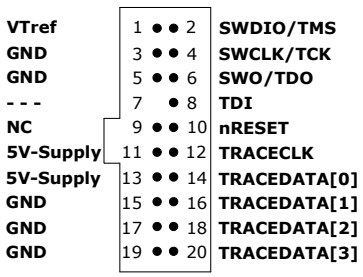
The diagram below shows the TRACECLK frequencies and the setup and hold timing of the trace signals with respect to TRACECLK.



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

19-pin JTAG/SWD and Trace connector

J-Trace for Cortex M3 provides a JTAG/SWD+Trace connector. This connector is a 19-pin connector. It connects to the target via an 1-1 cable.



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

PIN	SIGNAL	TYPE	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
2	SWDIO/TMS	I/O / output	JTAG mode set input of target CPU. This pin should be pulled up on the target. Typically connected to TMS of the target CPU.
4	SWCLK/TCK	Output	JTAG clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of the target CPU.
6	SWO/TDO	Input	JTAG data output from target CPU. Typically connected to TDO of the target CPU.
---	---	---	This pin (normally pin 7) is not existent on the 19-pin JTAG/SWD and Trace connector.
8	TDI	Output	JTAG data input of target CPU.- It is recommended that this pin is pulled to a defined state on the target board. Typically connected to TDI of the target CPU.
9	NC	NC	Not connected inside J-Link. Leave open on target hardware.
10	nRESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".

Table 23: 19-pin JTAG/SWD and Trace pinout

PIN	SIGNAL	TYPE	Description
11	5V-Supply	Output	This pin can be used to supply power to the target hardware. For more information about how to enable/disable the power supply, please refer to *Target power supply* on page 85.
12	TRACECLK	Input	Input trace clock. Trace clock = 1/2 CPU clock.
13	5V-Supply	Output	This pin can be used to supply power to the target hardware. For more information about how to enable/disable the power supply, please refer to *Target power supply* on page 85.
14	TRACEDATA[0]	Input	Input Trace data pin 0.
16	TRACEDATA[1]	Input	Input Trace data pin 0.
18	TRACEDATA[2]	Input	Input Trace data pin 0.
20	TRACEDATA[3]	Input	Input Trace data pin 0.

Table 23: 19-pin JTAG/SWD and Trace pinout (Continued)

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

TARGET POWER SUPPLY

Pin 19 of the connector can be used to supply power to the target hardware. Supply voltage is 5V, max. current is 300mA. The output current is monitored and protected against overload and short-circuit.

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

Command	Explanation
`power on`	Switch target power on
`power off`	Switch target power off
`power on perm`	Set target power supply default to on
`power off perm`	Set target power supply default to off

Table 24: Command List

RESET, nTRST

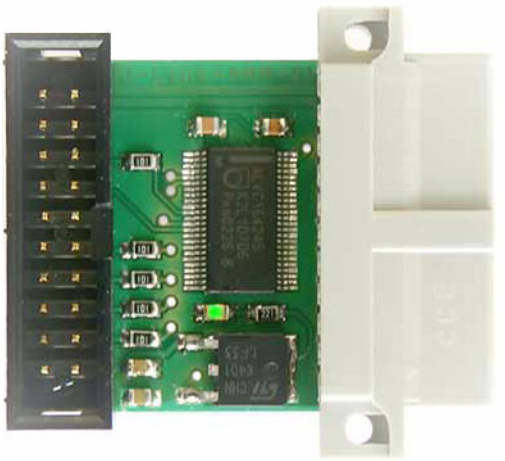
The TAP controller and ICE logic is reset independently from the ARM core with nTRST. For the ARM core to operate correctly, it is essential that both signals are asserted after power-up.

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).

Adapters

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.



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

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.

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.

J-Link / J-Trace models

This chapter gives an overview about the different J-Link / J-Trace models. In addition to that, it gives information about the changes between the hardware versions.

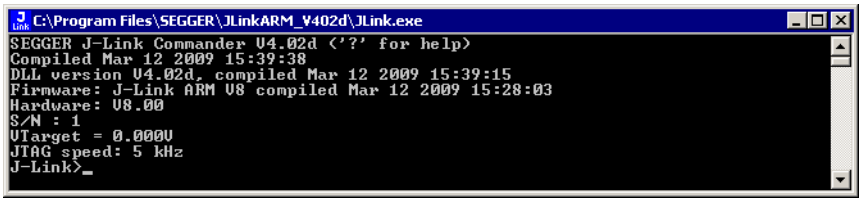
Introduction

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

- J-Link ARM
- J-Trace ARM
- J-Trace for Cortex-M3

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

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



J-Link ARM

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

ADDITIONAL FEATURES

- Serial Wire Debug supported *
- Serial Wire Viewer supported *
- Download speed up to 720 KBytes/second **
- DCC speed up to 800 Kbytes/second **

* = Supported by J-Link hardware version 6

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

SPECIFICATIONS*

Power Supply	USB powered <50mA if target power is off.
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 tolerant

Table 25: J-Link specifications

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 Microsoft Windows XP Microsoft Windows XP x64 Microsoft Windows 2003 Microsoft Windows 2003 x64 Microsoft Windows Vista Microsoft Windows Vista x64

Table 25: J-Link specifications (Continued)

* = J-Link hardware revision 5 and up.

DOWNLOAD SPEED

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

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

Table 26: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 107.

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

HARDWARE VERSIONS

Versions 1-3

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.

Target Interface

nTRST is open drain + 4K7 pull up
RESET is open drain

Version 4

Identical to version 3.0 with the following exception:

Target Interface

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

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

Version 5.2

Identical to version 5.0 with the following exception:

Target Interface

nTRST is push-pull type
RESET is open drain

Version 5.3

Identical to version 5.2 with the following exception:

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

Version 5.4

Identical to version 5.3 with the following exception:

- JTAG interface is 5V tolerant.

