

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Green Hills Probe User's Guide



Green Hills Software, Inc.

30 West Sola Street
Santa Barbara, California 93101
USA
Tel: 805-965-6044
Fax: 805-965-6343
www.ghs.com

Copyright © 1983-2002 by Green Hills Software, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Green Hills Software, Inc.

DISCLAIMER

GREEN HILLS SOFTWARE, INC., MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Green Hills Software, Inc., reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Green Hills Software, Inc., to notify any person of such revision or changes.

Green Hills Software, the Green Hills logo, and MULTI are registered trademarks, and INTEGRITY and the Green Hills Probe are trademarks, of Green Hills Software, Inc.
System V is a trademark of AT&T.
ThreadX is a registered trademark of Express Logic, Inc.
MIPS is a registered trademark of MIPS Technologies, Inc.
Intel, i960, and Pentium are registered trademarks, and i386 and i486 are trademarks, of Intel Corporation.
PowerPC is a registered trademark of International Business Machines Corporation.
X and X Window System are trademarks of the Massachusetts Institute of Technology.
Microsoft is a registered trademark, and Windows, Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, and Windows XP are trademarks, of Microsoft Corporation.
ColdFire is a registered trademark of Motorola, Inc.
Motif is a trademark of Open Software Foundation, Inc.
Oracle is a registered trademark, and Oracle Help for Java is a trademark, of Oracle Corporation.
OSE is a registered trademark of OSE Systems, Inc.
Java is a registered trademark, and Sun is a trademark, of Sun Microsystems, Inc.
4.2BSD is a trademark of the Board of Regents of the University of California at Berkeley.
UNIX and Open Look are registered trademarks of UNIX System Laboratories.
osCAN is a registered trademark of Vector Informatik, GmbH.
Wind River Systems and VxWorks are registered trademarks, and Tornado and VxSim are trademarks, of Wind River Systems, Inc.

All other trademarks (registered or otherwise) are the property of their respective companies.

PubID: 10140
April 15, 2002

Contents

1 Introduction 1

2 Quick Start Guide 3

Introduction 4

Step One: Installing Software 4

Step Two: Connecting the Probe to Your Host Machine 5

Step Three: Connecting the Probe to Your Target Board 6

Step Four: Configuring the Probe 6

Step Five: Testing the Connection to Your Target 8

Step Six: Configuring your Target through MULTI 3.5 9

Manually Configuring your Target 11

3 Running MULTI with MPserv 17

Connecting MULTI to Your Green Hills Probe 18

Preparing Your Target for a Debug Session 20

4 Green Hills Probe Command and Scripting Reference 21

Introduction 22

Specifying Numbers for Green Hills Probe-Specific Commands 22

Green Hills Probe Terminal Prompts 23

Green Hills Probe-Specific Commands 24

Summary 24

Detailed Syntax and Examples 26

Generic Debug Server Commands 38

MPserv Scripting 45

General Notes 45

Scripting Syntax 45

5 Advanced Configuration 53

Introduction 54

Setting the IP Address 54

Contents

Setting the Netmask 54
Setting the Gateway 55
Setting the Adapter Type 55
Configuring Your Target 56
Setting the Serial Communication Speed (Baud Rate) 57
Turning the Status Checker On or Off 57
Setting the JTAG Clock Speed 58
Setting Debug Message Generation 58
Setting Override of Sanity Checks On or Off 58
Turning Target Power Detection On or Off 59
Setting the User String 59
Setting the Reset Pulse Length 59
Setting the JTAG TAP Reset Pulse Length 59
Setting the Reset Settle Length 60
Setting the JTAG TAP Reset Settle Length 60
Setting Reset Handshaking Timeout 60
Setting the Endianness 61
Target-Specific Options 61
ARM 61
MIPS 62
PowerPC 63
Star*Core 64

6 Web Interface Reference 67
Introduction 68
Main Page 68
Setup Pages 69
Help Pages 70

7 Utilities and Troubleshooting 71
Updating the Probe Firmware 72
The **MPadmin** Utility 72
Troubleshooting 75
My Green Hills Probe is not booting properly 75

Contents

My Green Hills Probe is not Communicating Properly with my Host Machine via: 76
RS-232 Serial Port 76
Ethernet Port 77
USB Port 78
My Green Hills Probe is not communicating properly with my target device 79

A Declaration of Conformity 87

Index 89

Contents

Chapter 1

Introduction

Congratulations and thank you for selecting the Green Hills Probe!

