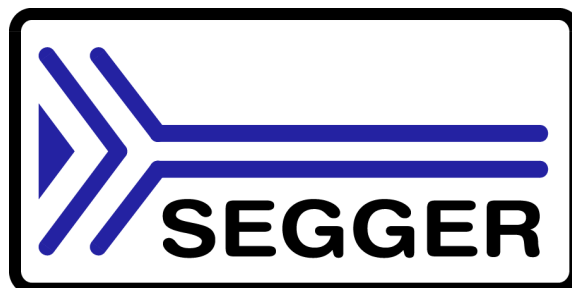# J-Link ARM GDB Server

User guide of the
J-Link ARM GDB Server

Software Version 4.12
Manual Rev. 1

Date: February 16, 2010

Document: UM08005

A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

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 GmbH & Co. KG (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.

© 2007 SEGGER Microcontroller GmbH & Co. KG, 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 GmbH & Co. KG
In den Weiden 11
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
4.12 Rev. 1	100216	AG	Chapter "Debugging with GDB" * Section "Command line options" updated.
4.12 Rev. 0	100112	KN	Chapter "Debugging with GDB" * "Command line options" updated Several corrections.

4.04 Rev. 0	090331	AG	Chapter "Debugging with GDB"
			  * Remote command "flash cpuclock" added.
			  * Remote command "interface" added.

3.91 Rev. 0	080821	AG	Chapter "Debugging with GDB" * "Debugging on Cortex-M3 devices" added. * "Supported remote commands" updated.
3.90 Rev. 0	080811	AG	Several corrections.
3.84 Rev. 1	080617	AG	Chapter "Debugging with GDB" * Remote command "AllowSimulation" added.
3.80 Rev. 2	080408	AG	Chapter "Licensing" added. Chapter "Intro" updated.

3.80 Rev. 1	080215	AG	Chapter "Debugging with GDB":
			  * Remote command "flash device" updated.
			  * Remote command "flash download" added.
			  * Remote command "flash breakpoints" added.

3.80 Rev. 0	080206	SK	Chapter "Debugging with GDB": * Remote command "flash device" added. Chapter "Flash download and Flash breakpoints" added.

Manual version	Date	By	Explanation
3.78 Rev. 1	071213	OO	General update.
3.64 Rev. 3	070308	SK	Commands WriteU8, WriteU16, WriteU32 added. Chapter "Debugging with GDB" * Subchapter "Yagarto" added. Chapter "Licensing" enhanced.
3.64 Rev. 2	070307	SK	Chapter "Licensing" added.
3.64 Rev. 1	070305	SK	Chapter "Command line options" added.
3.60 Rev. 1	070222	SK	Chapter "Debugging with GDB" Command "speed" updated.
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 47.
1.00 Rev. 1	060407	OO	Initial manual version.

Software versions

Software version	Date	By	Explanation
3.78a	071213	OO	Several improvements.
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.

4

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.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections

Table 1.1: Typographic conventions

SEGGER Microcontroller GmbH & Co. KG 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 efficent 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 developes 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

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 commitee under:
http://www.gnu.org/software/gdb/download/
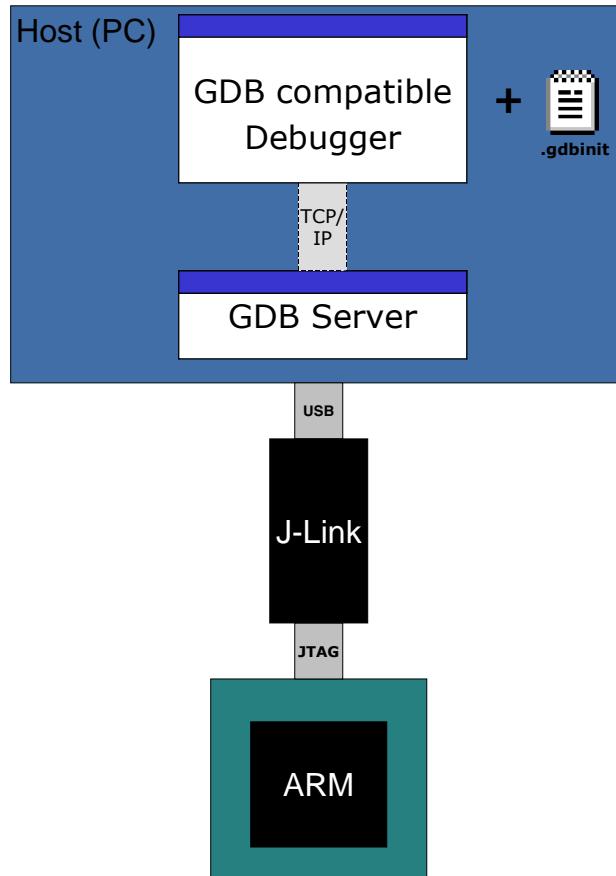
GDB Server is a remote server for GDB. When you run GDB in the GDB source directory, it will read a .gdb file. The GDB .gdb file contains default setting information and additional monitor commands. GDB and GDB Server communicate via a TCP/IP connection, using the standard GDB remote serial protocol. 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 / Vista
•   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*.

Chapter 2

Licensing

This chapter describes the license management of the software.

2.1 Introduction

J-Link GDB Server is distributed as "free for evaluation and non commercial use".

The software can be used free of charge for educational and nonprofit purposes without additional license. Without additional license, only 32 KBytes may be downloaded. To download bigger programs or to use the software for other, especially commercial purposes, a license is required. With such a license, the download size is not limited.

Free evaluation licenses are available upon request. To obtain an trial or unlimited license, please contact *sales@segger.com.*

Full and valid license terms are specified in file License.txt which comes with the J-Link software and documentation package.

SAM-ICE

J-Link GDB Server can be used free of charge without limitation of program size with SAM-ICE.

2.2 License types

For the J-Link GDB Server, three types of licenses are available, which will be explained in the following:

Built-in License

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

Key-based license

This type of license is used if you already have a J-Link, but want to enhance its functionality by using J-Link GDB Server. 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 from SEGGER. Free trial licenses are available upon request from *www.segger.com* This license key has to be added to the GDB Server license management. How to enter a license key is described in detail in section *Entering a license key* on page 13. 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 GDB Server with other J-Links, every J-Link needs a license.

"Free evaluation and non commercial use" license

This type of license comes with J-Link GDB Server and is a license for educational and nonprofit use of J-Link GDB Server. To use this license simply select **Start in free mode** when the license dialog is opened. In this "free mode" the maximum program size that can be downloaded with J-Link GDB Server is limited to 32 KBytes.

2.2.1 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.

2.2.2   Key-based license

When using a key-based license, a license key is required in order to unlock the full potential of J-Link GDB Server. License keys can be added via the J-Link GDB Server license management. To open the J-Link GDB Server license management simply select **Licenses...** from the Help menu in the main window. 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.

When using J-Link GDB Server and no license is found by the software, it will ask for a license key.

