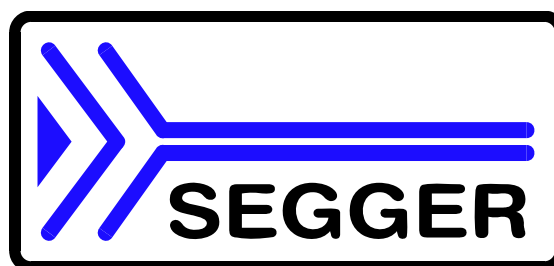


# ***J-Link GDB Server***

**User's guide of the  
J-Link GDB Server**

**Version 3.50  
Manual Rev. 1**



**A product of SEGGER Microcontroller Systeme GmbH**

## Disclaimer

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

## Copyright notice

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

© 2006 SEGGER Microcontroller Systeme GmbH, Hilden / Germany

## Trademarks

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

## Contact address

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

## Manual versions

This manual describes the latest software version. If any error occurs, please inform us and we will try to assist you as soon as possible.

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

Manual version	Date	By	Explanation
3.50 Rev. 1	061025	SK	Various layout and content improvements. Chapter "About this document" added. Changed chapter "Setup" and moved into chapter "Introduction". Chapter "Debugging with GDB" revised. Subchapter "GDB extensions" added.
3.30 Rev. 1	060703	OO	Subchapter "Supported remote commands": Added reset type listing.
3.23 Rev. 1	060526	TQ	Minor improvements.
3.21 Rev. 1	060512	TQ	Several corrections in chapter "Using DIGI evalboards" on page 29.
1.00 Rev. 1	060407	OO	Initial manual version.

## Software versions

Software version	Date	By	Explanation
3.30g	060703	OO	Several improvements.
3.23a	060526	TQ	Several improvements.
3.21b	060512	TQ	Dialog based user interface added.
1.00	060407	TQ	Initial software version.



# About this document

---

## Assumptions

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

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

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

## How to use this manual

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

## Typographic conventions for syntax

This manual uses the following typographic conventions:

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

**Table 1.1: Typographic conventions**



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

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

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet efficient real-time kernel. emWin, written entirely in ANSI C, can easily be used on any CPU and most any display. It is complemented by the available PC tools: Bitmap Converter, Font Converter, Simulator and Viewer. embOS supports most 8/16/32-bit CPUs. Its small memory footprint makes it suitable for single-chip applications.

Apart from its main focus on software tools, SEGGER develops and produces programming tools for flash microcontrollers, as well as J-Link, a JTAG emulator to assist in development, debugging and production, which has rapidly become the industry standard for debug access to ARM cores.

**Corporate Office:**

<http://www.segger.com>

**United States Office:**

<http://www.segger-us.com>

## EMBEDDED SOFTWARE (Middleware)



**emWin**

**Graphics software and GUI**

emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display. Starterkits, eval- and trial-versions are available.



**embOS**

**Real Time Operating System**

embOS is an RTOS designed to offer the benefits of a complete multitasking system for hard real time applications with minimal resources. The profiling PC tool embOSView is included.



**emFile**

**File system**

emFile is an embedded file system with FAT12, FAT16 and FAT32 support. emFile has been optimized for minimum memory consumption in RAM and ROM while maintaining high speed. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.



**emUSB**

**USB device stack**

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

## SEGGER TOOLS

**Flasher**

**Flash programmer**

Flash Programming tool primarily for microcontrollers.

**J-Link**

**JTAG emulator for ARM cores**

USB driven JTAG interface for ARM cores.

**J-Trace**

**JTAG emulator with trace**

USB driven JTAG interface for ARM cores with Trace memory. supporting the ARM ETM (Embedded Trace Macrocell).

**J-Link / J-Trace Related Software**

Add-on software to be used with SEGGER's industry standard JTAG emulator, this includes flash programming software and flash breakpoints.



# Table of Contents

1	Introduction .....	9
1.1	GDB / GDB Server overview .....	10
1.2	Hardware requirements .....	10
1.3	Setup.....	10
2	Debugging with GDB .....	11
2.1	Starting the J-Link GDB Server.....	12
2.1.1	User interface .....	12
2.2	Setting up the J-Link GDB Server.....	13
2.3	Setting up GDB .....	14
2.3.1	Generell GDB startup sequence .....	14
2.3.2	The .gdbinit file .....	14
2.3.3	Running GDB .....	15
2.4	Supported remote commands.....	16
2.4.1	ClrBP .....	17
2.4.2	Endian .....	17
2.4.3	Go .....	17
2.4.4	Halt .....	17
2.4.5	JTAGConf .....	18
2.4.6	Long .....	18
2.4.7	Reg .....	18
2.4.8	Reset .....	19
2.4.9	Select .....	20
2.4.10	SetBP.....	21
2.4.11	Sleep .....	21
2.4.12	Speed .....	21
2.4.13	Step .....	22
2.4.14	WaitHalt.....	22
2.4.15	WIce.....	22
2.5	Running GDB extensions (Insight, Eclipse, etc.).....	23
2.5.1	Insight .....	24
2.5.2	Eclipse .....	28
3	Using DIGI evalboards.....	29
3.1	Initial Steps .....	30
3.1.1	Copying .gdbinit files.....	30
3.2	Compiling the board support package (BSP).....	31
3.3	Compiling the sample application .....	32
3.4	Setting up the GDB configuration file.....	32
3.5	Debugging the sample application.....	33
4	Support .....	35
4.1	Troubleshooting .....	36
4.1.1	General procedure .....	36
4.1.2	Typical problem scenarios.....	36
4.2	Contacting support .....	37
4.3	FAQ .....	37
5	Glossary.....	39

6 Literature and references .....	41
-----------------------------------	----



# Chapter 1

## Introduction

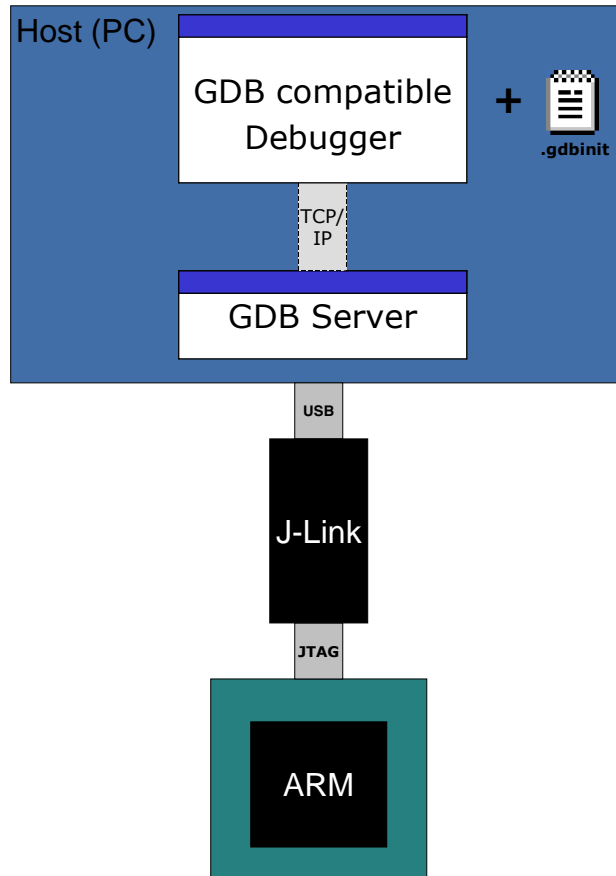
---

This chapter gives a short overview about how to start debugging your hardware with the GDB and the J-Link GDB Server.