The Green Hills Probe is an advanced Hardware Debug Instrument that communicates to a target system over a standard JTAG or BDM test port and enables the MULTI debugger to control, debug and test your target CPU system in many different ways. The Green Hills Probe:

- Enables you to debug a range of target systems – from simple single processor systems with only internal memory to large heterogeneous multi-core systems with custom, complex memory hierarchies.
- Supports a wide variety of CPU families including PowerPC, MIPS, ARM, Star*Core and ZSP. A simple change of the external CPU Adapter and a firmware update is all that is needed to switch between different CPU families.
- Gives you extensive visibility and control of the target through Ethernet or USB[1] host interface ports.
- Provides a serial terminal, a telnet server, USB connectivity and a web server interface (in addition to the Debugger), all of which allow convenient configuration and control over your target.
- Can be used from project to project, even if each project involves different processor families, with little or no learning curve between projects.
- Is ideal for all product development stages - from board bring-up (where nothing is running on your target), to complete application debugging, and even product testing and configuration in a manufacturing environment.
- Provides a complete Hardware Debug Interface that will speed your development time through fast application download and outstanding target visibility.

1. Support via USB is available only on Windows 98/2000/ME/XP.

Chapter 2

Quick Start Guide

This Chapter Contains:

- Introduction
- Step One: Installing Software
- Step Two: Connecting the Probe to Your Host Machine
- Step Three: Connecting the Probe to Your Target Board
- Step Four: Configuring the Probe
- Step Five: Testing the Connection to Your Target
- Step Six: Configuring your Target through MULTI 3.5

Introduction

Use this section to set up your Green Hills Probe and begin debugging. Before beginning, make sure that you have the following components at hand:

Item No.	Component
1	Green Hills Probe
2	Power Supply
3	Ethernet Cable
4	USB Target Cable
5	Null Modem Serial Cable
6	Probe CD
7	CPU Adapter
8	Ribbon Cable
9	Your Target Development System

Step One: Installing Software

Install the probe support files from the CD on your host machine.

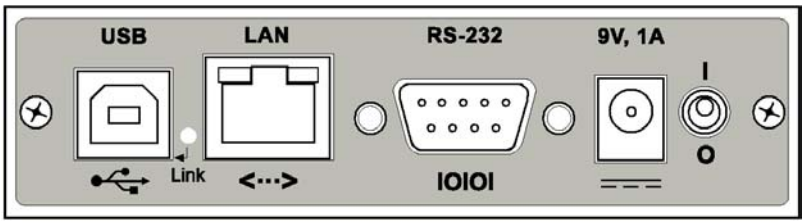
Windows	Solaris, Linux, and HPUX
Run the **setup.exe** program on the Probe CD and, when prompted for an install directory, specify your Green Hills tools directory.	Extract the appropriate **.tgz** file to your Green Hills installation directory, such as **/usr/green/**.



Note To review the capabilities of the probe through its terminal (RS232 or telnet) or Web server interface, you do not need to install host software. However, to integrate the probe with MULTI and make available its full capabilities, you must install the **mpserv** debug server and other supporting files on your host machine.

We assume, in what follows, that you have MULTI installed on your host machine.

Step Two: Connecting the Probe to Your Host Machine

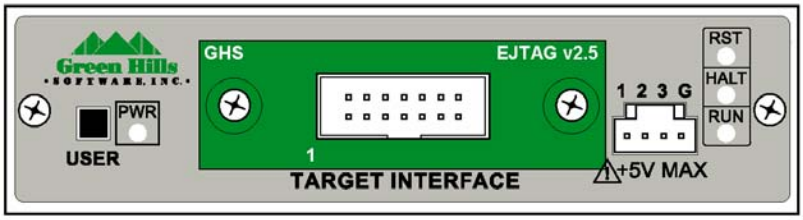


Ensure that the power switch for the probe is in the Off (0) position, and then connect the probe power supply between the probe's power input connector and an AC power outlet.

Using a USB[2] Connection	**Using an Ethernet Connection**
Connect the USB cable between the USB connector on the probe and a USB port on your host PC or USB hub. Note that the probe power supply must be used, even if using USB as a host connection.	Connect the Ethernet cable between the LAN connector on the probe and the network connection. You will also need, initially, to connect to the probe with an RS232 terminal or USB connection in order to configure its network settings. If you are planning to configure the probe via RS232, open up a terminal program using the RS232 port that the probe is connected to, and set the parameters to: • 9600 baud • 8 bits • no parity • 1 stop bit (8N1) • no flow control Hyperterminal, TeraTerm, and Tip are all appropriate terminal programs.