Version 6.0

Identical to version 5.4 with the following exception:

- Outputs can be tristated (Effectively disabling the JTAG interface)
- J-Link supports SWV (Speed limited to 500 kHz)

Version 7.0

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

Version 8.0

- SWD support for non-3.3V targets.

J-Trace ARM

J-Trace ARM is a JTAG emulator designed for ARM cores which includes trace (ETM) support. J-Trace can also be used as a J-Link.

ADDITIONAL FEATURES

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

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

SPECIFICATIONS FOR J-TRACE

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 Microsoft Windows XP Microsoft Windows XP x64 Microsoft Windows 2003 Microsoft Windows 2003 x64 Microsoft Windows Vista Microsoft Windows Vista x64

Table 27: J-Trace specifications

DOWNLOAD SPEED

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

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

Table 28: Download speed differences between hardware revisions

All tests have been performed in the testing environment which is described on *Measuring download speed* on page 107.

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

HARDWARE VERSIONS

Version 1

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

J-Trace for Cortex-M3

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

ADDITIONAL FEATURES

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

DOWNLOAD SPEED

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

Hardware	Cortex-M3 *via SWD*
J-Trace Rev. I	190 Kbytes/s (12MHz SWD)

Table 29: Download speed differences between hardware revisions

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

Background information

This chapter provides background information about JTAG and ARM. The ARM7 and ARM9 architecture is based on *Reduced Instruction Set Computer* (RISC) principles. The instruction set and the related decode mechanism are greatly simplified compared with microprogrammed *Complex Instruction Set Computer* (CISC).

JTAG

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

TEST ACCESS PORT (TAP)

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

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

Table 30: Test access port

DATA REGISTERS

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

Bypass data register

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

Boundary-scan data register

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

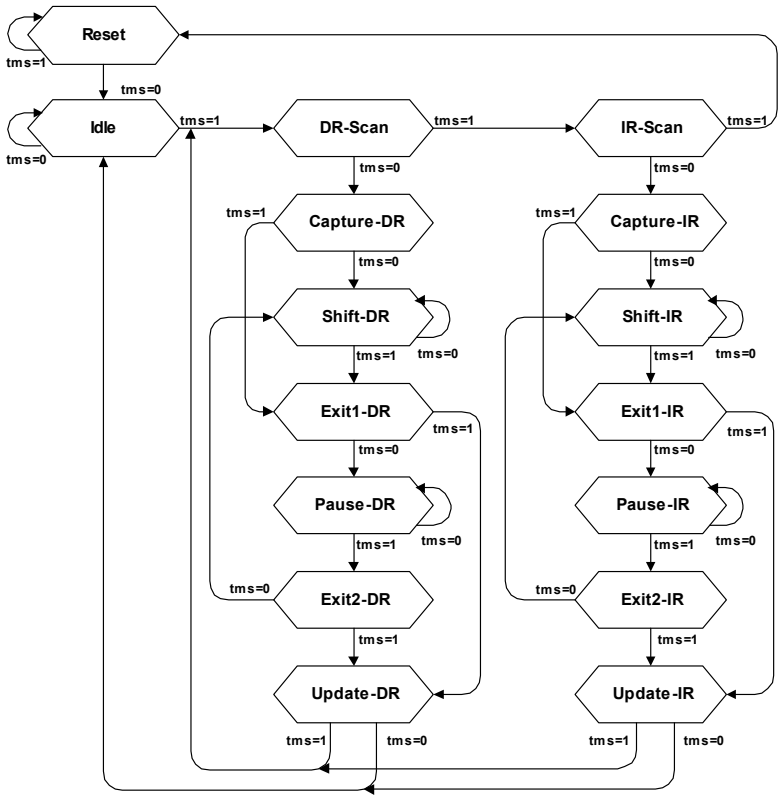
INSTRUCTION REGISTER

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

THE TAP CONTROLLER

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

TAP controller state diagram



State descriptions

Reset

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

Idle

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

DR-Scan

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

IR-Scan

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

Capture-DR

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

Shift-DR

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

Exit1-DR

Temporary controller state.

Pause-DR

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

Exit2-DR

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

Update-DR

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

Capture-IR

Instructions may be loaded in parallel into the instruction register.

Shift-IR

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

Exit1-IR

Temporary controller state.

Pause-IR

Wait state that temporarily halts the instruction shifting.

Exit2-IR

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

Update-IR

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

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.

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

Table 31: ARM processor modes

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

Table 32: Registers of the ARM core

= 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.

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, whereas 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 any limitations.

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).

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, or 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.

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

Table 33: Function and mapping of EmbeddedICE registers

Register	Width	Function
0x14	9	Watchpoint 1 control value
0x15	8	Watchpoint 1 control mask

Table 33: Function and mapping of EmbeddedICE registers

For more information about EmbeddedICE, see the technical reference manual of your ARM CPU. (*www.arm.com*)

Embedded Trace Macrocell (ETM)

Embedded Trace Macrocell (ETM) provides comprehensive debug and trace facilities for ARM processors. ETM allows to capture information on the processor's state without affecting the processor's performance. The trace information is exported immediately after it has been captured, through a special trace port.

Microcontrollers that include an ETM allow detailed program execution to be recorded and saved in real time. This information can be used to analyze program flow and execution time, perform profiling and locate software bugs that are otherwise very hard to locate. A typical situation in which code trace is extremely valuable, is to find out how and why a "program crash" occurred in case of a runaway program count.

A debugger provides the user interface to J-Trace and the stored trace data. The debugger enables all the ETM facilities and displays the trace information that has been captured. J-Trace is seamlessly integrated into the IAR Embedded Workbench® IDE. The advanced trace debugging features is used with the IAR C-SPY debugger.