## 1.1 GDB / GDB Server overview

The GNU Project Debugger (GDB) is a freely available debugger, distributed under the terms of the GPL. It connects to an emulator via a TCP/IP connection. It can connect to every emulator for which a GDB Server software is available. The latest Unix version of the GDB is freely available from the GNU committee under:

<http://www.gnu.org/software/gdb/download/>



GDB Server is a remote server for GDB. When you run GDB in the GDB source directory, it will read a `.gdbinit` file. The GDB `.gdbinit` file contains default setting informations and additional monitor commands. GDB and GDB Server communicate via a TCP/IP connection, using the standard GDB remote serial protocol. The GDB Server translates the GDB monitor commands into J-Link commands.

## 1.2 Hardware requirements

To use J-Link GDB Server, you have to meet the following hardware requirements:

- PC running Win2K / XP
- USB port
- J-Link / J-Trace

## 1.3 Setup

The J-Link setup procedure required in order to work with the J-Link GDB Server is described in chapter 2 of the *J-Link User's Guide*. The *J-Link User's Guide* is part of the J-Link software package which is available for download under [www.segger.com](http://www.segger.com).

# Chapter 2

## Debugging with GDB

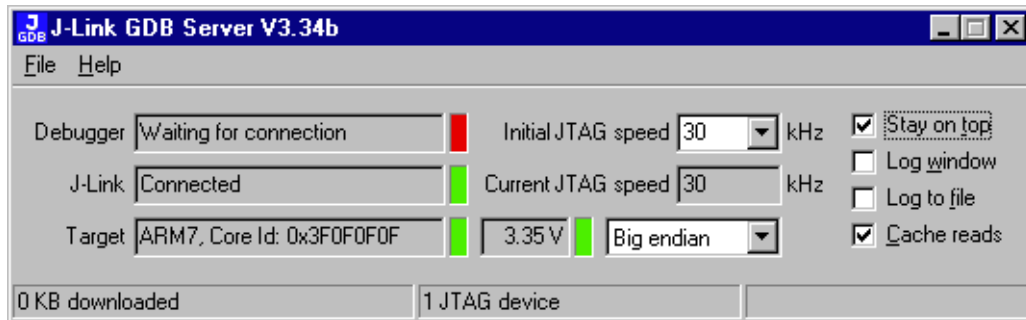
---

This chapter describes the setup procedure required in order to use the GDB with the J-Link GDB Server.

## 2.1 Starting the J-Link GDB Server

Start the J-Link GDB Server by double-clicking the executable file. Connect a J-Link to the host system, as described in chapter *Installing the USB driver* on page 10.

If a J-Link and target system is connected, the J-Link GDB Server should look similar to the screenshot below.



### 2.1.1 User interface

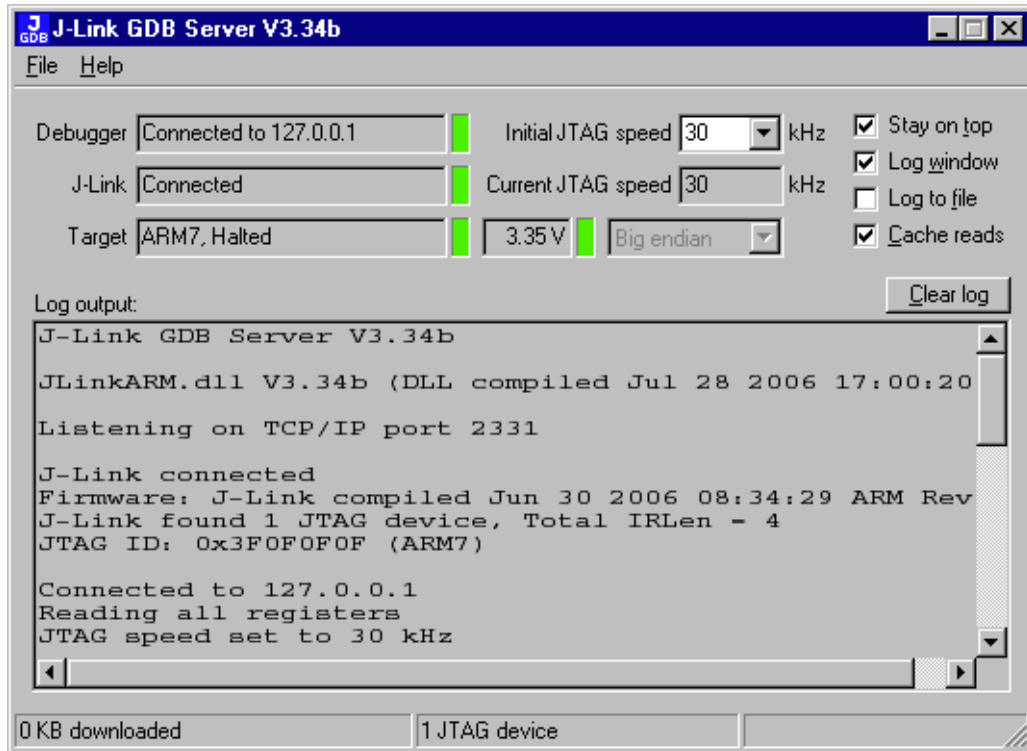
The J-Link GDB Server's user interface shows information about the debugging process and the target connected via JTAG.

It shows:

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

## 2.2 Setting up the J-Link GDB Server

Typically, most of the GDB setup is done from GDB via remote commands (monitor) in the `.gdbinit` file. However it is also possible to do the setup manually via user interface.



Via the **Initial JTAG speed** dropdown box the JTAG speed can be selected and with the box below the endianness of the target can be set.

These two boxes will be grayed out while debugging, although their values can be changed from the debugger console using remote commands.

### Stay on top

Allows you to force a window to "stay on top" of the other windows.

### Log window

The log window is only visible if the checkbox **Log window** is selected. The **Log output** window shows all commands which the GDB sends to the GDBServer. The **Log output** window is primarily useful for troubleshooting.

### Log file

If the **Log to file** checkbox is selected, the GDBServer generates the log file `C:\JLinkGDB.log`.

### Cache reads

Enables a memory read-ahead optimization which can speed up debugging.

## 2.3 Setting up GDB

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

### 2.3.1 Generell GDB startup sequence

1. Sets up the command interpreter as specified by the command line.
2. Reads the init file (if any) in your home directory and executes all the commands in that file.
3. Processes command line options and operands.
4. Reads and executes the commands from init file (if any) in the current working directory. This is only done if the current directory is different from your home directory.
5. Reads command files specified by the `-x` option.
6. Reads the command history recorded in the history file.

For more details about the GDB startup sequence refer to <http://www.gnu.org/software/gdb/documentation/>.

### 2.3.2 The .gdbinit file

When you run the GDB an initialization file, called `.gdbinit`, is searched in the GDB home directory. If the GDB finds a `.gdbinit` file, GDB executes all the commands in that file.

It is a good approach to store the setup informations for the remote debugging session in the `.gdbinit` file. Some sample files are supplied in the `GDBInit` folder of the GDB Server installation directory. Choose the sample that best fits to your target board, customize it and copy it into your GDB source directory.

You can use the `.gdbinit_template` as base for the implementation of new hardware.