2. Support via USB is available on Windows 98/ME/2000/XP hosts only.

Step Three: Connecting the Probe to Your Target Board



Ensure that power to the target board is turned off, in order to avoid damage to your hardware. If you have been supplied with multiple CPU adapters, then make sure you have selected the correct one for your target.

Connect the ribbon cable between the CPU Adaptor on the front of the Probe and the JTAG/BDM connector on the target board, being careful to orient the cable correctly.

If you are uncertain about where on the target to connect the ribbon cable, refer to your board's reference manual.

Once the connections are complete, you can apply power to your target system, but this is not required at this time.



Note The USER button, located on the front left side of the Probe, enables and disables the target outputs during normal operation. When the outputs are enabled (LEDs are not flashing) pressing the USER button is equivalent to running the **jp off** command. Doing so will cause the LEDs to flash and disable the outputs. Pressing the button again will cause the LEDs to stop flashing and re-enable the outputs, equivalent to **jp on**.

Step Four: Configuring the Probe

Turn the probe power on by flipping the rear panel toggle switch to the On (|) position.

USB Connection	**Ethernet/RS232 Connection**
A **New Hardware Detected** box will appear briefly. If an **Install New Hardware Wizard** screen appears, then the probe USB system files were probably not installed. Verify that the **setup.exe** program from the probe CD has been run on your host PC. On Windows ME/98, you must reboot after running **setup.exe**. Open a DOS prompt, and type the following: *install_dir*`/mpadmin -usb -setup`	A startup banner and prompt will appear in the serial terminal window or on the telnet console. Type `setup` at the terminal prompt.

The interactive configuration script starts, and takes you through the following steps. At each step, you can press `enter` to accept the default setting which is indicated in square brackets in the terminal window. For more information on these configuration options, see Chapter 5, “Advanced Configuration”.

1. The Network Configuration requires setting three addresses. First, set `ip`, the IP address of the probe. This must be an IP address in dotted quad form. You may need to contact your system or network administrator for assistance with this and the following two options.
2. Next, set `netmask`, the Probe’s IP network mask. This must be an IP address mask in dotted quad form. The default setting is appropriate for most networks.
3. Finally, set `gateway`, the IP address of the gateway the Probe should use. This must be an IP address in dotted quad form.
4. The Target Interface configuration requires setting four parameters. First, set `adapter`; you should be able to use the `auto` default for the adapter setting. If that does not work, you can force the adapter type manually with the **set adapter** command, after completing the interactive setup (see “Setting the Adapter Type” on page 55).
5. Next, set `logic_high:` voltage to match your target system. Most CPU systems run at 3.3 volts, although some run at lower or higher voltage potentials.



Warning It is very important to set this value correctly for your target CPU type. Setting the value incorrectly may result in damage to the target system.

6. Next, set `multicore:`, to denote single-core or multi-core operation. For single-core operation, select **off**. For multi-core operation, select **on**.
7. The final Target Interface configuration is `target:`, the device type. If your multi-core setting is **off**, this option will be named **target**. If your multi-core setting is **on**, it will read **chain**. To see a list of supported device names, enter **?**, or see the table in "Configuring Your Target" on page 56.
8. Finally, you will be prompted for a number of target configuration settings, depending on your target. See Chapter 5, "Advanced Configuration" for details about these configuration options.

The following message appears:

```
Saving options to EEPROM...
```

If you have changed any options that require a probe reboot (such as changing its network parameters), the following message will also appear:

```
Options have been changed that require the Probe to be rebooted before taking effect:
Reboot now: [yes]
```

Press `enter` to reboot the probe. If you have configured your probe for Ethernet operation, you should now be able to ping it, connect to it via telnet, or open a web browser on its IP address.

Step Five: Testing the Connection to Your Target

The probe is now ready to view and control your target CPU and system. You can perform the following brief test to verify a working debug connection:

1. If you have not already done so, power up your target system.
2. For MIPS targets, the probe can automatically detect the type of core with the **detect** command. Enter the command:

   ```
   detect mips
   ```

   to automatically detect and configure the probe for your MIPS target.

3. At this time, the target system and Probe may not be synchronized. To ensure that they are communicating correctly, you should reset the target by opening a terminal window through Ethernet or RS232. (Connecting a command line terminal through USB is currently not supported.)

   Then type the following commands:

   - **tr r** (target reset and run)
   - **th** (target halt)
   - **rr** (register read)

   The **rr** command dumps the CPU registers to the terminal screen. If the registers do not appear, there might be a configuration problem. Return to "Step Four: Configuring the Probe" on page 6, or see "Troubleshooting" on page 75.

Step Six: Configuring your Target through MULTI 3.5

Before beginning your first debugging session, you should ensure that your target board is properly configured.