TRIGGER CONDITION

The ETM can be configured in software to store trace information only after a specific sequence of conditions. When the trigger condition occurs the trace capture stops after a programmable period.

CODE TRACING AND DATA TRACING

Code trace

Code tracing means that the processor outputs trace data which contain information about the instructions that have been executed at last.

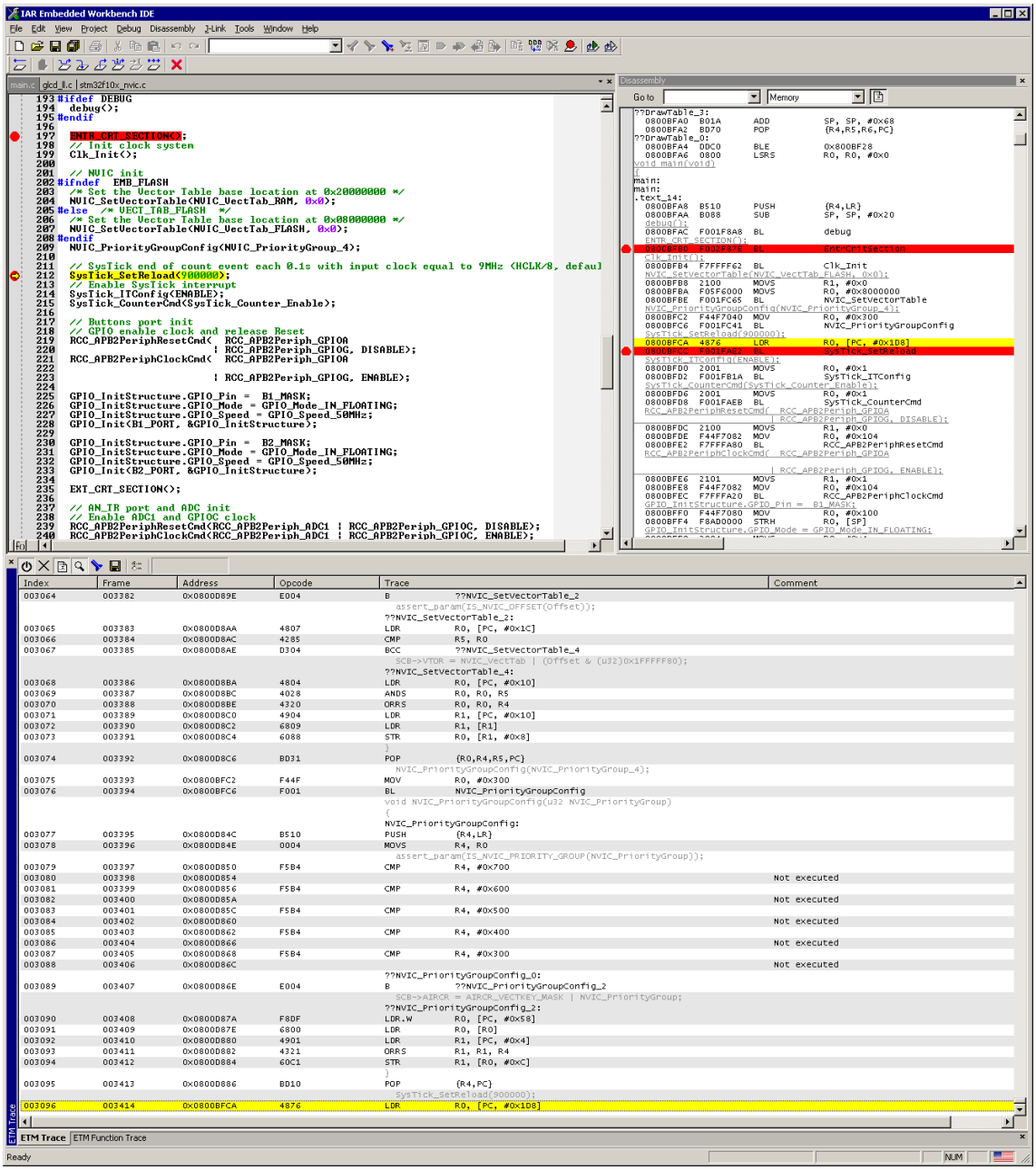
Data trace

Data tracing means that the processor outputs trace data about memory accesses (read / write access to which address and which data has been read / stored). In general, J-Trace supports data tracing, but it depends on the debugger if this option is available or not. Note that when using data trace, the amount of trace data to be captured rises enormously.