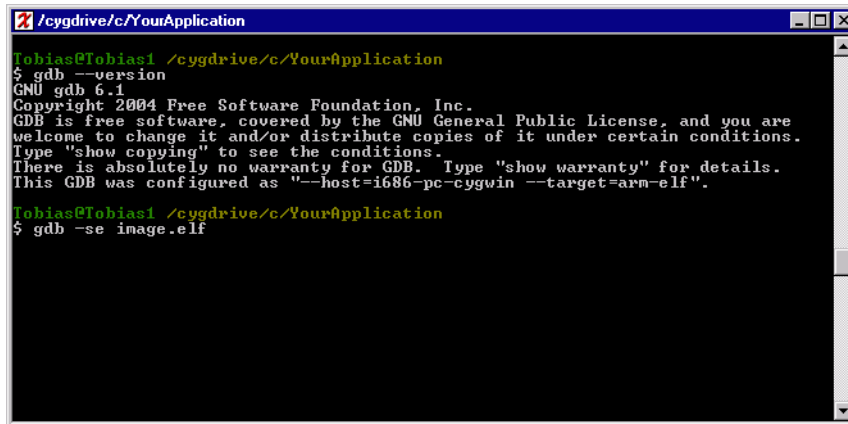
```
#
# J-LINK GDB SERVER initialization
#
# This connects to a GDB Server listening
# for commands on localhost at tcp port 2331
target remote localhost:2331
# Set JTAG speed to 30 kHz
monitor speed 30
# Set GDBServer to big endian
monitor endian big
# Reset the chip to get to a known state.
monitor reset

#
# CPU core initialization (to be done by user)
#

# Set the processor mode
monitor reg cpsr = 0xd3
# Set auto JTAG speed
monitor speed auto
# Setup GDB FOR FASTER DOWNLOADS
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
# Load the program executable called "image.elf"
# load image.elf
```

## 2.3.3 Running GDB

To start GDB enter `gdb -se <NameOfYourProgram>` in the console window. The option `-se` followed by a file name specifies the file which is used as symbol file and executable file for the debug session. GDB tries to load a `.gdbinit` file and executes all commands in that file.



```

/cygdrive/c/YourApplication
Tobias@Tobias1 /cygdrive/c/YourApplication
$ gdb --version
GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-elf".
Tobias@Tobias1 /cygdrive/c/YourApplication
$ gdb -se image.elf

```

We advise to use our supplied `.gdbinit` files, if one that fits to your hardware is available. The supplied `.gdbinit` files initializes the connection to J-Link GDB Server with the default settings (J-Link GDB Server running on localhost (127.0.0.1), listening on port 2331), initializes the core and downloads the specified executable. The last command in the supplied `.gdbinit` files, is the command to download your program to the target.

After the download process has finished, you must start program execution with `continue` rather than `run`, as the program is already started.

You can stop the program by pressing `control + c`. A list of debugger commands can be found by using the console command `help`.

## 2.4 Supported remote commands

J-Link GDB Server supports several remote commands from the GDB. This commands can be used from within a `.gdbinit` file or the GDB console to initialize the target board and to setup J-Link GDB Server specific parameters.

Remote command	Explanation
<code>ClrBP</code>	Removes an instruction breakpoint.
<code>Endian</code>	Sets endianness of the target.
<code>Go</code>	Starts the target CPU.
<code>Halt</code>	Halts the target CPU.
<code>JTAGConf</code>	Configures a JTAG scan chain with multiple devices on it.
<code>Long</code>	Reads or writes from/to given address.
<code>Reg</code>	Reads or writes from/to given register.
<code>Reset</code>	Resets and halts the target CPU.
<code>Select</code>	Selects the way J-Link is connected to host system.
<code>SetBP</code>	Sets an instruction breakpoint at a given address.
<code>Sleep</code>	Sleeps for a given time period.
<code>Speed</code>	Sets the JTAG speed of J-Link / J-Trace.
<code>Step</code>	Performs one or more single instruction steps.
<code>WaitHalt</code>	Waits for target to halt code execution.
<code>WIce</code>	Writes to given IceBreaker register.

**Table 2.1: GDB remote commands**

GDB sends the remote commands to the GDB Server. Remote command are what follows the GDB command `monitor` on the same line. If for example you want to start the target CPU, you have to either enter `monitor go` in the GDB console window or include this line in the `.gdbinit` file.



## 2.4.1 ClrBP

### Syntax

```
monitor ClrBP [<BPHandle>]  
or  
monitor ci [<BPHandle>]
```

### Description

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

### Example

```
monitor ClrBP 1
```

## 2.4.2 Endian

### Syntax

```
monitor endian <endian>
```

### Description

Sets endianness of target, where <endian> can either be big or little.

### Example

```
monitor endian big
```

## 2.4.3 Go

### Syntax

```
monitor go
```

### Description

Starts the target CPU.

### Example

```
monitor go
```

## 2.4.4 Halt

### Syntax

```
monitor halt
```

### Description

Halts the target CPU.

### Example

```
monitor halt
```

## 2.4.5 JTAGConf

### Syntax

```
monitor JTAGConf <IRPre> <DRPre>
```

### Description

Configures a JTAG scan chain with multiple devices on it. <IRPre> is the sum of IRLens of all devices closer to TDI, where IRLen is the number of bits in the IR (Instruction Register) of one device. <DRPre> is the number of devices closer to TDI. For more detailed information of how to configure a scan chain with multiple devices please refer to the *J-Link ARM User's Guide*.

### Example

```
monitor JTAGConf 4 1
```

## 2.4.6 Long

### Syntax

```
monitor long <address> [= <value>]
```

### Description

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

### Example

```
monitor long 0x50000000
monitor long 0x50000000 = 0xFFFF
```

## 2.4.7 Reg

### Syntax

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

### Description

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

### Add. information

Please note that in the current version of J-Link GDB Server only reading and writing of the `cpsr` and the `pc` register is supported.

### Example

```
monitor reg pc    = 0x00
monitor reg cpsr = 0x1F
```

## 2.4.8 Reset

### Syntax

```
monitor reset [<ResetType>]
```

### Description

Resets and halts the target CPU using the given reset type. If no reset type is specified, the reset type 0 ("Normal") will be used.

### Add. information

The following reset types are available:

Reset type	Explanation
0	<p>Normal (default if no reset type is specified)</p> <p>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.</p> <p>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.</p>
1	<p>Breakpoint @0</p> <p>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:</p> <ul style="list-style-type: none"> <li>a) 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.</li> <li>b) Some MCUs contain a bootloader program (sometimes called kernel), which needs to be executed to enable JTAG access.</li> </ul>
2	<p>Analog Devices</p> <p>The following sequence is executed:</p> <ul style="list-style-type: none"> <li>- The CPU is halted</li> <li>- A soft reset sequence is downloaded to RAM</li> <li>- A breakpoint at 0 is set</li> <li>- The soft reset sequence is executed</li> </ul> <p>This sequence performs a reset of CPU and peripherals and halts the CPU before executing instructions of the user program. It is recommended reset sequence for Analog Devices ADuC7xxx MCUs and works with these chips only.</p>
3	No reset is performed. Nothing happens.

**Table 2.2:**

Reset type	Explanation
4	<p>Halt @watchpoint</p> <p>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.</p> <p>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.</p>
5	<p>Halt @DBGREQ</p> <p>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.</p> <p>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.</p>
6	<p>Software reset.</p> <p>Sets the CPU registers to their after-Reset values:</p> <p>PC = 0</p> <p>CPSR = 0xD3 (Supervisor mode, ARM, IRQ / FIQ disabled)</p> <p>All SPSR registers = 0x10</p> <p>All other registers (which are unpredictable after reset) are set to 0.</p> <p>The hardware RESET pin is not affected.</p>

**Table 2.2:****Example**

```
monitor reset
monitor reset 1
```

## 2.4.9 Select

**Syntax**

```
monitor select USB
or
monitor select IP = <hostname>
```

**Description**

Selects the way J-Link / J-Trace is connected to the host system.

**Example**

```
monitor select USB
monitor select IP = localhost
```

## 2.4.10 SetBP

### Syntax

```
monitor SetBP <Addr> [<Mask>]
or
monitor bi <Addr> [<Mask>]
```

### Description

Sets an instruction breakpoint at the given address, where <Mask> is the address mask to be used. If no mask is specified a default mask of 0x03 is used (matches for breakpoints on ARM instructions). For breakpoints on THUMB instructions a mask of 0x01 should be specified.