To help you with this, Green Hills provides example projects containing preconfigured setup script files and small test applications for many of the most popular processor/board combinations. If you are able to compile, download, and debug one of these example programs, your debug connection is working correctly.

To run an example project that will use a setup script to configure your target system, perform the following steps:

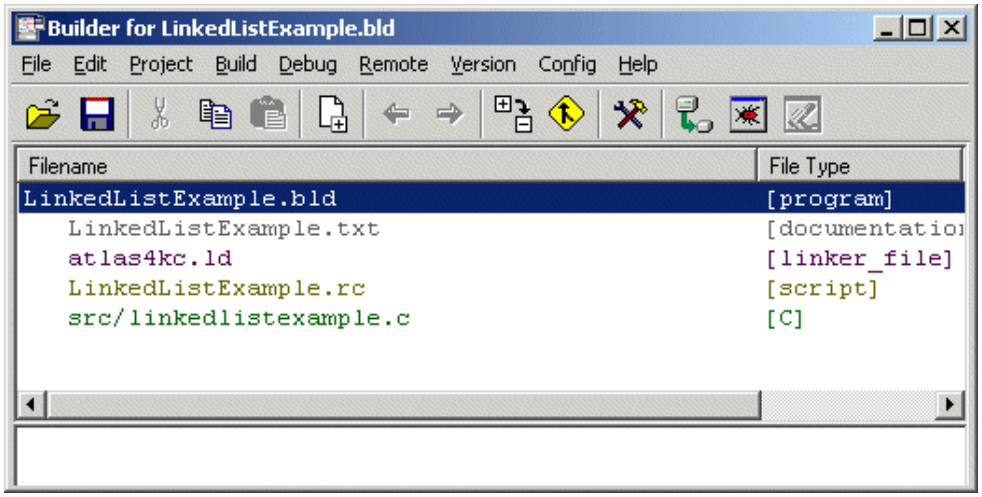
1. Open one of the Example Projects for your processor/board combination.

Windows	**UNIX**
From the **Start** button, select **Programs** → **MULTI** → **Examples** → *Your Board* → **Linked List Example**.	Navigate to *install_dir***/examples/***Your Board*



Note If you cannot find a processor/board combination which exactly matches your target, go to "Manually Configuring your Target" on page 11.

2. The MULTI **Builder** window opens



3. Select **LinkedListExample.bld** and click  to build the project.

4. Click  and enter one of the connections listed below:

USB Connection	Ethernet Connection
`mpserv -usb`	`mpserv` *ProbeIP*|*ProbeName*

5. Click  to open the **Debugger**.

   As soon as you start the program, MULTI will automatically download it to your target. You will now be able to test your connection by setting breakpoints, and performing register reads and writes.

6. If the test is successful, you are now ready to begin debugging your own application. For guidance on using the MULTI Debugger, see Chapter 3, “Running MULTI with **MPserv**”, and the *MULTI 2000 User’s Guide*.

   If you encounter problems, review Steps Five and Six above, and see “Manually Configuring your Target” on page 11 and “Troubleshooting” on page 75.

Manually Configuring your Target

If no example project is provided for your particular processor/board combination, (see “Step Six: Configuring your Target through MULTI 3.5” on page 9), you will need to edit the files in board directory which is closest to your target system. To do this, navigate to *install_dir*/**examples**, copy the relevant subdirectory, and rename it appropriately.

Open MULTI and navigate to your newly created directory.

Step One: Edit the Setup Script File

Open the **.dbs** setup script file in the editor.

If your Board can Initialize Itself

If your target has a valid ROM image which will initialize it upon reset, then delete the contents of the **.dbs** file and replace them with:

```
rst
run
sleep 3
halt
```



Note Depending on how long the board takes to set itself up, you may need to adjust the parameter of the **sleep** command.

Save the file and skip to “Step Two: Edit the Linker Directives File” on page 14.

If Your Board cannot Initialize Itself

If your target does not have a valid ROM image which will initialize it upon reset, then you will need to complete the following tasks in order to edit the setup script for your target:

1. In most cases your setup file should begin with the **rst** command, which resets and halts the target. All targets must be halted before any debugging activity begins, or unpredictable results may occur.
2. Configure your target’s memory controller based on your board’s memory resources. This is done by a sequence of register reads and writes and

memory reads and writes. If your memory controller and memory resources are already properly configured, skip to step 4. The following are general steps for configuring memory using a setup file:

a. Determine what memory resources your board has by answering the following questions:

- How fast and big is the board's memory, and where do you want to map it?
- Does the board have SRAM? If so, where is it?
- Does the board have DRAM? If so, where is it, and where is the DRAM controller for it?
- Does the DRAM controller need refresh timing information or knowledge of any special modes the DRAM chips may have, such as EDO Mode or Fast Page Mode?
- Does the board require a peripheral memory base register to access memory controllers or other on-chip peripherals?

b. If your processor requires you to set up the base register before you can access your memory controllers or other on-chip peripherals, set the base register now.

c. Using your processor's documentation and memory resources, determine which memory-related registers you need to set and what values these registers must have in order to properly configure your memory resources.

d. Using the **MPserv** command line, determine which **MPserv** commands set these memory-related registers to their appropriate values. (See Chapter 4, "Green Hills Probe Command and Scripting Reference" for a list of **MPserv** commands.)

e. Copy the necessary **MPserv** commands to your setup file.

3. Disable any interrupt sources that may disrupt the setup or destabilize the board's memory. For example, for some targets, you must disable the watchdog timer to prevent it from interrupting and ruining your target board setup. The following steps provide a general procedure for disabling interrupt sources:

a. Determine whether the processor has any interrupt sources that will disturb your debugging session.

b. Using the processor's documentation, determine which registers affect interrupt sources and then determine the values these registers must have to disable the interrupt sources.

c. Using the **MPserv** command line, determine which **MPserv** commands set the values of the registers you need to alter.

d. Copy the necessary **MPserv** commands to your setup file.

4. Make sure that the **.dbs** file now contains all of the commands required to prepare the target system, and then save it.



Note As you edit this setup script, keep the following in mind:

- The **mpserv** commands and scripting language are documented in Chapter 4, "Green Hills Probe Command and Scripting Reference".
- You can use **mpserv** to confirm the success or failure of each command individually instead of trying to debug an entire script. However, you should not start debugging with **mpserv** until you have completely finished setting up your debugging environment. Debugging with a board that has not been properly set up can leave the target in an uncertain state. To start **mpserv**, see "Connecting MULTI to Your Green Hills Probe" on page 18.
- Some commands cannot be tested individually because they must be executed within a certain time period in relation to other commands. For example, a watchdog timer may need to be disabled seconds after an **rst** command. It is difficult to type in the two commands that will disable the watchdog timer before the timer expires. In cases like this, put the relevant commands into a small script file and run the file using the **script** command.

The following example script can be used as a guide as you edit your setup script.

Example 1. Setup Script

The following is an example setup script for an Atmel AT91EB40 (ARM7TDMI-based evaluation board).

```
# Setup script for the Atmel AT91EB40 board
# EBI_CSR0
mw 0xffe00000 0x01002535
# EBI_CSR1
mw 0xffe00004 0x02002121
# REMAP Done
mw 0xffe00020 0x00000001
#2M bytes per CS
mw long 0xffe00024 0x00000006
#Set the CPSR register as it would be when reset
reg cpsr 0xd3
```

Step Two: Edit the Linker Directives File

After you have edited and saved the **.dbs** file, open the **.ld** linker directives file. Edit this file so that it matches the memory layout specified in your setup script file, and then save it. For more information on linker directives files, see the *Development Guide* for your target processor.

Step Three: Test Your Debugging Environment

After you have edited the **.dbs** and **.ld** files, you should use the following procedure to test whether your debugging environment is properly configured before downloading and debugging a program.

1. Make sure all connections are tight and secure. Make sure both the target system and Green Hills Probe are powered.
2. Reset the target system by running your setup file. To run the setup file, use the **script** command in the **MPserv Target** window. For example, to run a setup file called **test.dbs**, use the command:

   ```
   script test.dbs
   ```



Note Do not use the command line option **-setup** to test the script. This option runs the setup file only when a program is downloaded, and you should not be downloading any code until after you have tested the debugging environment.

3. Test whether reads and writes to the register are reaching the target by performing the following steps:

a. Pick a harmless register, such as a general purpose register.

b. Read the register and note its value. For example, on a PowerPC, enter `reg r0`.

c. Write a different value to the same register. For example, on a PowerPC, enter `reg r0 deadbeef`.

d. Read the register again and see if it changes to the new value. If it does, the target connection is able to alter registers.

4. Test whether you can access the target memory through your Green Hills processor probe. We recommend that you test memory locations where you will be downloading a program. To test the target memory:

   a. Read a memory location and note its value. For example, enter:

      ```
      m a0020000
      ```

   b. Write a different value to the same memory location. For example, enter:

      ```
      m a0020000=deadbeef
      ```

   c. Read the memory location again and see if it changes to the new value.