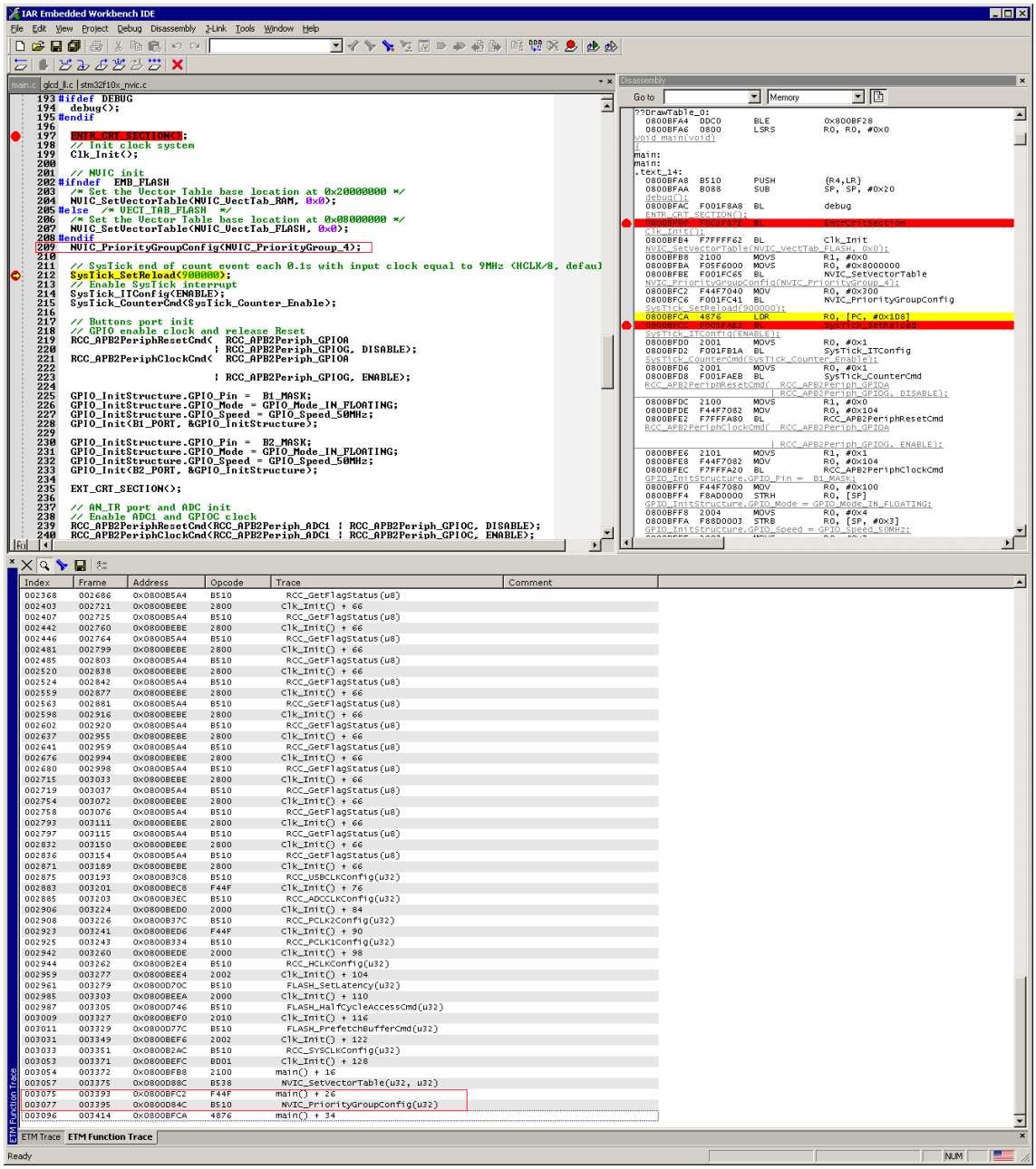
J-TRACE INTEGRATION EXAMPLE

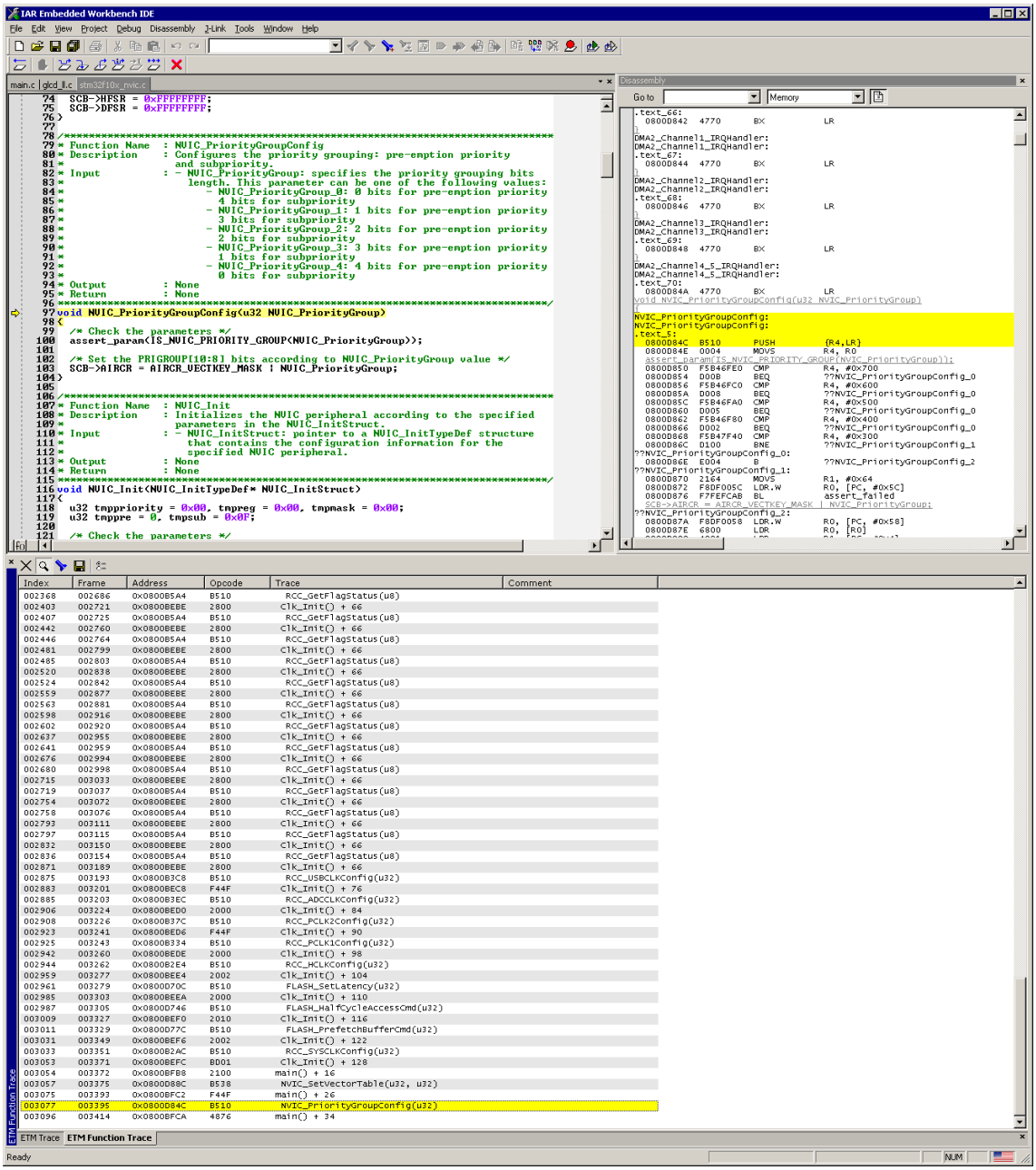
In the following a sample integration of J-Trace and the trace functionality on the debugger side is shown.