### Example

```
monitor SetBP 0x00
monitor SetBP 0x100 0x01
```

## 2.4.11 Sleep

### Syntax

```
monitor Sleep <Delay>
```

### Description

Sleeps for a given time, where <Delay> is the time period in milliseconds to delay.

### Example

```
monitor Sleep 1000
```

## 2.4.12 Speed

### Syntax

```
monitor speed <kHz>
or
monitor speed auto
```

### Description

Sets the JTAG speed of J-Link / J-Trace.

### Example

```
monitor speed 1000
monitor speed auto
```

## 2.4.13 Step

### Syntax

```
monitor Step [<NumSteps>]  
or  
monitor si [<NumSteps>]
```

### Description

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

### Example

```
monitor Step 3
```

## 2.4.14 WaitHalt

### Syntax

```
monitor WaitHalt <Timeout>  
or  
monitor wh <Timeout>
```

### Description

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

### Example

```
monitor wh 2000
```

## 2.4.15 Wlce

### Syntax

```
monitor wlce <RegIndex> <value>  
or  
monitor rmib <RegIndex> <value>
```

### Description

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

### Example

```
monitor wlce 0x0C 0x100
```

## 2.5 Running GDB extensions (Insight, Eclipse, etc.)

There are many extensions and graphical user interfaces for GDB available. The range of products reaches from standalone implementations like GNU Insight (<http://sources.redhat.com/insight/>), frontends like DataDisplayDebugger (DDD) (<http://www.gnu.org/software/ddd/>) and IDEs like Eclipse (<http://www.eclipse.org>).

The J-Link GDB Server is tested with:

GDB version 6.1

Insight version 6.1

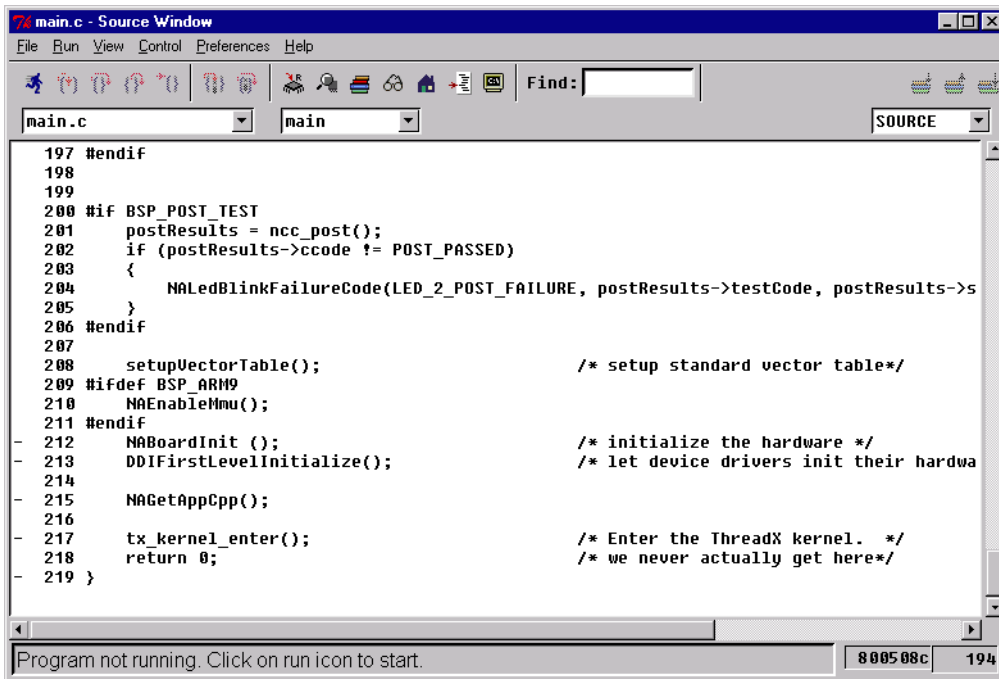
Eclipse version 3.2.0 and CDT version 3.1.0.

We apologize that you are familiar with all tools, which you use for the development of your application. The following information should only help to round out the context in which the J-Link GDB Server can be used.

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

## 2.5.1 Insight

Insight is a version of GDB that uses Tcl/Tk to implement a graphical user inter-face. It is a fully integrated GUI, not a separate front-end program.



Refer to <http://sources.redhat.com/insight/> for detailed information about Insight.

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



### 2.5.1.1 Start a debug session with Insight and J-Link GDB Server

You can start a debug session with Insight in the following way:

1. Start Insight
2. Open your program in Insight.
3. Connect to J-Link GDB Server
4. Download your program to your target
5. Run and debug your program

#### Start Insight

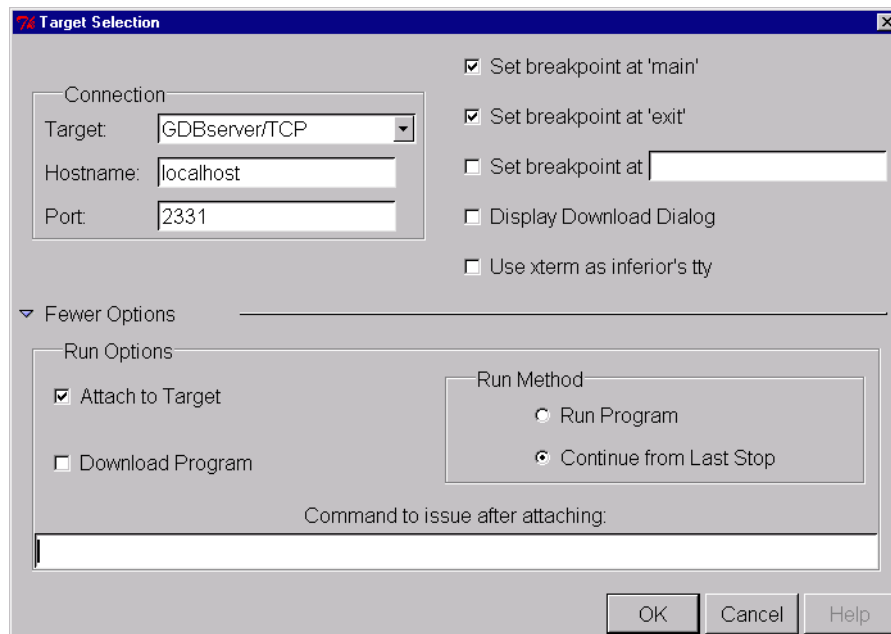
Enter `gdbtk` in your console window to start Insight.

#### Open your program in Insight.

To open your program in Insight, choose **File | Open** and select the executable that you want to debug.