2.2.2.1   Entering a license key

The **J-Link GDB Server license management** dialog shows the available licenses and allows to add and remove licenses as well.
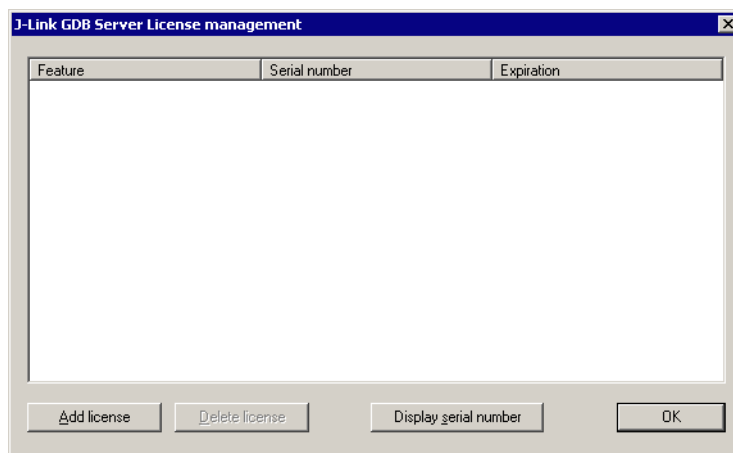

Select **Add license** to enter a license key or **Display serial number** to get the serial number of the connected J-Link. J-Link GDB server licenses are accorded on a one per J-Link basis. This means that a license key is only valid for the J-Link with the suitable serial number.

## 2.2.3   "Free evaluation and non commercial use" license

When using the "free evaluation and non commercial use" license of J-Link GDB Server, the program size which can be downloaded with the J-Link GDB Server is limited to 32 KBytes. To use this type of license J-Link GDB Server has to be started in "free mode". How to start J-Link GDB Server in "free mode" is explained in the following.

2.2.3.1   License dialog

A startup license dialog will be opened with every start of the J-Link GDB, if no additional license is installed.
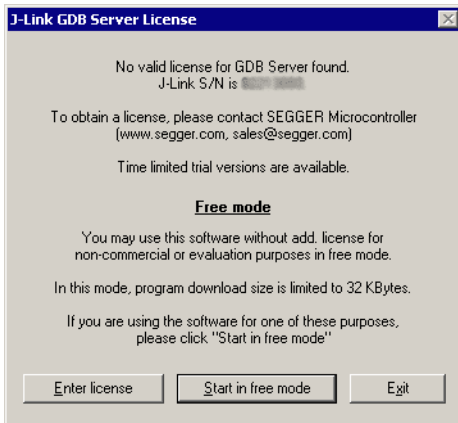
Select **Start in free mode** to start J-Link GDB Server with 32 Kbytes download limitation or select **Enter license** to remove the limitation and deactivate the startup dialog. If the J-Link GDB Server is started in free mode, a status message appears in the log window.
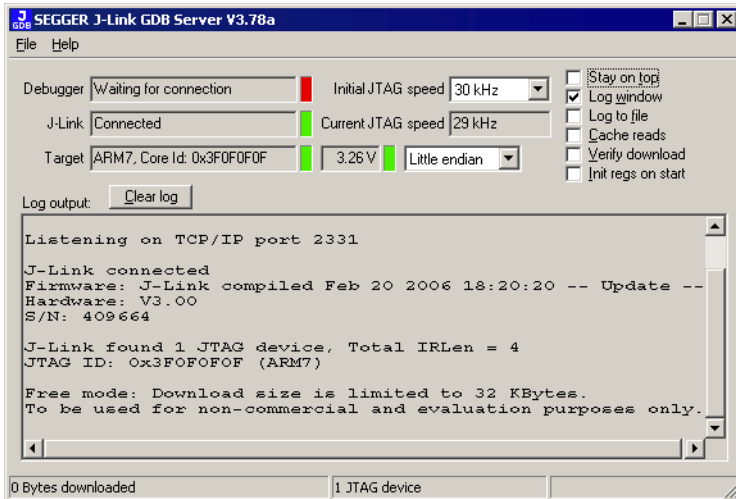
Chapter 3

Debugging with GDB

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

3.1   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.
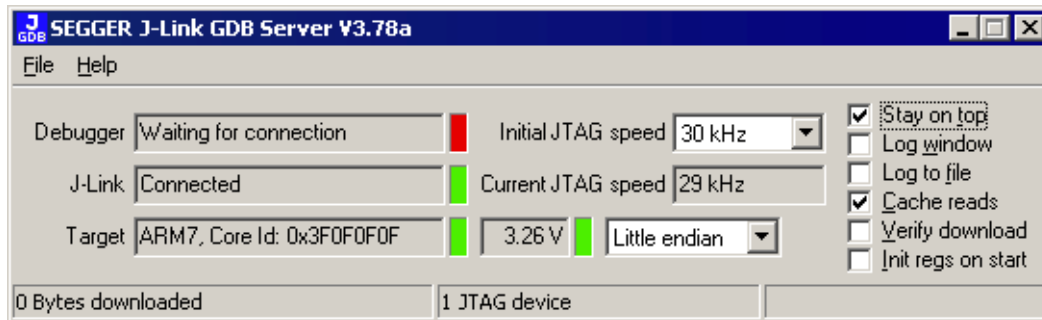

3.1.1   User interface

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 endianess.

3.2   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.
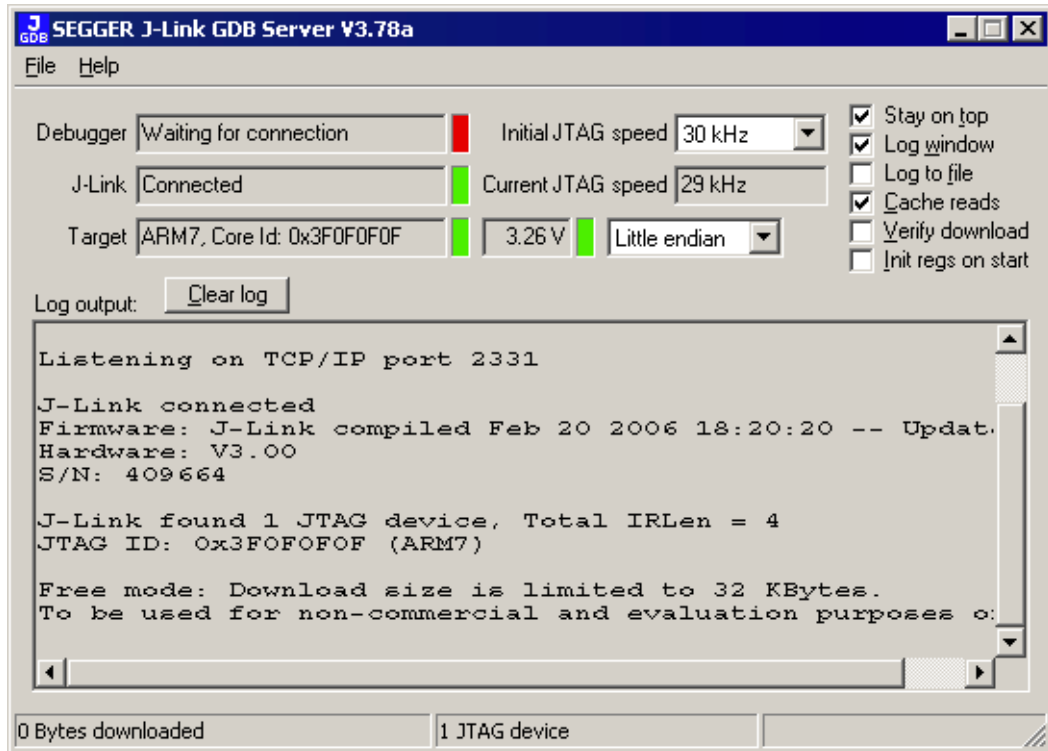
Via the **Initial JTAG speed** dropdown box the JTAG speed can be selected and with the box below the endianess of the target can be set.