Code coverage - Disassembly tracing



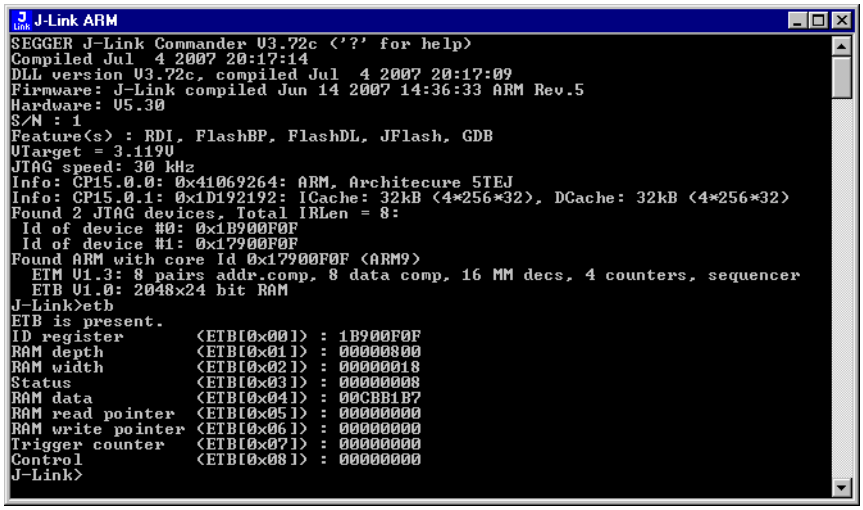
Code coverage - Source code tracing




IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly J-Link Tools Window Help
main.c | gicd_ll.c | stm32f10x_nvic.c
Disassembly
Go to
Memory
ETM Trace
ETM Function Trace
Ready

Embedded Trace Buffer (ETB)

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



The result of the limited buffer size is that not more data can be traced than the buffer can hold. Throu this limitation is an ETB not in every case an fully-fledged alternative to the direct access to an ETM via J-Trace.

Flash programming

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

HOW DOES FLASH PROGRAMMING VIA J-LINK / J-TRACE WORK?

This requires extra code. This extra code typically downloads a program into the RAM of the target system, which is able to erase and program the flash. This program is called RAM code and "knows" how to program the flash; it contains an implementation of the flash programming algorithm for the particular flash. Different flash chips have different programming algorithms; the programming algorithm also depends on other things such as endianess of the target system and organization of the flash memory (for example 1 * 8 bits, 1 * 16 bits, 2 * 16 bits or 32 bits). The RAM code requires data to be programmed into the flash memory. There are 2 ways of supplying this data: Data download to RAM or data download via DCC.

DATA DOWNLOAD TO RAM

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

DATA DOWNLOAD VIA DCC

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

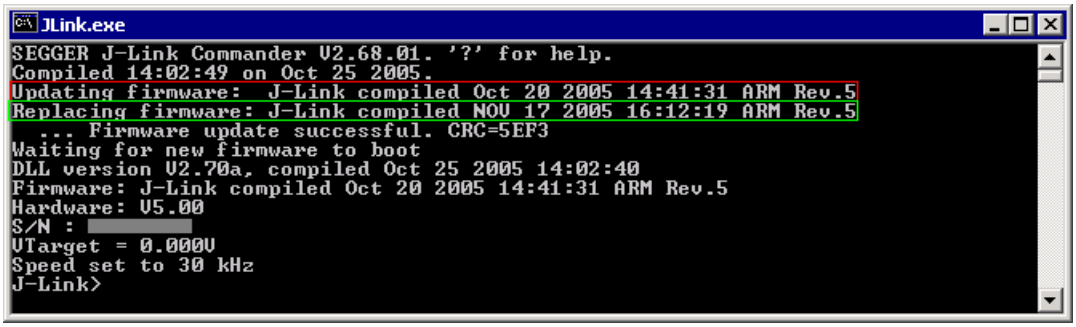
J-Link / J-Trace firmware

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

FIRMWARE UPDATE

Every time you connect to J-Link / J-Trace, JLinkARM.dll checks if its embedded firmware is newer than the one used the J-Link / J-Trace. The DLL will then update the firmware automatically. This process takes less than 3 seconds and does not require a reboot.