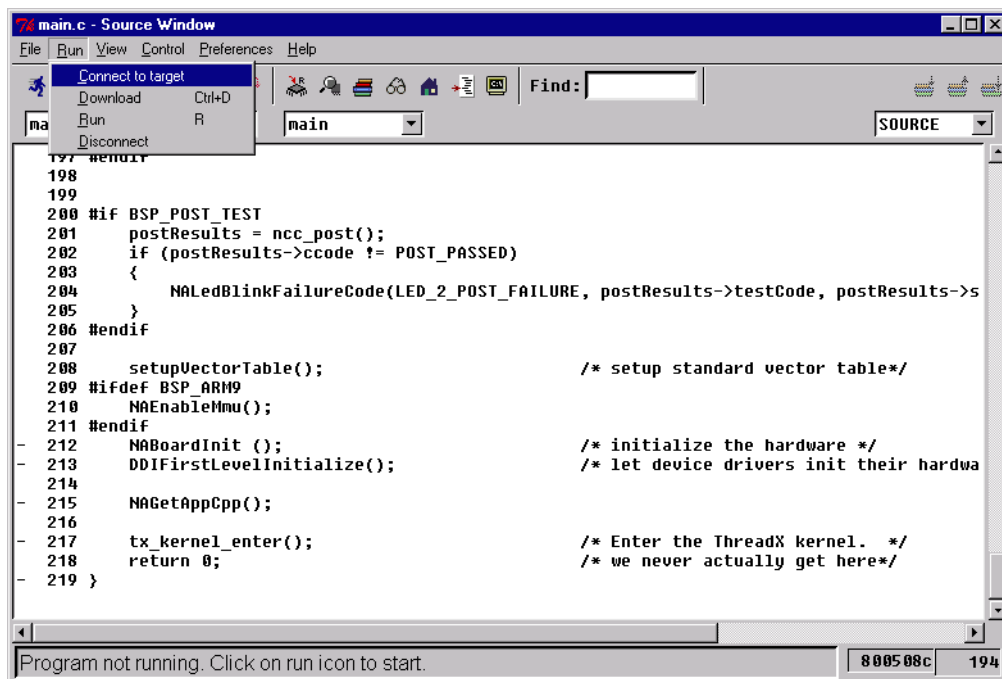
#### Connect to J-Link GDB Server

Set the settings for the connection to J-Link GDB Server. Open the **Target selection** dialog (File | Target settings) and select GDBServer/TCP in the **Target** choice list. Enter the IP address of the host which runs the J-Link GDB Server, for example local-host (127.0.0.1) and select the port the server listens for connections. By default, the server listens on port 2331.

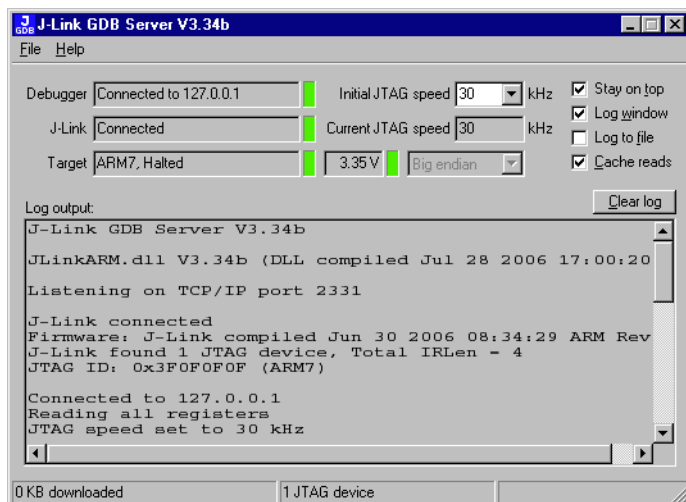


You can insert additional commands, for example for the initialization of your target core, in the textfield **Commands to issue after attaching** of the **Target Selection** dialog.

Choose **Run | Connect to target** to connect to your target via the J-Link GDB Server.



Check the status of the connection to J-Link GDB Server. The **Debugger** status message has changed to **Connected** to 127.0.0.1 and the status field aside has changed from red to green.



## Download your program to your target

To download your program, choose **Run | Download**. Afterwards, you can run and debug your program on your target hardware.

```

INIT.s - Source Window
File Run View Control Preferences Help

INIT.s Reset_Handler ASSEMBLY

- 0x800a92c <Reset_Handler_ROM>:      mov     r0, #19 ; 0x13
- 0x800a930 <Reset_Handler_ROM+4>:    orr     r0, r0, #128 ; 0x80
- 0x800a934 <Reset_Handler_ROM+8>:    orr     r0, r0, #64 ; 0x40
- 0x800a938 <Reset_Handler_ROM+12>:   msr     CPSR_Fc, r0
- 0x800a93c <Reset_Handler_ROM+16>:   ldr     r0, [pc, #708] ; 0x800ac08 <START+8>
- 0x800a940 <Reset_Handler_ROM+20>:   ldr     r0, [r0]
- 0x800a944 <Reset_Handler_ROM+24>:   and     r0, r0, #-16777216 ; 0xff000000
- 0x800a948 <Reset_Handler_ROM+28>:   mov     r0, r0, lsr #24
- 0x800a94c <Reset_Handler_ROM+32>:   cmp     r0, #40 ; 0x28
- 0x800a950 <Reset_Handler_ROM+36>:   bge     0x800a994 <done_p11>
- 0x800a954 <Reset_Handler_ROM+40>:   ldr     r0, [pc, #684] ; 0x800ac08 <START+8>
- 0x800a958 <Reset_Handler_ROM+44>:   ldr     r0, [r0]
- 0x800a95c <Reset_Handler_ROM+48>:   and     r0, r0, #-16777216 ; 0xff000000
- 0x800a960 <Reset_Handler_ROM+52>:   mov     r0, r0, lsr #22
- 0x800a964 <Reset_Handler_ROM+56>:   ldr     r1, [pc, #672] ; 0x800ac0c <START+12>
- 0x800a968 <Reset_Handler_ROM+60>:   add     r0, r0, r1
- 0x800a96c <Reset_Handler_ROM+64>:   ldr     r0, [r0]
- 0x800a970 <Reset_Handler_ROM+68>:   ldr     r1, [pc, #664] ; 0x800ac10 <START+16>
- 0x800a974 <Reset_Handler_ROM+72>:   str     r0, [r1]
- 0x800a978 <Reset_Handler_ROM+76>:   ldr     r5, [pc, #660] ; 0x800ac14 <START+20>
- 0x800a97c <Reset_Handler_ROM+80>:   ldr     r6, [r5]
- 0x800a980 <Reset_Handler_ROM+84>:   orr     r4, r6, #1073741824 ; 0x40000000
- 0x800a984 <Reset_Handler_ROM+88>:   str     r4, [r5]
- 0x800a988 <Reset_Handler_ROM+92>:   mov     r0, #16384 ; 0x4000

Program stopped at 0x800a94c      800a94c 262