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 to 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.

Verify download

Verifies the program in the target after a download.

Init regs on start

Initializes target registers with good start values.

3.3   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.

3.3.1   General 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/*.

3.3.2   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 information 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.

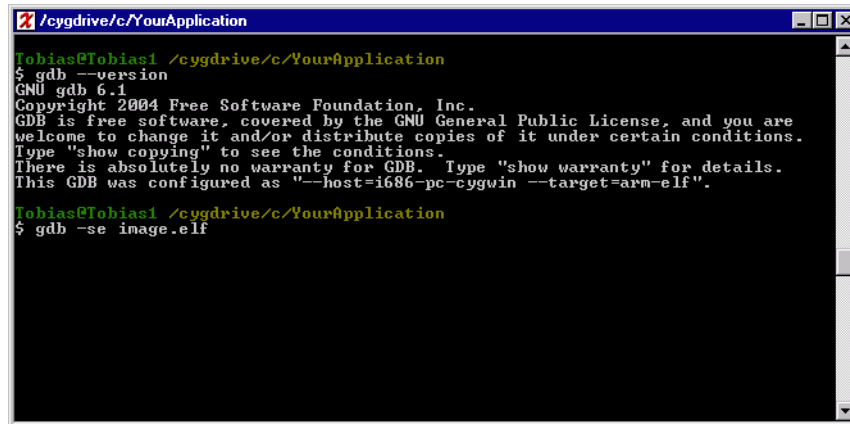You can use the .gdbinit_template as base for the implementation of new hardware.

```
#
# 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
```

3.3.3   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.

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`.

3.4   Debugging on Cortex-M3 devices

J-Link GDB Server supports debugging on Cortex-M3 devices. Both, debugging in RAM and flash are supported.
Flash breakpoints (FlashBP) are also available but require a separate license.

3.4.1   Debugging in RAM

When debugging in RAM on a Cortex-M3 device, the stack pointer (SP, R13) and program counter (PC, R15) are not automatically set to the start values by most debuggers.
In order to debug an application in RAM, please ensure that PC and SP have correct values before you start to debug the application.
Typically, the start of the vector is the start of the RAM area used for download. On most devices, this is 0x20000000. This means that the initial value of the stack pointer (SP) can be read from 0x20000000 and the initial value of the PC can be read from 0x20000004.
To ensure that the stack pointer and the PC are initialized correctly you can set them in the .gdbinit file as shown below.

Sample GDB init sequence

The following sample GDB init sequence should work on any Cortex-M3 device when debugging in RAM:

```
#*************************************************
#
#  Connect to J-Link and debug application in flash on Cortex-M3
#
#  Download to flash is performed.
#
# Connect to the J-Link gdb server
target remote localhost:2331
monitor speed 1000
load ST_MB525_RAM.elf
# Initializing PC and stack pointer
# RAM_START_ADDR is at 0x20000000
monitor reg r13 = (0x20000000)
monitor reg pc = (0x20000004)
```

3.4.2   Debugging in flash

When debugging in flash the stack pointer and the PC are set automatically when the target is reset after the flash download.

Without reset after download, the stack pointer and the PC need to be initialized correctly, typically in the .gdbinit file. The following sample GDB init sequence is for a STM32.

Sample GDB init sequence

```
#*************************************************
#
#  Connect to J-Link and debug application in flash on Cortex-M3
#
#  no download is performed.
#
# Connect to the J-Link gdb server
target remote localhost:2331
monitor speed 1000
# Can not load into flash if device is not specified. load ST_MB525_FLASH.elf
# Initializing PC and stack pointer
monitor reg r13 = (0x00000000)
monitor reg pc = (0x00000004)
```

3.4.2.1 Download to flash

To enable download to flash, the device needs to be selected and flash download has to be enabled. For more information, please refer to *Enabling flash download and flash breakpoints* on page 44

Sample GDB init sequence

```
#*************************************************
#
#  Connect to J-Link and debug application in flash on Cortex-M3
#
#  Download to flash is performed.
#
# Connect to the J-Link gdb server
target remote localhost:2331
monitor speed 1000
monitor flash device = STM32F103VB
monitor flash download = 1
load ST_MB525_FLASH.elf
# Initializing PC and stack pointer
monitor reg r13 = (0x00000000)
monitor reg pc = (0x00000004)
```

3.4.2.2 Flash breakpoints

Most Cortex-M3 devices offer 6 hardware breakpoints. Can be very convenient for debugging to have more than 6 breakpoints. Flash breakpoints are supported for most popular Cortex-M3 devices.

To enable flash breakpoints, the device needs to be selected and flash breakpoints have to be enabled. For more information, please refer to *Enabling flash download and flash breakpoints* on page 44.

```
#*************************************************
#
#  Connect to J-Link and debug application in flash on Cortex-M3
#
#  Download to flash is performed.
#
# Connect to the J-Link gdb server
target remote localhost:2331
monitor speed 1000
monitor flash device = STM32F103VB
monitor flash breakpoints = 1
monitor flash download = 1
load ST_MB525_FLASH.elf
# Initializing PC and stack pointer
monitor reg r13 = (0x00000000)
monitor reg pc = (0x00000004)
```

3.5   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
AllowSimulation	Enables/Disables ARM instruction set simulation.
clrbp	Removes an instruction breakpoint.
cp15	Reads or writes from/to cp15 register.
endian	Sets endianess of the target.
flash breakpoints	Enables/Disables flash breakpoints.
flash cpuclock	Sets the clock frequency the CPU is currently running with (Required for correct operation of some flash algorithms).
flash device	Selects the target's flash device.
flash download	Enables/Disables flash download.
go	Starts the target CPU.
halt	Halts the target CPU.
interface	Selects the target interface which is used by J-Link
jtagconf	Configures a JTAG scan chain with multiple devices on it.
long	Reads or writes a word from/to given address.
memU8	Reads or writes a byte from/to given address.
memU16	Reads or writes a halfword from/to given address.
memU32	Reads or writes a word from/to given address.
reg	Reads or writes from/to given register.
remoteport	Changes the GDB server remote port.
reset	Resets and halts the target CPU.
select	Selects the way J-Link is connected to host system.
semihosting enable	Enables semi-hosting.
semihosting ARMSWI	Sets the SWI number used for semi-hosting in ARM mode.
semihosting ThumbSWI	Sets the SWI number used for semi-hosting in thumb mode.
setbp	Sets an instruction breakpoint at a given address.
sleep	Sleeps for a given time period.
speed	Sets the JTAG speed of J-Link / J-Trace.
step	Performs one or more single instruction steps.
waithalt	Waits for target to halt code execution.
wice	Writes to given IceBreaker register.

Table 3.1: GDB remote commands

GDB sends the remote commands to the GDB Server. Remote commands are preceded by the `monitor` keyword. 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.

3.5.1   AllowSimulation

Syntax

```
AllowSimulation <Value>
```

Description

Enables or disables ARM instruction set simulation.

Example

```
monitor AllowSimulation 1  // Enables ARM instruction set simulation
monitor AllowSimulation 0  // Disables ARM instruction set simulation
```

3.5.2   clrbp

Syntax

```
ClrBP [<BPHandle>]
```

or

```
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
```

or

```
monitor ci 1
```

3.5.3   cp15

Syntax

```
cp15 <CRn>, <CRm>, <op1>, <op2> [= <data>]
```

Description

Reads or writes from/to cp15 register. If `<data>` is specified, this command writes the data to the cp15 register. Otherwise this command reads from the cp15 register. For further information please refer to the ARM reference manual.