If your target configuration appears to be correct, you are now ready to connect to your target and begin debugging. To do this, return to the Builder window that contains your example project and perform the following steps:

1. Select **LinkedListExample.bld** and click  to build the project.
2. Click  and enter one of the connections listed below:

USB Connection	**Ethernet Connection**
`mpserv -usb`	`mpserv` *ProbeIP|ProbeName*

3. Click  to open the **Debugger**.

   As soon as you start the program, MULTI will automatically download it to your target. You will now be able to test your connection by setting breakpoints, and performing register reads and writes.

4. If the test is successful, you are now ready to begin debugging your own application. For guidance on using the MULTI Debugger, see Chapter 3, "Running MULTI with **MPserv**", and the *MULTI 2000 User's Guide*.

If you encounter problems, review Steps Five and Six above, and see “Manually Configuring your Target” on page 11 and “Troubleshooting” on page 75.

Chapter 3

Running MULTI with MPserv

This Chapter Contains:

- Connecting MULTI to Your Green Hills Probe
- Preparing Your Target for a Debug Session

MULTI uses the **MPserv** debug server to communicate with the probe. Once you have tested your debugging connection (see “Step Six: Configuring your Target through MULTI 3.5” on page 9 in Chapter 2, “Quick Start Guide”), you can begin a debugging session using the Probe together with MULTI and **MPserv**.

Connecting MULTI to Your Green Hills Probe

To connect to your target from the MULTI Debugger command line, enter the following command:

If you are using a USB connection:

`remote mpserv -usb [`*index*`] [`*arguments*`]...`

If you are using an Ethernet connection:

`remote mpserv` *hostname* `[`*arguments*`]...`

where *arguments* can be any combination of the **MPserv** startup options listed below:

Argument	Description
-force_coreid *core*	Allows multiple instances of MULTI to connect to a multi-core target system, and to limit debugging information to a particular, single *core*. This argument allows you to debug multi-core target systems from versions of MULTI prior to version 3.5 (non-multi-core versions).
-log *filename*	Creates a log of actions performed by **MPserv** and stores it in the specified file. If you have problems using **MPserv** with your target, this log file will assist Green Hills customer support in finding a solution.
-notext	Excludes the `.text` section from every host-to-target download. The `.text` section contains code and is downloaded by default.
-nodata	Excludes the `.data` section from every host-to-target download. The `.data` section contains initialized variables and is downloaded by default.
-bss	Includes the `.bss` section in every host-to-target download. By default, the `.bss` section is not downloaded. It contains uninitialized data and is cleared to all zeros by programs compiled with Green Hills tools.

Argument	Description
-noload	Excludes all program sections (`.text`, `data`, and `.bss`) from every host-to-target download.
-set *option*=*value*	Sets a configuration *option* to *value* before beginning the current session. Any changes will be automatically saved to `EEPROM`.
-setup *file*	Runs the commands in the specified setup file immediately prior to every host-to-target download.
-usb [*index*]	Connects to the Probe using the USB port. The optional *index* parameter can be used to specify a particular probe in the event that multiple probes are connected to a single host.

Example 1.

The following command:

```
remote mpserv gh_probe1
```

- Connects **MPserv** via ethernet to a probe with the hostname `gh_probe1` using the default options

Example 2.

The following command:

```
remote mpserv 192.168.101.88 -setup init.dbs
```

- Connects **MPserv** via ethernet to a probe with the IP address 192.168.101.88.
- Ensures that the commands in the setup file **init.dbs** are automatically run immediately prior to every host-target download.

Example 3.

The following command:

```
remote mpserv orion -setup board_abc.dbs
```

- Connects **MPserv** via ethernet to a probe with the hostname `orion`.
- Ensures that the commands in the setup file **board_abc.dbs** are automatically run immediately prior to every host-target download.

Preparing Your Target for a Debug Session

It is important to configure your target system before beginning to debug your application

It is important that the target is “clean” and stable before downloading code. When you need to reset the target, you can use a setup file to do this in one of the following ways:

- Use the **-setup** start-up option to ensure that the setup file is automatically run immediately prior to every host-to-target download.
- Run the setup file manually in the **MPserv** target window using the **script** command.
- Add a button to MULTI that runs the setup file when clicked. For example, to create a button called **reset** that runs a setup file named **target_script.dbs**, use the following MULTI command:

```
debugbutton reset c=target script target_script.dbs
```

- You can use the **reset** button any time you need to reset the target to ensure a clean debugging environment. For more detailed information on this command and configuring MULTI so that this button is created by default when MULTI is run, see the “Customizing and Configuring” section of the *MULTI 2000 User’s Guide*.