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



In the screenshot:

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

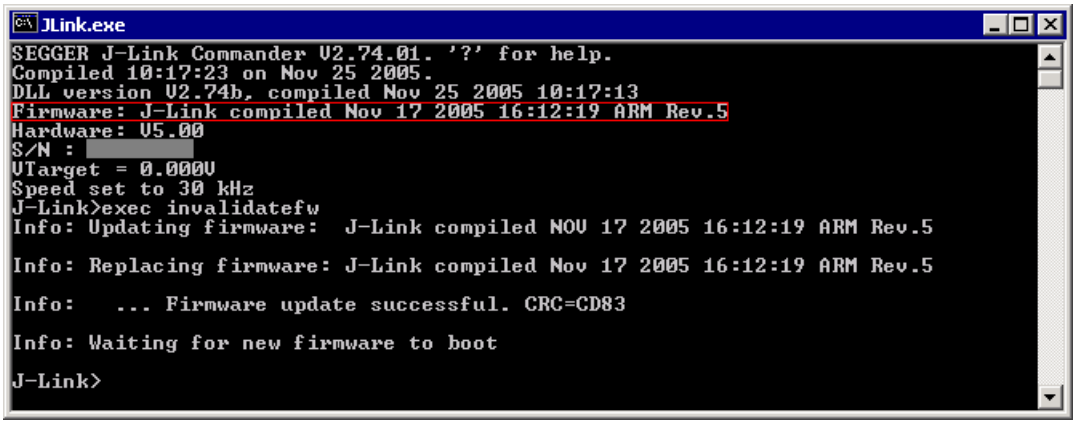
INVALIDATING THE FIRMWARE

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

Note: Downdating J-Link / J-Trace is not recommended, you do it at your own risk!

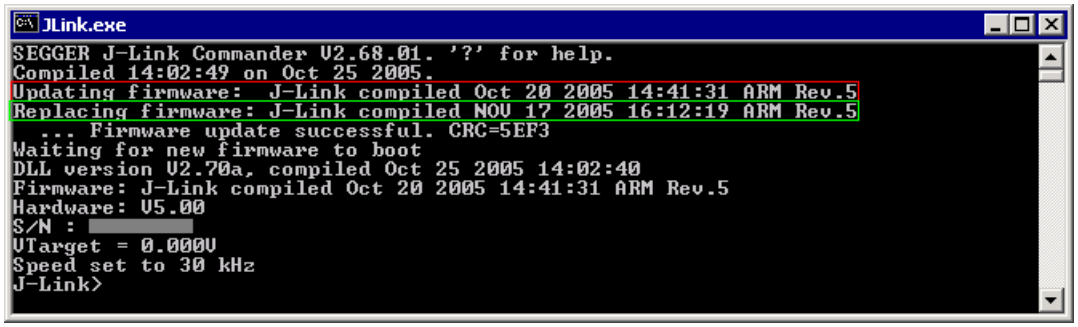
Note: Note also the firmware embedded in older versions of JLinkARM.dll might not execute properly with newer hardware versions.

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



In the screenshot, the red box contains information about the formerly used J-Link / J-Trace firmware version.

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



In the screenshot:

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

Designing the target board for trace

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

Overview of high-speed board design

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

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

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

AVOIDING STUBS

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

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

MINIMIZING SIGNAL SKEW (BALANCING PCB TRACK LENGTHS)

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

MINIMIZING CROSSTALK

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

USING IMPEDANCE MATCHING AND TERMINATION

Termination is almost certainly necessary, but there are some circumstances where it is not required. The decision is related to track length between the ASIC and the JTAG+Trace connector, see *Terminating the trace signal*, page 105 for further reference.

Terminating the trace signal

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

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

Matched impedance

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

Series (source) termination

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

DC parallel termination

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

Caution:

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

RULES FOR SERIES TERMINATORS

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

3. The series resistor must be placed as close as possible to the ASIC pin (less than 0.5 inches).
4. 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.
5. A source terminated signal is only valid at the end of the signal path. At any point between the source and the end of the track, the signal appears distorted because of reflections. Any device connected between the source and the end of the signal path therefore sees the distorted signal and might not operate correctly. Care must be taken not to connect devices in this way, unless the distortion does not affect device operation.

Signal requirements

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

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

Table 34: Signal requirements

Support and FAQs

This chapter contains troubleshooting tips together with solutions for common problems which might occur when using J-Link / J-Trace. There are several steps you can take before contacting support. Performing these steps can solve many problems and often eliminates the need for assistance. This chapter also contains a collection of frequently asked questions (FAQs) with answers.

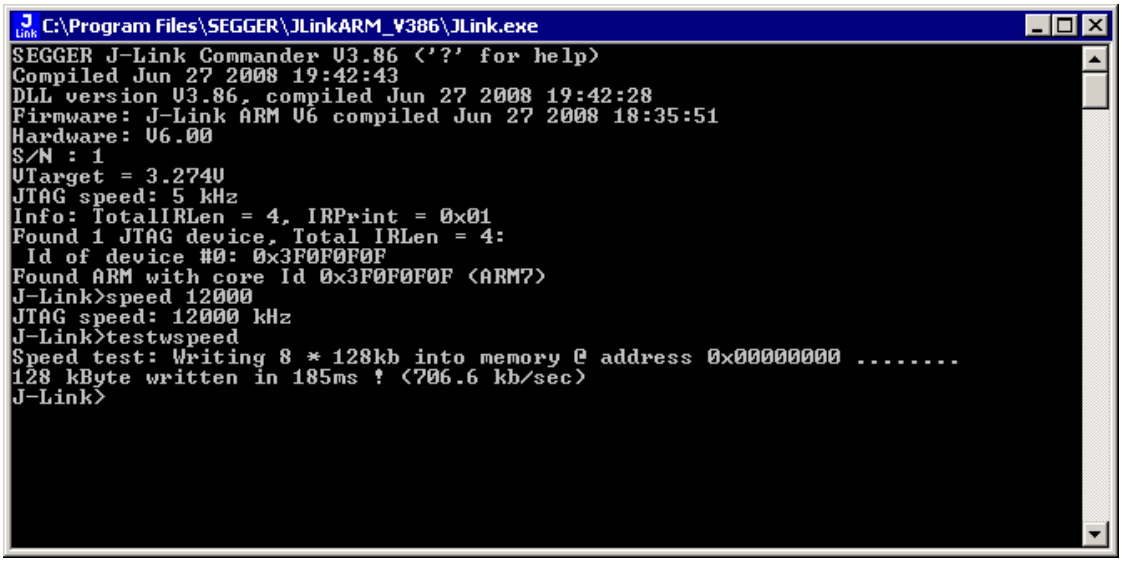
Measuring download speed

TEST ENVIRONMENT

JLink.exe has been used for measurement performance. The hardware consisted of:

- PC with 2.6 GHz Pentium 4, running Win2K
- USB 2.0 port
- USB 2.0 hub
- J-Link
- Target with ARM7 running at 50MHz.