```

## Using .gdbinit files

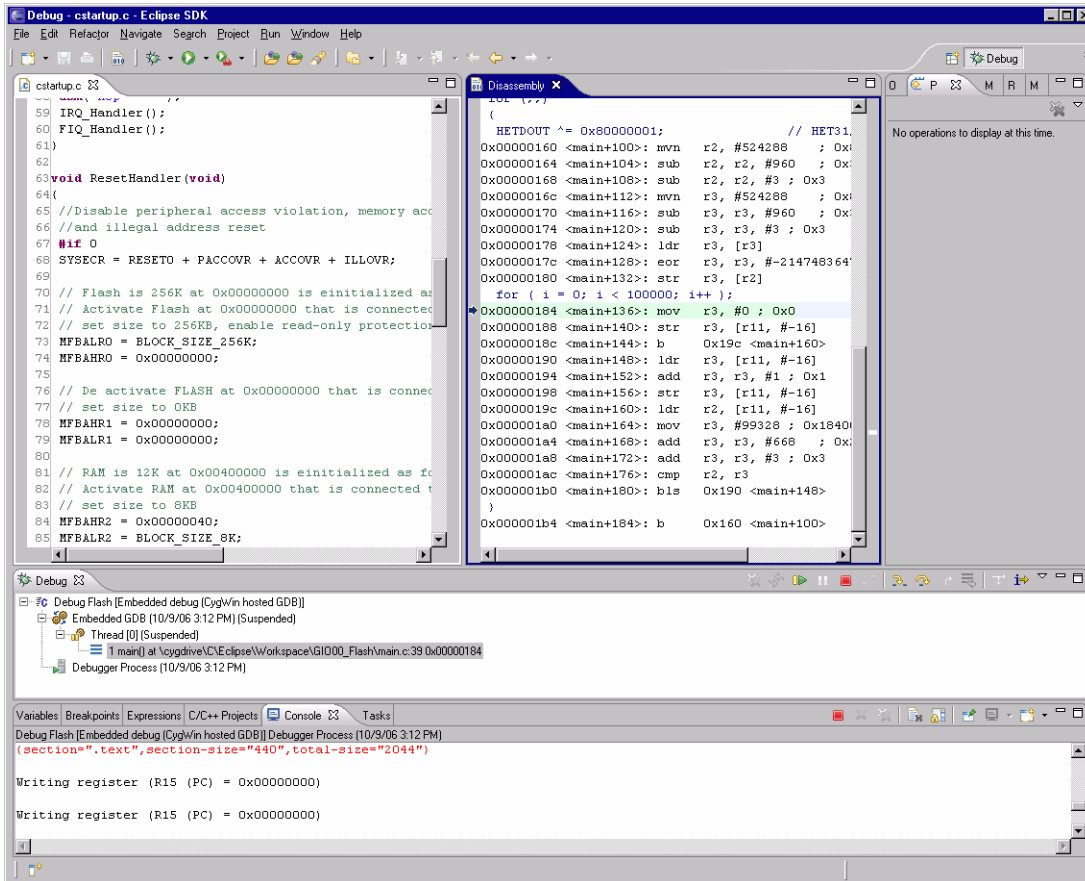
If you use one of our supplied .gdbinit files, you can abbreviate this sequence. The .gdbinit files initialize the connection to J-Link GDB Server with the default settings (J-Link GDB Server running on Localhost, listening on port 2331), initialize the core and download the specified executable. We advise to use our supplied .gdbinit files. To start Insight with a supplied .gdbinit file, choose the .gdbinit file that is suitable for your target hardware, copy it to your project folder and start Insight from your project folder with `gdbtk -se <NameOfYourExecutable>` and start debugging.

**Note:** By default, GDB will send relatively small memory write packets during download and reduces so the possible download speed. In the supplied .gdbinit files is the memory-write-packet-size enlarged to 1024 bytes. This enlargement helps to increase the download speed and should not lead to problems with your target hardware. Therefore, ignore the warning from the GDB and select **Yes** in the dialog.

## 2.5.2 Eclipse

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

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



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

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

# Chapter 3

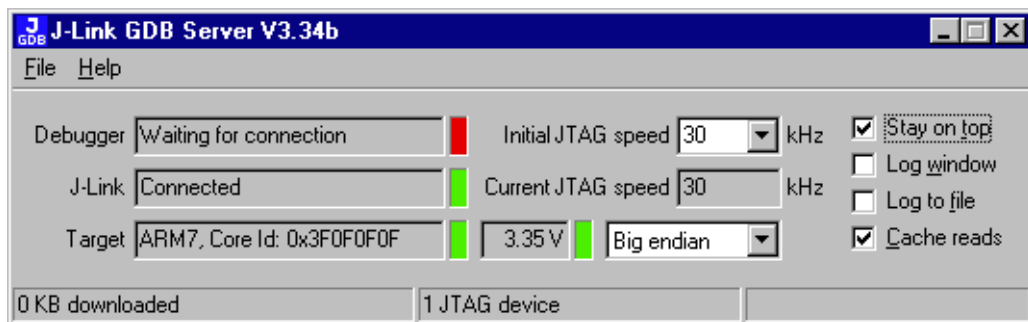
## Using DIGI evalboards

---

This chapter describes the setup procedure required in order to debug DIGI boards with the GDB and the J-Link GDB Server. This includes primarily the compilation routines and configuration hints for the DIGI sample applications. In this case we will refer to the DIGI Connect ME hardware.

## 3.1 Initial Steps

First of all, please start the J-Link GDB Server by double-clicking the executable file. You will see the J-Link GDB Server:



### 3.1.1 Copying .gdbinit files

To make things easy, the J-Link GDB Server package contains ready-to-go \*.gdbinit files for the various DIGI boards.

In order to use these .gdbinit files with the DIGI boards and NET+Works GNU Software, please copy the \*.jlink files found in the GDBInit folder to %NETOS-DIR%\debugger\_files, which is per default C:\netos63\_gnu\debugger\_files\.

## 3.2 Compiling the board support package (BSP)

After starting the Net+Works 6.3 Build Environment, you will find yourself in a Unix like shell environment with a command prompt.

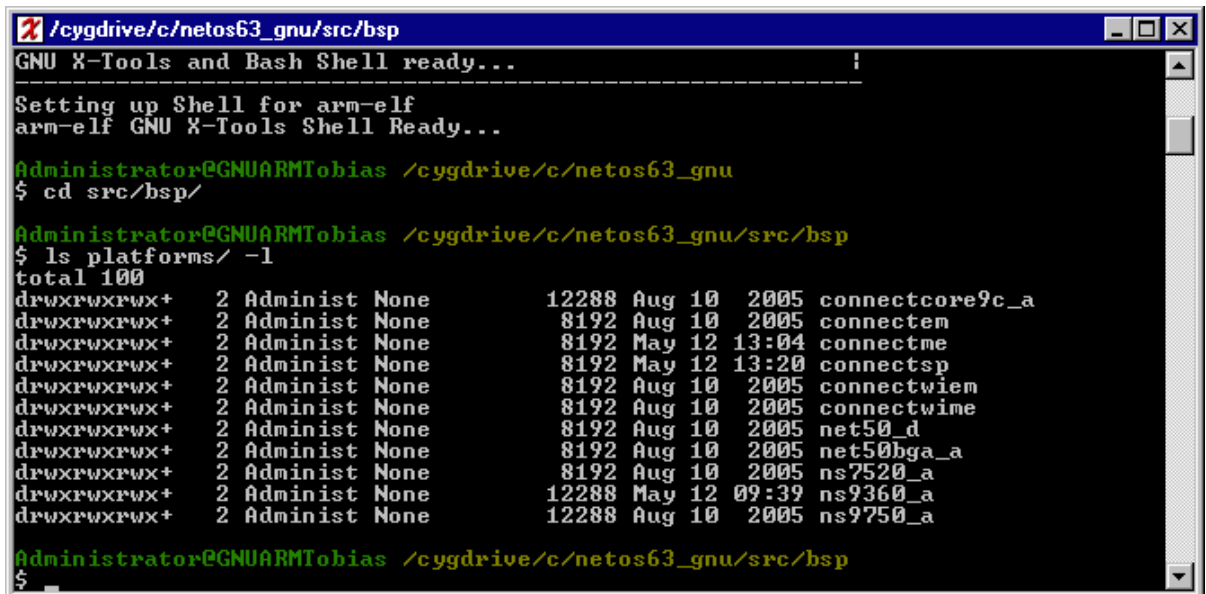
To compile the board support package (BSP), change to the bsp directory, which is located under `/cygdrive/c/netos63_gnu/src/bsp` by using the `cd` command which stands for "change directory".

```
cd src/bsp
```

To list the available BSPs, please use the following command:

```
ls platforms/ -l
```