Example

```
Read: monitor cp15 1, 2, 6, 7              // Read
Write: monitor cp15 1, 2, 6, 7 = 0xFFFFFFFF  // Write
```

3.5.4   endian

Syntax

```
endian <endianess>
```

Description

Sets endianess of target, where `<endian>` can either be `big` or `little`. Example

```
monitor endian little
```

Additional information

By default, the GDB server is configured to use big endianess.

3.5.5   flash breakpoints

Syntax

```
monitor flash breakpoints = <Value>
```

Description

This command enables/disables the flash breakpoints feature.

Example

```
monitor flash breakpoints = 1  // Enable flash breakpoints
monitor flash breakpoints = 0  // Disable flash breakpoints
```

3.5.6   flash cpuclock

Syntax

```
flash cpuclock = <Value>
```

Description

Sets the CPU clock frequency the CPU is currently running with. `<Value>` is given in Hertz (Hz). Some flash algorithms require the current CPU clock frequency for correct flash programming operation, but on some cores the current CPU clock speed can not be measured by the flash algorithm so it is necessary to set it manually.

Example

```
monitor flash cpuclock = 12000000 // Tells the flash algo that
                                   // the CPU is running at 12 MHz
```

3.5.7   flash device

Syntax

```
flash device = <DeviceID>
```

Description

Selects the target's flash device. The flash device is selected by its device identifier. For more information about the supported device identifiers, please refer to the *J-Link / J-Trace User Guide* chapter *Flash programming and flash breakpoints*.

Example

```
monitor flash device = AT91SAM7S256
```

3.5.8   flash download

Syntax

```
monitor flash download = <Value>
```

Description

This command enables/disables the flash download feature.

Example

```
monitor flash download = 1 // Enables flash download
monitor flash download = 0 // Disables flash download
```

3.5.9   go

Syntax

```
go
```

Description

Starts the target CPU.

Example

```
monitor go
```

3.5.10 halt

Syntax

```
halt
```

Description

Halts the target CPU.

Example

```
monitor halt
```

3.5.11 interface

Syntax

```
interface JTAG | SWD
```

Description

Selects the target interface which is used by J-Link. Currently JTAG and SWD are supported.

Example

```
monitor interface SWD // Selects SWD as target interface
```

3.5.12 jtagconf

Syntax

```
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
```

3.5.13 long

Syntax

```
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. This command is similar to the WriteU32 command. Refer to *memU32* on page 26 for more information.

Example

```
monitor long 0x50000000          // Read
monitor long 0x50000000 = 0xFFFF  // Write
```

3.5.14 memU8

Syntax

```
MemU8 <address> [= <value>]
```

Description

Reads or writes a byte from/to a 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 memU8 0x50000000       // Read
monitor memU8 0x50000000 = 0xFF // Write
```

3.5.15 memU16

Syntax

```
memU16 <address> [= <value>]
```

Description

Reads or writes a halfword from/to a 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 memU16 0x50000000       // Read
monitor memU16 0x50000000 = 0xFFFF  // Write
```

3.5.16 memU32

Syntax

```
MemU32 <address> [= <value>]
```

Description

Reads or writes a word from/to a given address. If `<value>` is specified, this command writes the value to the given address. Otherwise this command reads from the given address. This command is similar to the long command. Refer to *long* on page 25 for more information.

Example

```
monitor MemU32 0x50000000             // Read
monitor MemU32 0x50000000 = 0xFFFFFFFF  // Write
```

3.5.17 reg

Syntax

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

or

```
reg <RegName> [= (<address>)]
```

Description

Reads or writes from/to given register. If `<value>` is specified, this command writes the value into the given register. If `<address>` is specified, this command writes the memory content at address `<address>` to register `<RegName>`. Otherwise this command reads the given register.

Example

```
monitor reg pc   = 0x00
monitor reg cpsr = 0x1F
monitor reg r0   = (0x40)
monitor reg pc   = (0x100)
```

3.5.18 remoteport

Syntax

```
remoteport <port>
```

Description

Changes the port an which the GDB server listens for connections.

Example

```
monitor remoteport 8000
```

3.5.19 reset

Syntax

```
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	Normal (default if no reset type is specified) 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.
1	Breakpoint @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: 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. b) Some MCUs contain a bootloader program (sometimes called kernel), which needs to be executed to enable JTAG access.
2	Analog Devices The following sequence is executed: - The CPU is halted - A soft reset sequence is downloaded to RAM - A breakpoint at 0 is set - The soft reset sequence is executed This sequence performs a reset of CPU and peripherals and halts the CPU before executing instructions of the user program. It is recommended reset sequence for Analog Devices ADuC7xxx MCUs and works with these chips only.
3	No reset is performed. Nothing happens.
4	Halt @watchpoint 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.

Table 3.2: Reset types

Reset type	Explanation
5	Halt @DBGRQ 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.
6	Software reset. 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.
7	Reserved.
8	Software, for ATMEL AT91SAM7 MCUs. The reset pin of the device is per default disabled. This means that the reset strategies which rely on the reset pin (low pulse on reset) do not work per default. For this reason a special reset strategy has been made available. It is recommended to use this reset strategy. This special reset strategy resets the peripherals by writing to the `RSTC_CR` register. Resetting the peripherals puts all peripherals in the defined reset state. This includes memory mapping register, which means that after reset flash is mapped to address 0. It is also possible to achieve the same effect by writing 0x4 to the `RSTC_CR` register located at address 0xfffffd00.

Table 3.2: Reset types

Example

```
monitor reset
monitor reset 1
```

3.5.20 select

Syntax

```
select USB
```

or

```
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
```

3.5.21 semihosting enable

Syntax

```
semihosting enable [<VectorAddr>]
```

Description

Enables semi-hosting with the specified vector address. If no vector address is specified, the SWI vector (at address 0x8) will be used.

Example

```
monitor semihosting enable
```

3.5.22 semihosting ARMSWI

Syntax

```
semihosting ARMSWI <Value>
```

Description

Sets the SWI number used for semi-hosting in ARM mode. The default value for the ARMSWI is 0x123456.

Example

```
monitor semihosting ARMSWI 0x123456
```

3.5.23 semihosting ThumbSWI

Syntax

```
semihosting ThumbSWI <Value>
```

Description

Sets the SWI number used for semi-hosting in thumb mode. The default value for the ThumbSWI is 0xAB

Example

```
monitor semihosting ThumbSWI 0xAB
```

3.5.24 setbp

Syntax

```
setbp <Addr> [<Mask>]
```

or

```
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
```

3.5.25 sleep

Syntax

```
sleep <Delay>
```

Description

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

Example

```
monitor sleep 1000
```

3.5.26 speed

Syntax

```
speed <kHz>|auto|adaptive
```

Description

Sets the JTAG speed of J-Link / J-Trace. Speed can be either fixed (in kHz), automatic recognition or adaptive. In general, Adaptive is recommended if the target has an RTCK signal which is connected to the corresponding RTCK pin of the device (S-cores only). Refer to *J-Link / J-Trace User Manual* for detailed information about the different modes.