Below is a screenshot of `JLink.exe` after the measurement has been performed.



Troubleshooting

GENERAL PROCEDURE

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

6. Close all running applications on your host system.
7. Disconnect the J-Link / J-Trace device from USB.
8. Disable power supply on the target.
9. Re-connect J-Link / J-Trace with the host system (attach USB cable).
10. Enable power supply on the target.

11 Try your target application again. If the problem remains, with the rest of the procedure.

12 Close all running applications on your host system again.

13 Disconnect the J-Link / J-Trace device from USB.

14 Disable power supply on the target.

15 Re-connect J-Link / J-Trace with the host system (attach the USB cable).

16 Enable power supply on the target.

17 Start `JLink.exe`.

18 If `JLink.exe` displays the J-Link / J-Trace serial number and the target processor's core ID, the J-Link / J-Trace is working properly and cannot be the cause of your problem.

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

TYPICAL PROBLEM SCENARIOS

J-Link / J-Trace LED is off

Meaning:

The USB connection does not work.

Remedy:

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

J-Link / J-Trace LED is flashing at a high frequency

Meaning:

J-Link / J-Trace could not be enumerated by the USB controller.

Most likely reasons:

a.) Another program is already using J-Link / J-Trace.
b.) The J-Link USB driver does not work correctly.

Remedy:

a.) Close all running applications and try to reinitialize J-Link / J-Trace by disconnecting and reconnecting it.
b.) If the LED blinks permanently, check the correct installation of the J-Link USB driver. Deinstall and reinstall the driver as shown in chapter *Setup* on page 17.

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:

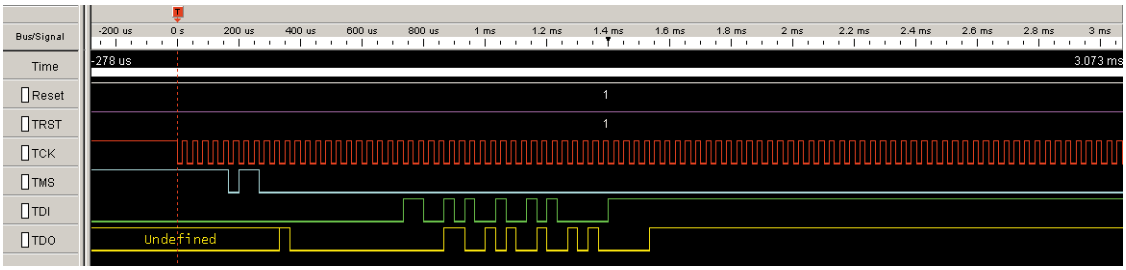
Follow the steps described in section *General procedure* on page 107.

Signal analysis

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

START SEQUENCE

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



The sequence consists of the following sections:

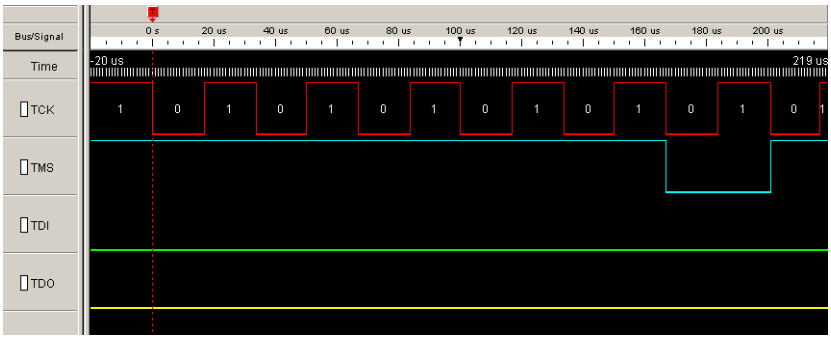
- 5 clocks: TDI low, TMS high. Brings TAP controller into RESET state
- 1 clock: TDI low, TMS low: Brings TAP controller into IDLE state
- 2 clocks: TDI low, TMS high: Brings TAP controller into IR-SCAN state
- 2 clocks: TDI low, TMS low: Brings TAP controller into SHIFT-IR state
- 32 clocks: TMS low,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

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

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

Zoom-in

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



TROUBLESHOOTING

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

Contacting support

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section *General procedure* on page 107. You may also try your J-Link / J-Trace with another PC and if possible with another target system to see if it works there. If the device functions correctly, the USB setup on the original machine or your target hardware is the source of the problem, not J-Link / J-Trace.

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

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

Frequently Asked Questions

Supported CPUs

Q: Which CPUs are supported?

A: J-Link / J-Trace should work with any ARM7/9 and Cortex-M3 core. For a list of supported cores, see section *Supported ARM Cores* on page 12.

Maximum JTAG speed

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.

Maximum download speed

Q: What is the maximum download speed?

A: The maximum download speed is currently about 720 Kbytes/second when downloading into RAM; Communication with a RAM-image via DCC can be still faster. However, the actual speed depends on various factors, such as JTAG, clock speed, host CPU core etc.