You will see a list of available platforms as shown in the screenshot below:



```

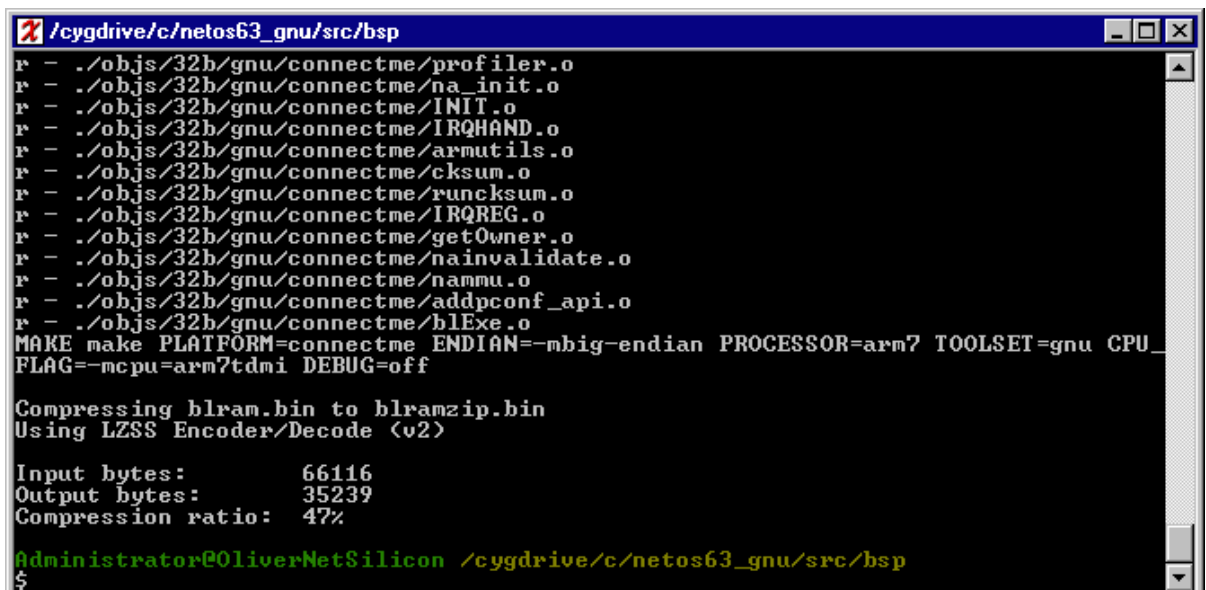
/cygdrive/c/netos63_gnu/src/bsp
GNU X-Tools and Bash Shell ready...
Setting up Shell for arm-elf
arm-elf GNU X-Tools Shell Ready...
Administrator@GNUARMTObias /cygdrive/c/netos63_gnu
$ cd src/bsp/
Administrator@GNUARMTObias /cygdrive/c/netos63_gnu/src/bsp
$ ls platforms/ -l
total 100
drwxrwxrwx+ 2 Administ None      12288 Aug 10  2005 connectcore9c_a
drwxrwxrwx+ 2 Administ None      8192 Aug 10  2005 connectme
drwxrwxrwx+ 2 Administ None      8192 May 12 13:04 connectme
drwxrwxrwx+ 2 Administ None      8192 May 12 13:20 connectsp
drwxrwxrwx+ 2 Administ None      8192 Aug 10  2005 connectwiem
drwxrwxrwx+ 2 Administ None      8192 Aug 10  2005 connectwime
drwxrwxrwx+ 2 Administ None      8192 Aug 10  2005 net50_d
drwxrwxrwx+ 2 Administ None      8192 Aug 10  2005 net50bga_a
drwxrwxrwx+ 2 Administ None      8192 Aug 10  2005 ns7520_a
drwxrwxrwx+ 2 Administ None     12288 May 12 09:39 ns9360_a
drwxrwxrwx+ 2 Administ None     12288 Aug 10  2005 ns9750_a
Administrator@GNUARMTObias /cygdrive/c/netos63_gnu/src/bsp
$

```

Then compile the board support package (BSP) by using the `make` command. You will have to specify for which target you want to compile the BSP. This is done by the `PLATFORM` parameter and the `clean all` parameters to clean up the directories before building the BSP library.

```
make PLATFORM=connectme clean all
```

After the BSP is built, your screen should look similar to the screenshot below.



```

/cygdrive/c/netos63_gnu/src/bsp
r - ./objs/32b/gnu/connectme/profiler.o
r - ./objs/32b/gnu/connectme/na_init.o
r - ./objs/32b/gnu/connectme/INIT.o
r - ./objs/32b/gnu/connectme/IRQHAND.o
r - ./objs/32b/gnu/connectme/armutils.o
r - ./objs/32b/gnu/connectme/cksum.o
r - ./objs/32b/gnu/connectme/runcksum.o
r - ./objs/32b/gnu/connectme/IRQREG.o
r - ./objs/32b/gnu/connectme/getOwner.o
r - ./objs/32b/gnu/connectme/nainvalidate.o
r - ./objs/32b/gnu/connectme/nammu.o
r - ./objs/32b/gnu/connectme/addpconf_api.o
r - ./objs/32b/gnu/connectme/blExe.o
MAKE make PLATFORM=connectme ENDIAN=-mbig-endian PROCESSOR=arm7 TOOLSET=gnu CPU
FLAG=-mcpu=arm7tdmi DEBUG=off

Compressing blram.bin to blramzip.bin
Using LZSS Encoder/Decode (v2)

Input bytes:      66116
Output bytes:     35239
Compression ratio: 47%

Administrator@OliverNetSilicon /cygdrive/c/netos63_gnu/src/bsp
$

```

Now you can start to compile the sample application, which is described under *Compiling the sample application* on page 32.

### 3.3 Compiling the sample application

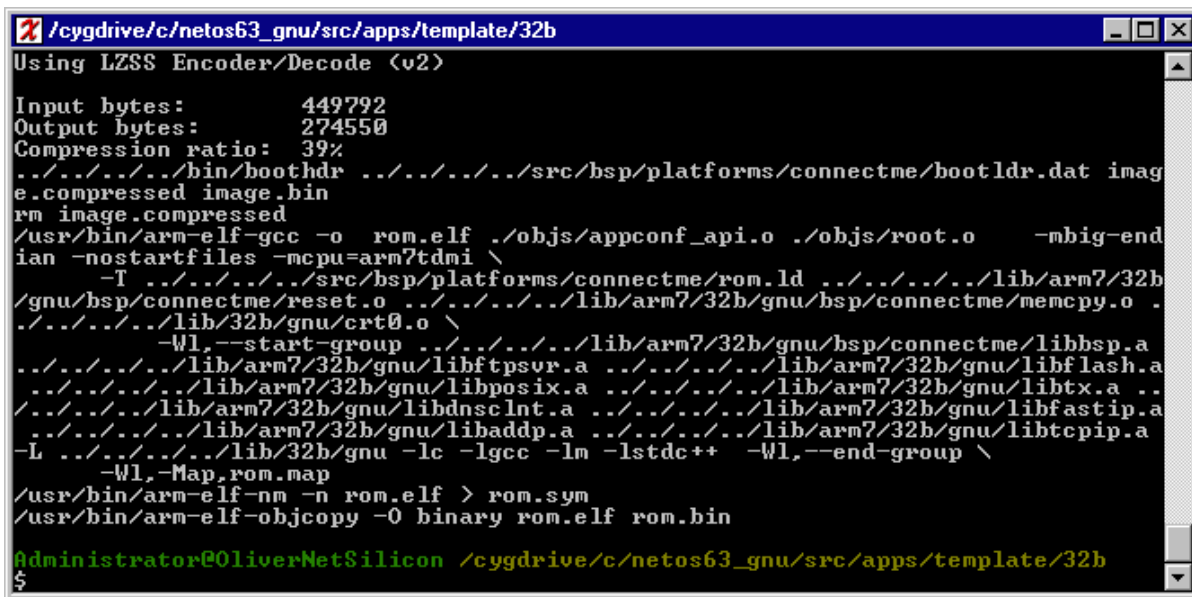
After building the BSP for your DIGI board, change to the `/cygdrive/c/netos63_gnu/src/apps/template/32b` directory by using the following command:

```
cd /cygdrive/c/netos63_gnu/src/apps/template/32b
```