Chapter 4

Green Hills Probe Command and Scripting Reference

This Chapter Contains:

- Introduction
- Specifying Numbers for Green Hills Probe-Specific Commands
- Green Hills Probe Terminal Prompts
- Green Hills Probe-Specific Commands
- Generic Debug Server Commands
- **MPserv** Scripting

Introduction

The commands in this chapter are divided into two sections, “Green Hills Probe-Specific Commands” on page 24, and “Generic Debug Server Commands” on page 38. The Probe-specific commands (with certain, noted, exceptions) can be entered in the MULTI Debugger **Target** window, the MULTI Debugger **Command** Window (if prefixed with the **xmit**, **target**, or **xmitio** command), or a telnet or serial terminal window.

The generic commands can only be entered in the MULTI Debugger **Target** and **Command** windows.

All of the commands for **MPserv** are case-insensitive.

Specifying Numbers for Green Hills Probe-Specific Commands

Numbers may be specified either in regular decimal form (for example: `1024`), or in hexadecimal form, prefixed by `0x` (for example: `0x400`). Decimal numbers may not exceed 32 bits in size, while hexadecimal numbers can be as large as the application requires.

When the number of bits of a number is known (for example, when specifying a register value), you can use two special fill digits: `h` (for high) and `z` (for zero).

You cannot replace both the `0s` and `1s` in `0x0a01ffff` with a `z` and `h` fill, since it would be ambiguous. To overcome this, you can place delimiters in the number.

Every 16 bits you can place an underscore (“_”), and every 32 bits you can place a colon (“:”).

The following table illustrates the use of these fill digits and delimiters:

Original Hexadecimal Number	Number with Fill Digits and Delimiters
`0xa0000000`	`0xaz`
`0xa001ffff`	`0xaz1ffff`
`0xa001ffff`	`0xa001h`
`0xa001ffff`	`0xaz1_h`
`0xffffffffa0010000`	`0xh:a001z`

Green Hills Probe Terminal Prompts

At a terminal prompt, the Green Hills Probe will indicate the target type, position in scan chain, and current status on its command line prompt. Commands will be directed to the core displayed in the prompt. The command line prompt for single core uses the following notation:

```
coretype[status]
```

The command line prompt for multi-core uses the following notation:

```
coretype[position, status]
```

where:

Command or Option	Description
coretype	The CPU core type.
position	Position in JTAG scan chain.
status	The current execution status.

The current execution status consists of the following options:

Current Execution Status	Description
?	Status is currently unknown. This may be due to an error, or the probe may have not requested target information.
h	Target is currently halted because a probe requested the halt.
s	Target is currently halted after completing a single instruction step.
b	Target is currently halted due to a software breakpoint.
B	Target is currently halted due to a hardware execution breakpoint or data watchpoint.
E	Target is currently halted due to an unknown exception.
C	Probe has requested the target resume, but it has not successfully left debug mode when the Probe last queried target status.
H	Probe has requested the target to halt, but it has not successfully entered debug mode when the Probe last queried the target status.
R	Target is currently held in reset.

Current Execution Status	Description
f	Target is running free. This could occur if the target is reset external to the Probe.
n	Target is not connected; pins are `tristated`.
Y	Target is busy; status cannot be received.
!	Target is not in a recognizable state; this can be caused by incorrect Probe configuration or faulty hardware.

For example:

```
mips32_4kep[h] %
```

indicates the Probe is connected to a single MIPS Technologies MIPS32 4KEp core that is presently halted. The following multi-core example:

```
arm7tdmi[0,b]
```

indicates the Probe's commands are directed toward the ARM7TDMI core that is the first in the scan chain.

Green Hills Probe-Specific Commands

The Probe-specific commands are grouped into six command actions: configuration, front panel I/O, system, target, test, and JTAG. The following tables list the Green Hills Probe-specific commands in summary form by command type. For detailed command descriptions and full syntax, see "Detailed Syntax and Examples" on page 26.

Summary

Configuration Commands

de	Attempts to detect target endianness.
detect	Attempts to determine the target type and endianness.
restore	Restores the current configuration settings from non-volatile memory.
save	Saves the current configuration settings to non-volatile memory.

set	Sets an option by name.
setup	Interactively configures the probe.

System Commands

alias	Makes a command alias.
debug	Determines the destination of debug messages.
exit	Exits a terminal session.
help	Displays help messages.
info	Displays basic probe information.
reboot	Reboots the probe.
w	Displays a list of all active connections.