ICE register access

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.

Using DCC with J-Link

Q: Can I use J-Link / J-Trace to communicate with a running target via DCC?

A: Yes. The DLL includes functions to communicate via DCC. However, you can also program DCC communication yourself by accessing the relevant ICE registers through J-Link / J-Trace.

Read status of JTAG pins

Q: Can J-Link / J-Trace read back the status of the JTAG pins?

A: Yes, the status of all pins can be read. This includes the outputs of J-Link / J-Trace as well as the supply voltage, which can be useful to detect hardware problems on the target system.

Advantage of more expensive JTAG probes

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.

J-Link support of ETM

Q: Does J-Link support the Embedded Trace Macrocell (ETM)?

A: No. ETM requires another connection to the ARM chip and a CPU with built-in ETM. Most current ARM7 / ARM9 devices do not have ETM built-in.

J-Link support of ETB

Q: Does J-Link support the Embedded Trace Buffer (ETB)?

A: Yes. J-Link supports ETB. Most current ARM7 / ARM9 devices do not have ETB 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.

Registers on ARM 7 / ARM 9 targets

Q: I'm running `J-Link.exe` in parallel to my debugger, on an ARM 7 target. I can read memory okay, but the processor registers are different. Is this normal?

A: If memory on an ARM 7/9 target is read or written the processor registers are modified. When memory read or write operations are performed, J-Link preserves the register values before they are modified. The register values shown in the debugger's register window are the preserved ones. If now a second instance, in this case `J-Link.exe`, reads the processor registers, it reads the values from the hardware, which are the modified ones. This is why it shows different register values.

Glossary

This chapter describes important terms used throughout this manual.

Adaptive clocking

A technique in which a clock signal is sent out by J-Link / J-Trace. J-Link / J-Trace waits for the returned clock before generating the next clock pulse. The technique allows the J-Link / J-Trace interface unit to adapt to differing signal drive capabilities and differing cable lengths.

Application Program Interface

A specification of a set of procedures, functions, data structures, and constants that are used to interface two or more software components together.

Big-endian

Memory organization where the least significant byte of a word is at a higher address than the most significant byte. See Little-endian.

Cache cleaning

The process of writing dirty data in a cache to main memory.

Coprocessor

An additional processor that is used for certain operations, for example, for floating-point math calculations, signal processing, or memory management.

Dirty data

When referring to a processor data cache, data that has been written to the cache but has not been written to main memory is referred to as dirty data. Only write-back caches can have dirty data because a write-through cache writes data to the cache and to main memory simultaneously. See also cache cleaning.

Dynamic Linked Library (DLL)

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

Embedded Trace Macrocell (ETM)

ETM is additional hardware provided by debuggable ARM processors to aid debugging with trace functionality.

Embedded Trace Buffer (ETB)

ETB is a small, circular on-chip memory area where trace information is stored during capture.

EmbeddedICE

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

Halfword

A 16-bit unit of information. Contents are taken as being an `unsigned integer` unless otherwise stated.

Host

A computer which provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

ICache

Instruction cache.

ICE Extension Unit

A hardware extension to the EmbeddedICE logic that provides more breakpoint units.

ID

Identifier.

IEEE 1149.1

The IEEE Standard which defines TAP. Commonly (but incorrectly) referred to as JTAG.

Image

An executable file that has been loaded onto a processor for execution.

In-Circuit Emulator (ICE)

A device enabling access to and modification of the signals of a circuit while that circuit is operating.

Instruction Register

When referring to a TAP controller, a register that controls the operation of the TAP.

IR

See Instruction Register.

Joint Test Action Group (JTAG)

The name of the standards group which created the IEEE 1149.1 specification.

Little-endian

Memory organization where the least significant byte of a word is at a lower address than the most significant byte. See also Big-endian.

Memory coherency

A memory is coherent if the value read by a data read or instruction fetch is the value that was most recently written to that location. Obtaining memory coherency is difficult when there are multiple possible physical locations that are involved, such as a system that has main memory, a write buffer, and a cache.

Memory management unit (MMU)

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

Memory Protection Unit (MPU)

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

Multi-ICE

Multi-processor EmbeddedICE interface. ARM registered trademark.

RESET

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

nTRST

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

Open collector

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

Processor Core

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

Program Status Register (PSR)

Contains some information about the current program and some information about the current processor state. Often, therefore, also referred to as Processor Status Register.

Also referred to as Current PSR (CPSR), to emphasize the distinction to the Saved PSR (SPSR). The SPSR holds the value the PSR had when the current function was called, and which will be restored when control is returned.

Remapping

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

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

Remote Debug Interface (RDI)

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

RTCK

Returned TCK. The signal which enables Adaptive Clocking.

RTOS

Real Time Operating System.

Scan Chain

A group of one or more registers from one or more TAP controllers connected between TDI and TDO, through which test data is shifted.

Semihosting

A mechanism whereby the target communicates I/O requests made in the application code to the host system, rather than attempting to support the I/O itself.

SWI

Software Interrupt. An instruction that causes the processor to call a programer-specified subroutine. Used by ARM to handle semihosting.

TAP Controller

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

Target

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

TCK

The electronic clock signal which times data on the TAP data lines TMS, TDI, and TDO.

TDI

The electronic signal input to a TAP controller from the data source (upstream). Usually, this is seen connecting the J-Link / J-Trace Interface Unit to the first TAP controller.

TDO

The electronic signal output from a TAP controller to the data sink (downstream). Usually, this is seen connecting the last TAP controller to the J-Link / J-Trace Interface Unit.

Test Access Port (TAP)

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

Transistor-transistor logic (TTL)

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

Watchpoint

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

Word

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

Index

T

W

Numerics