Example

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

3.5.27 step

Syntax

```
step [<NumSteps>]
```

or

```
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
```

3.5.28   waithalt

Syntax

```
waithalt <Timeout>
```

or

```
wh <Timeout>
```

Description

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

Example

```
monitor waithalt 2000
```

or

```
monitor wh 2000
```

3.5.29   wice

Syntax

```
wice <RegIndex> <value>
```

or

```
rmib <RegIndex> <value>
```

Description

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

Example

```
monitor wice 0x0C 0x100
```

or

```
monitor rmib 0x0C 0x100
```

3.6   Command line options

You can use command line configuration options to start the J-Link GDB Server with a configuration file or set the listening port of the J-Link GDB Server.

The configuration file can be used as an alternative to the gdbinit file. All commands listed in *Supported remote commands* on page 22 can be used.

Sample configuration file

```
//
// J-LINK GDB SERVER Configuration file
//
// Set the listening port of GDB Server to tcp port 2331
port 2331
// Set JTAG speed to 30 kHz
speed 30
// Set GDBServer to little endian
endian little
// Reset the chip to get to a known state.
reset
```

3.6.1   -cr

Description

Starts the GDB Server and enables a memory read-ahead optimization which can speed up debugging.

Example

```
jlinkgdbserver -cr
```

3.6.2   -endian

Description

Sets the endianess of the target where endianess can either be "little" or "big".

Example

```
jlinkgdbserver -endian little
```

3.6.3   -if

Description

Selects the target interface which is used by J-Link. Currently JTAG and SWD are supported. The default value is "JTAG".

Example

```
jlinkgdbserver -if swd
```

3.6.4   -ir

Description

Initializes the CPU register with default values on startup.

Example

```
jlinkgdbserver -ir
```

3.6.5   -log

Description

Starts the GDB Server with the option to write the output into a given log file.
The file will be created if it does not exist. If it exists the previous content will be
removed. Paths including spaces need to be set between quotes

Example

```
jlinkgdbserver -log "C:\my path\to\file.log"
```

3.6.6   -noreset

Description

Starts the GDB Server with preventing it to issue a reset when connecting to the tar-
get.

Example

```
jlinkgdbserver –noreset
```

3.6.7   -port

Description

Starts the GDB Server listening on a specified port. This option overrides the default
listening port of the J-Link GDB Server. The default port is 2331.

Example

```
jlinkgdbserver -port 2345
```

3.6.8   -scriptfile

Description

Passes the path of a J-Link script file to the GDB Server. This scriptfile is executed
before the GDB Server starts the debugging / identifying communication with the tar-
get. J-Link scriptfiles are mainly used to connect to targets which need a special con-
nection sequence before communication with the core is possible. For more
information about J-Link script files, please refer to *UM08001*, chapter *"Working with
J-Link and J-Trace"*, section *"J-Link script files"*.

Example

```
–scriptfile "C:\My Projects\Default.JLinkScript"
```

3.6.9   -select

Description

Starts the GDB Server and specifies the connection to J-Link via USB or TCP/IP.

Example

```
jlinkgdbserver –select usb
```

3.6.10   -speed

Description

Starts GDB Server with a given initial speed.
Available parameters are "adaptive", "auto" or a freely selectable integer value in
kHz.

Example

```
jlinkgdbserver -speed auto
jlinkgdbserver -speed 2000
```

3.6.11 -silent

Description

Starts the GDB Server in silent mode. No log window messages will be shown.

Example

```
jlinkgdbserver -silent
```

3.6.12 -vd

Description

Starts the GDB Server with the option to verify the downloaded program in the target.

Example

```
jlinkgdbserver -vd
```

3.6.13 -x

Description

Starts the J-Link GDB server with a configuration file. In contrast to the -xc command line option the J-Link GDB Server runs the commands in the configuration file once only direct after the start of the J-Link GDB Server.

Example

```
jlinkgdbserver -x <YourConfigurationFile>
```

3.6.14 -xc

Description

Starts the GDB server with a configuration file. The commands in the configuration file will not be executed until a debugging session is started. The J-Link GDB Server executes the commands specified in the configuration file with every start of a debugging session.

Example

```
jlinkgdbserver -xc <YourConfigurationFile>
```

3.7   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.

3.7.1   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.
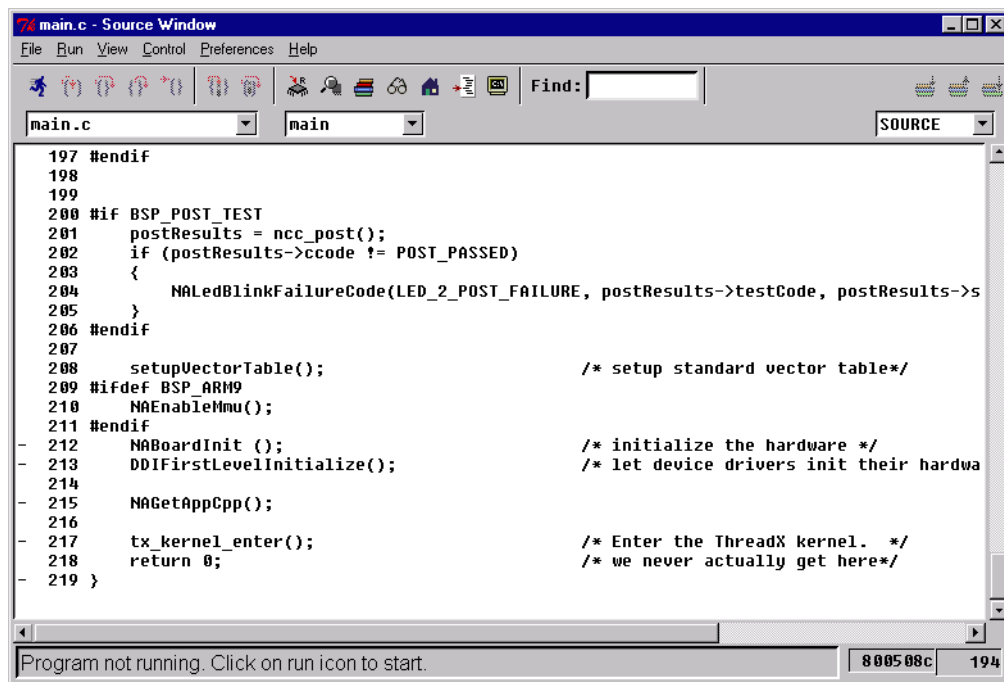
Refer to *http://sources.redhat.com/insight/* for detailed information about Insight.

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.

3.7.1.1   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.
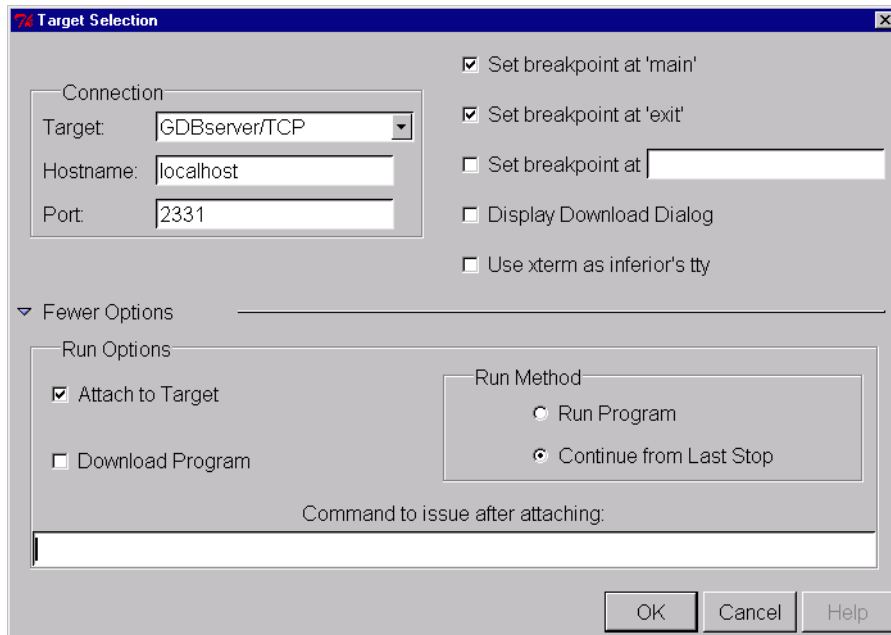
You can insert additional commands, for example for the initialization of your target core, in the textfield **Commands to issue after attaching** of the **Target Selection** dialog.