Then compile the sample project by using the `make` command. You will have to specify for which target you want to compile the sample project. This is done by the `PLATFORM` parameter and the `clean all` parameters to clean up the directories before building the BSP library.

```
make PLATFORM=connectme clean all
```

After the sample application is built, your screen should look similar to the screenshot below.



```

/cygdrive/c/netos63_gnu/src/apps/template/32b
Using LZSS Encoder/Decode (v2)
Input bytes:      449792
Output bytes:     274550
Compression ratio: 39%
../../../../bin/boothdr ../../../../src/bsp/platforms/connectme/bootldr.dat image.compressed image.bin
rm image.compressed
/usr/bin/arm-elf-gcc -o rom.elf ./objs/appconf_api.o ./objs/root.o -mbig-endian -nostartfiles -mcpu=arm7tdmi \
-T ../../../../src/bsp/platforms/connectme/rom.ld ../../../../lib/arm7/32b/gnu/bsp/connectme/reset.o ../../../../lib/arm7/32b/gnu/bsp/connectme/memcpy.o \
../../../../lib/32b/gnu/crt0.o \
-Wl,--start-group ../../../../lib/arm7/32b/gnu/bsp/connectme/libbsp.a \
../../../../lib/arm7/32b/gnu/libftpsvr.a ../../../../lib/arm7/32b/gnu/libflash.a \
../../../../lib/arm7/32b/gnu/libposix.a ../../../../lib/arm7/32b/gnu/libtx.a \
../../../../lib/arm7/32b/gnu/libdnscnt.a ../../../../lib/arm7/32b/gnu/libfastip.a \
../../../../lib/arm7/32b/gnu/libadp.a ../../../../lib/arm7/32b/gnu/libtcpip.a \
-L ../../../../lib/32b/gnu -lc -lgcc -lm -lstdc++ -Wl,--end-group \
-Wl,-Map,rom.map
/usr/bin/arm-elf-nm -n rom.elf > rom.sym
/usr/bin/arm-elf-objcopy -O binary rom.elf rom.bin
Administrator@OliverNetSilicon /cygdrive/c/netos63_gnu/src/apps/template/32b
$

```

### 3.4 Setting up the GDB configuration file

The GDB will search for the `.gdbinit` file in the workspace folder, from where you start the GDB out of. For this, you need to copy the init file, corresponding to your board, from `/cygdrive/c/netos63_gnu/debugger_files` into your workspace folder. This can easily be done by the following two commands:

```
cd /cygdrive/c/netos63_gnu/src/apps/template/32b
```

```
cp ../../../../debugger_files/gdbconnectme.jlink ../gdbinit
```



## 3.5 Debugging the sample application

To debug your sample application, change your directory to `/cygdrive/c/netos63_gnu/src/apps/template/32b`.

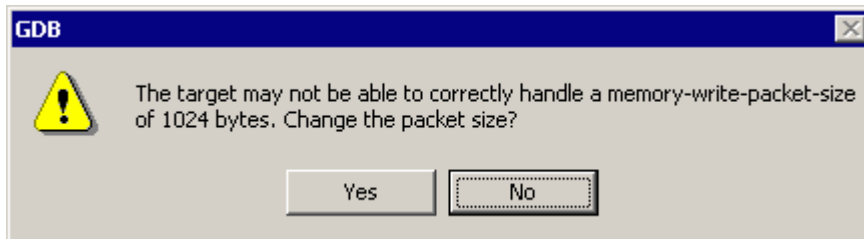
```
cd /cygdrive/c/netos63_gnu/src/apps/template/32b
```

Start the GNU Project Debugger (GDB) with the compiled sample project by typing

```
gdbtk -se image.elf
```

into the NET+Works 6.3 Build Environment.

The GDB will ask, if you would like to change the packet size. Select **Yes**.



After this, GDB starts to download the sample application into the target. When the download is finished, you can debug through the application.

For further information please refer to the GDB documentation, which is freely available from the GNU committee under:

<http://www.gnu.org/software/gdb/documentation/>



# Chapter 4

## Support

---

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.

Further instructions are explained in the J-Link / J-Trace User's Guide.

## 4.1 Troubleshooting

### 4.1.1 General procedure

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

1. Close all running applications on your host system.
2. Disconnect the J-Link / J-Trace device from USB.
3. Power-off target.
4. Re-connect J-Link / J-Trace with host system (attach USB cable).
5. Power-on target.
6. Try your target application again. If the problem vanished, you are done; otherwise continue.
7. Close all running applications on your host system again.
8. Disconnect the J-Link / J-Trace device from USB.
9. Power-off target.
10. Re-connect J-Link / J-Trace with host system (attach USB cable).
11. Power-on target.
12. Start `JLink.exe`.
13. If `JLink.exe` reports the J-Link / J-Trace serial number and the target processor's core ID, the J-Link / J-Trace is working properly and cannot be the cause of your problem.
14. If `JLink.exe` is unable to read the target processor's core ID you should analyze the communication between your target and J-Link / J-Trace with a logic analyzer or oscilloscope. Follow the instructions in section 9.2 in the J-Link / J-Trace User's Guide.
15. If your problem persists and you own an original product (not an OEM version), see section *Contacting support* on page 37.

### 4.1.2 Typical problem scenarios

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

Meaning:

The USB connection does not work.

Remedy:

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

#### **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 9.

## **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:

Please follow the steps described in section 9.1.1 in the J-Link / J-Trace User's Guide.

## **4.2 Contacting support**

Before contacting support, make sure you tried to solve your problem by following the steps outlined in section "General procedure" in the J-Link / J-Trace User's Guide. 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, please send the following information to [support@segger.com](mailto:support@segger.com):

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

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

## **4.3 FAQ**

Q: Which CPUs are supported?

A: Every CPU supported by J-Link / J-Trace is supported.



# Chapter 5

## Glossary

---

This chapter explains important terms used throughout this manual.

**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.

**GDB**

A GNU Project Debugger that is freely available.

**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.

**Image**

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

**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.

**Target**

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

**Watchpoint**

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



# Chapter 6

## Literature and references

---

This chapter lists documents, which we think may be useful to gain deeper understanding of technical details.

Reference	Title	Comments
[GDB]	GDB Documentation	This document describes the GDB Server usage in detail. It is publicly available from the GNU committee ( <a href="http://www.gnu.org">www.gnu.org</a> ).
[JUG]	J-Link / J-Trace User's Guide	This document describes the J-Link / J-Trace debug interfaces in detail. It is publicly available from SEGGER ( <a href="http://www.segger.com">www.segger.com</a> ).

**Table 6.1: Literature and reference**

# Index

---

<b>B</b>		Support .....	39
Big-endian .....	40	Syntax, conventions used .....	5
<b>C</b>		<b>T</b>	
Cache cleaning .....	40	Target .....	40
<b>G</b>		<b>W</b>	
GDB .....	40	Watchpoint .....	40
<b>H</b>			
Host .....	40		
<b>I</b>			
ICache .....	40		
Image .....	40		
<b>J</b>			
Joint Test Action Group (JTAG) .....	40		
<b>L</b>			
Little-endian .....	40		
<b>S</b>			
Server command			
ClrBP .....	17		
Endian .....	17		
Go .....	17		
Halt .....	17		
JTAGConf .....	18		
Long .....	18		
Reg .....	18		
Reset .....	19		
Select .....	20		
SetBP .....	21		
Sleep .....	21		
Speed .....	21		
Step .....	22		
WaitHalt .....	22		
WIce .....	22		
Server commands .....	16		