Choose **Run** | **Connect to target** to connect to your target via the J-Link GDB Server.
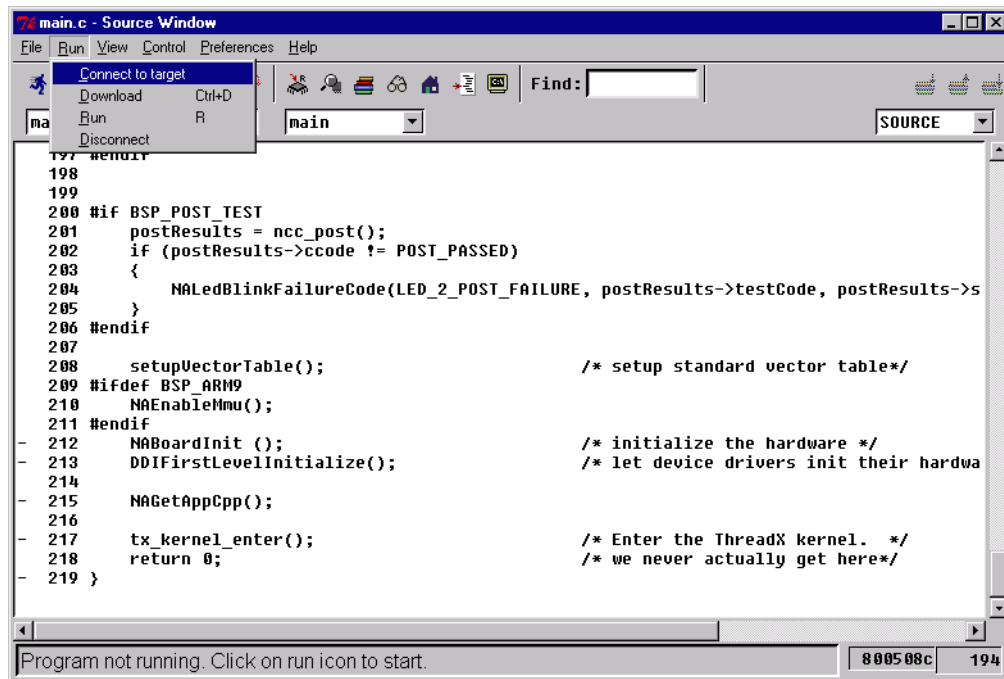
Check the status of the connection to J-Link GDB Server. The **Debugger** status message has changed to `Connected to 127.0.0.1` and the status field aside has changed from red to green.
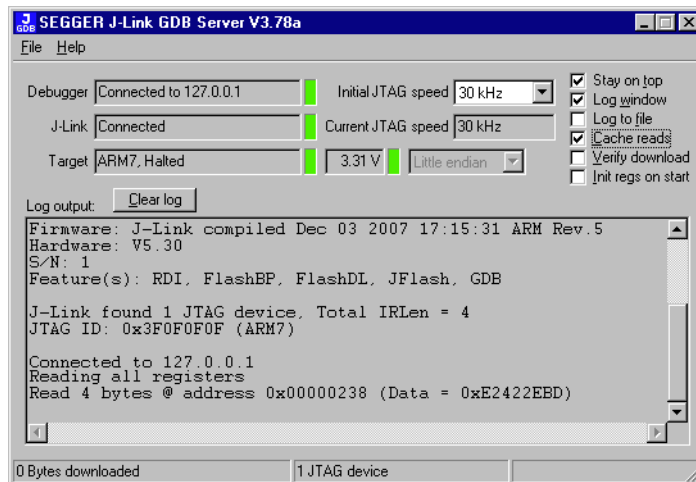
Download your program to your target

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

Using .gdbinit files

If you use one of our supplied `.gdbinit` files, you can abbreviate this sequence. The `.gdbinit` files initializes the connection to J-Link GDB Server with the default settings (J-Link GDB Server running on Localhost, listening on port 2331), initializes the core and downloads 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.

3.7.2   Eclipse

Eclipse is an open source platform-independent software framework, which has typi-
cally 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.
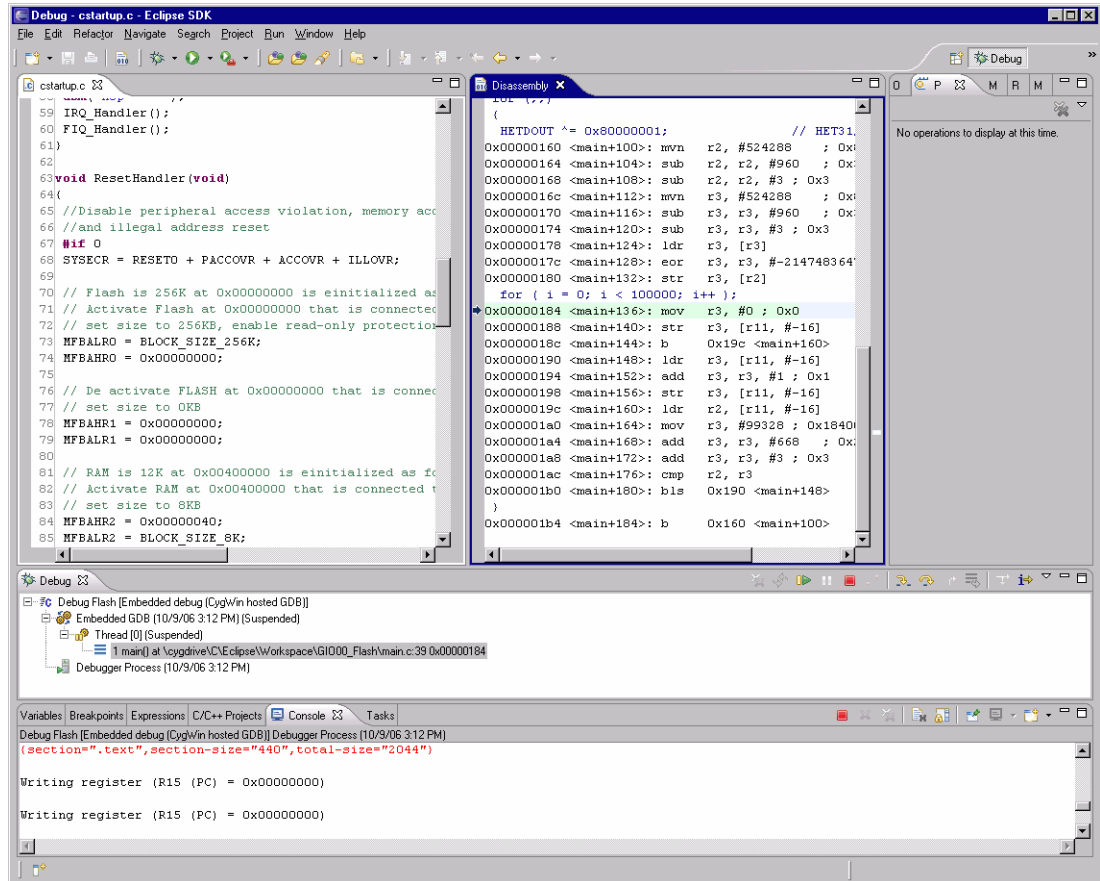
Refer to *http://www.eclipse.org* for detailed information about Eclipse.

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

3.7.3   Yagarto

The name Yagarto stands for "Yet another GNU ARM Toolchain". Yagarto is an Eclipse
compatible native Windows toolchain and consists of several packages. Yagarto itself
is the GNU toolchain (Binutils, Newlib, GCC compiler, and the Insight debugger) and
Yagarto IDE is a compilation of Eclipse, CDT and an additional plug-in which improves
the support for GDB embedded debugging in CDT.

The maintainer of Yagarto also offers some sample projects for various ARM cores
and some tutorials for the work with Eclipse and the GNU ARM toolchain.

Refer to *http://www.yagarto.de* for detailed information.

Chapter 4

Flash download and Flash break-points

This chapter describes how to use the J-Link flash download and Flash breakpoint features with the GDB Server.

4.1   Licensing

Flash breakpoints are features of the J-Link software which require separat licenses from SEGGER.

4.2   Enabling flash download and flash breakpoints

To use flash download and flash breakpoints with the J-Link GDB Server you have to enable them first. This is done by the remote commands `flash device` flash download and flash breakpoints. For more information about the `flash device` command please refer to chapter *Supported remote commands* on page 22. For example, if you want to enable flash download and flash breakpoints for a Atmel AT91SAM7S256 device simply add the following three lines to the `.gdb` file:

```
monitor flash device = AT91SAM7S256
monitor flash download = 1
monitor flash breakpoints = 1
```
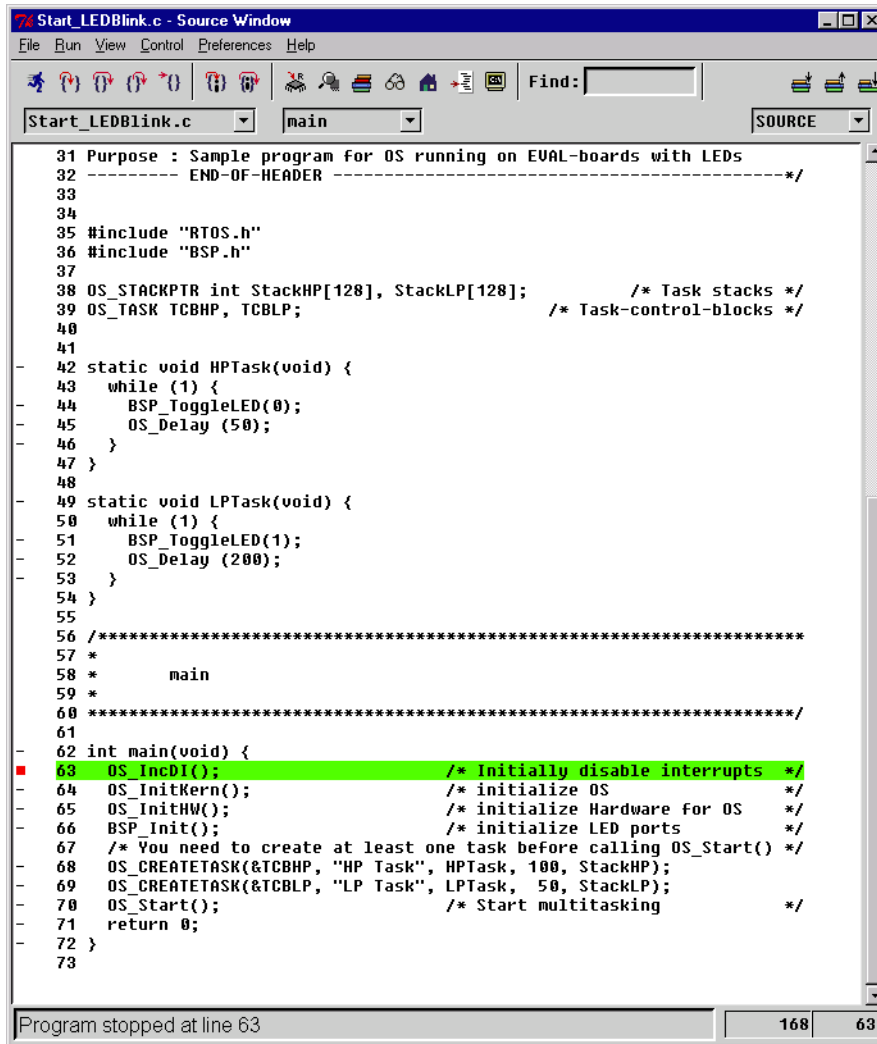
The J-Link GDB Server comes with sample projects for the most common flash micro-controllers. The sample projects can be found at `Samples\GDB\Projects` of the installation directory of the J-Link software and documentation package.

4.2.1   How to use the sample projects

First of all you have to choose an appropriate sample project for your device and unzip it, into a directory of your choice. After unzipping the project you can start the J-Link GDB Server.
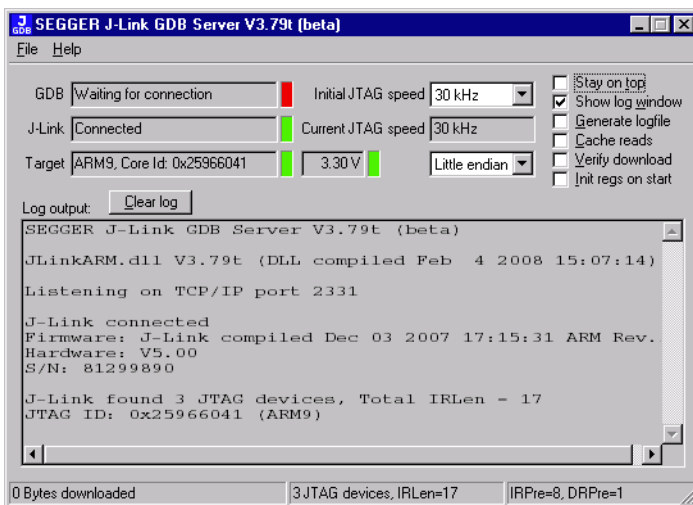
To start the sample project, simply start the `Debug_Flash.bat` file in the directory of the sample project. The sample projects already include an executable `Debug_Flash.elf` file so you don't have to recompile the project. After starting the debugger, the source windows should look as follows:

From now on you can set an unlimited number of breakpoints and debug through the application. Below a sample screenshot for using Flash breakpoints when debugging a sample project.

Chapter 5

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.

5.1   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:
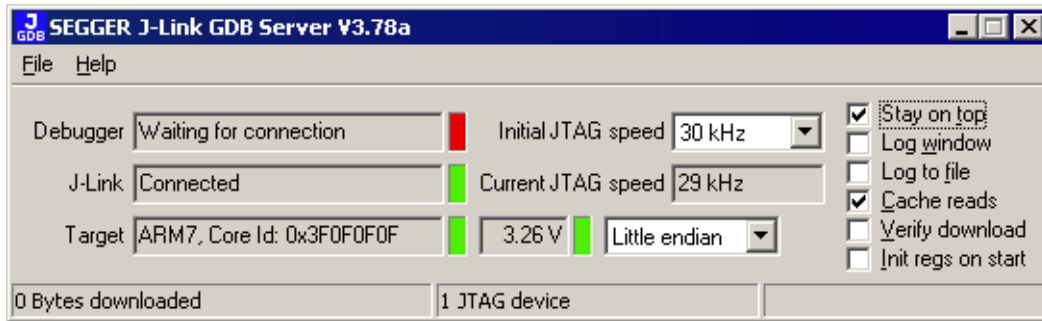
5.1.1   Copying .gdb files

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

In order to use these .gdb 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\.

5.2 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.
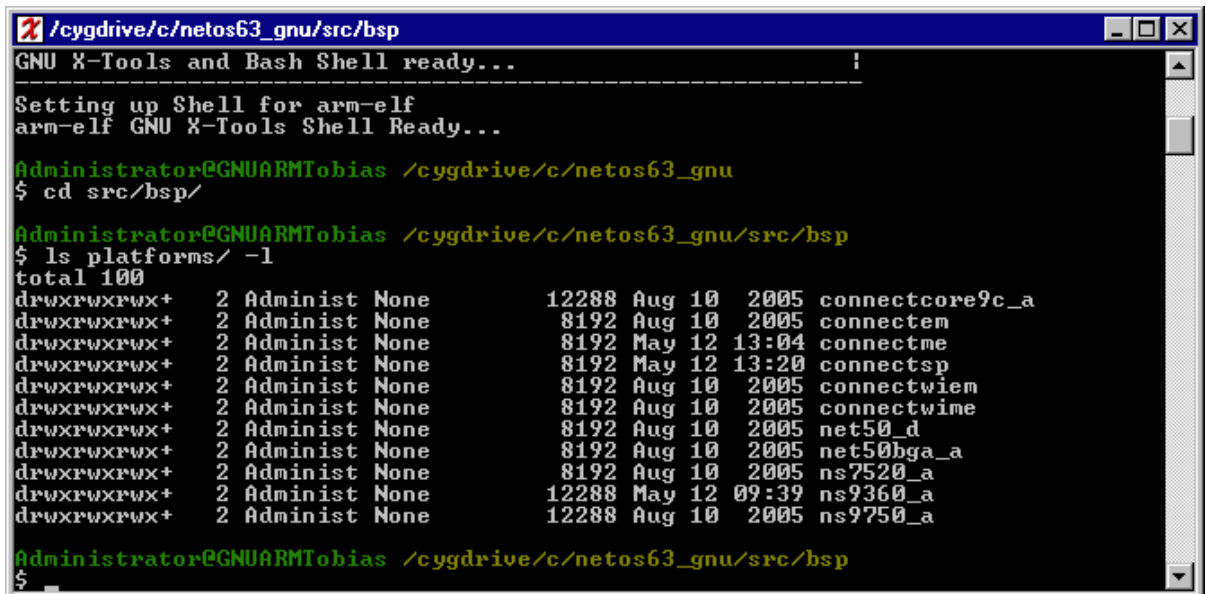
To compile the board support package (BSP), change to the bsp directory, which is located under `/cygdrive/c/netos63_gnu/src/bsp` by using the `cd` command which stands for "change directory".

```
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:

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.

Now you can start to compile the sample application, which is described under *Compiling the sample application* on page 50.

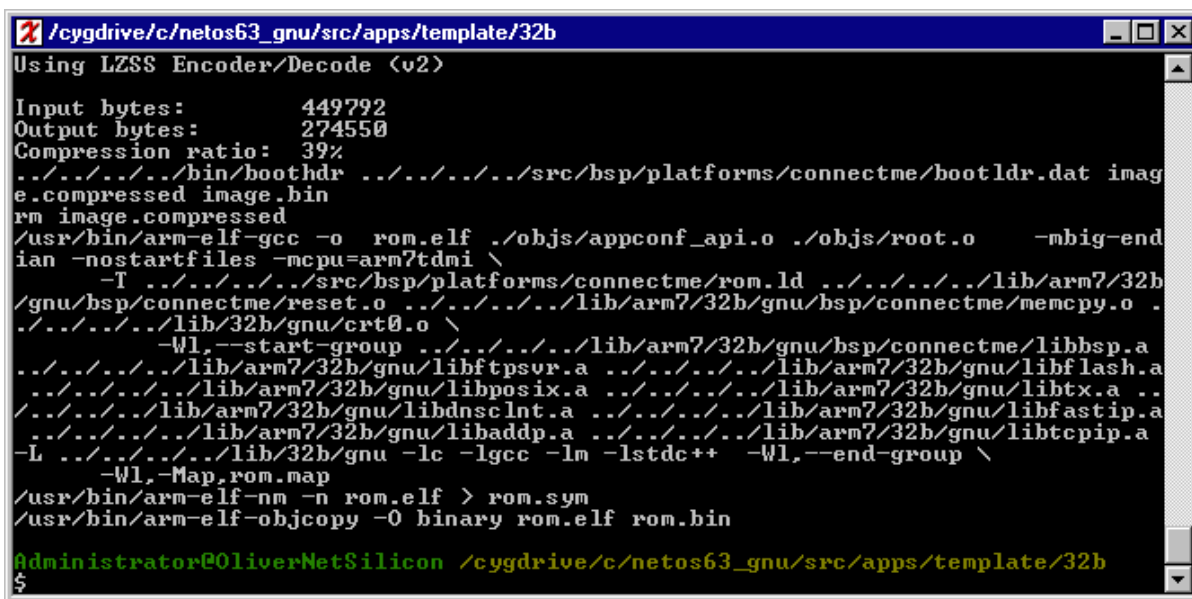# 5.3   Compiling the sample application

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.

5.4   Setting up the GDB configuration file

The GDB will search for the `.gdb` file in the workspace folder, from where you start the GDB out of. For this, you need to copy the 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 ./.gdb
```

5.5　Debugging the sample application

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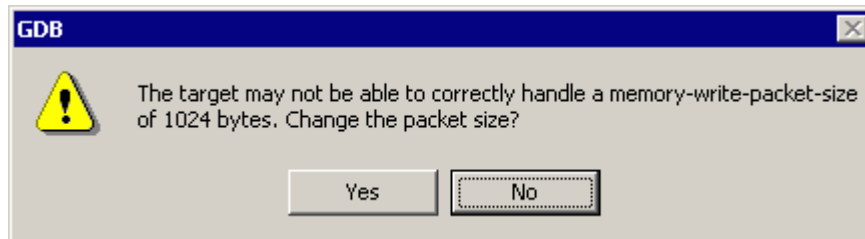
```
cd /cygdrive/c/netos63_gnu/src/apps/template/32b
```

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 commitee under:

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

Chapter 6

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.

6.1   Troubleshooting

6.1.1   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 55.

6.1.2   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-ialize 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 reialize 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.

6.2   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:

- 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.

6.3   FAQ

Q: Which CPUs are supported?
A: Every CPU supported by J-Link / J-Trace is supported.

Chapter 7

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 addressthan the most significant byte. See also Big-endian.

Target

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

Watchpoint

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

Chapter 8

Literature and references

This chapter lists documents, which we think may be useful to gain deeper under-standing 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 commitee (*www.gnu.org*).
[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 (*www.segger.com*).

Table 8.1: Literature and reference

Index