Target Commands

bc	Clears a breakpoint.
bca	Clears all breakpoints.
bl	Lists all breakpoints.
bs	Sets a breakpoint.
ma	Defines debug memory access permissions.
md	Displays a block of target memory.
mr	Performs a sized memory read.
mw	Performs a sized memory write.
rr	Reads target registers.
rw	Writes a value to a register.
t	Selects a target (only for multi-core systems).
tc	Continues running the target from the current PC (program counter).
th	Target halt.
ti	Displays current target status information.
tr	Resets the target system.
ts	Single steps one instruction on the target.

Test Commands

vb	Verifies that the JTAG bypass register can be scanned properly.
vbp	Verifies breakpoints.
vc	Verifies code.
vlb	Verifies internal loop back.
vm	Verifies memory.
vrh	Verifies asynchronous run/halt control.
vsi	Verifies single step.
vta	Verifies Target Adapter.

JTAG Commands

jd	Performs a JTAG data scan.
ji	Performs a JTAG instruction scan.
jp	Displays and controls target adapter I/O pins.
jr	Resets the JTAG TAP controller (not available for `BDM` targets).

Front Panel I/O Pin Commands

gpin	Sets or displays the present state of the front panel I/O pins.
gpincfg	Sets or displays the configuration of the front panel I/O pins.

Detailed Syntax and Examples

The Green Hills Probe-specific commands are as follows:

alias [*alias*] [*expansion*]

Queries and sets command aliases. If no parameters are given then all aliases and their current values are listed.

If only an alias name is specified then the value is displayed. Note that aliases are not saved in non-volatile memory.

Examples:

`alias`: Displays all aliases.

`alias mr1`: Shows the expansion of the `mr1` alias

`alias dbuf md 0xff004020`: Defines a new alias `dbuf` which will dump the memory at `0xff004020` when run.

bc *id*

Clears a breakpoint with id *id*.

bca

Clears all breakpoints.

bl

Lists all breakpoints.

bs [r] [w] [x] *address* [/*address_mask*] [*size*] [d=*data*[/*data_mask*]] [c=*count*]

Sets a breakpoint at address *address*.

Any combination of `r` (read), `w` (write), or `x` (execute) specifies a hardware breakpoint, otherwise a software breakpoint is set.

The optional argument *data* specifies data to use for hardware compare breakpoints.

You may specify an *address_mask* to apply to a target address before comparing it to *address*, and/or a *data_mask* to apply to the target data before comparing it to *data*.

The optional argument *size* specifies the size of the breakpoint. The default size is the target's instruction length.

The optional argument `c=`*count* specifies the number of times the breakpoint should be hit before the target actually halts.

Note that a mask of `0xffffffff` means all bits are significant, while a mask of `0xffffff00` means the least significant byte is ignored in the comparison. Masks and Read/Write/Execute filtering are only available if hardware breakpoints are supported by the target CPU.

Also, hardware breakpoint support varies between target CPUs. Consult your CPU or core manufacturer's manual for more details on your target's level of hardware breakpoint support.

Examples:

`bs x 0x200000`: Breaks if the target executes an instruction at this address.

`bs 0xa00000000`: Sets a software breakpoint at `0xa0000000`.

`bs rw 0x4000/0xff 2 d=0x1234 c=3`: Breaks the third time a 16-bit value of `0x1234` is read in the address range `0x1000` through `0x10fe`.

`bs w 0x1000 4 d=0x12340000/0xffff0000`: Breaks when a 32-bit value with the most significant bits equal to `0x1234` is written to `0x1000`.

de

Detects whether the target is running in Big endian or Little endian mode. If detection is successful, the endianness option will be changed accordingly, not NOT stored to non-volatile memory. To save the change to non-volatile memory, use the **save** command. This change will stay in effect until the next reboot. Not available from **MPserv**, Probe terminal specific.

Example:

`de`

attempts to detect the target endianness.

debug

Requests that all internal debug messages be directed to the current communications port. See the help message for the **set debug** command to customize which debug messages are generated.

detect

Detects what kind of processor is connected to the Probe. If detection is successful, endianness will also be detected. These options will be changed, but NOT stored to non-volatile memory. To save the changes to non-volatile member, use the **save** command.Not available from **MPserv**, Probe terminal specific.

The parameter **$***family* specifies the processor family. Currently, **de** is only supported for MIPS.

Example:

```
detect $mips
```

detects a target with a MIPS core.

exit

Exits a telnet session. You cannot exit an RS232 serial terminal session.

(Not available in the MULTI Debugger **Target** and **Command** windows)

gpin

Sets or displays the present state of the front panel I/O pins. Pins are set and displayed in hexadecimal, with the values from 0 to 7 representing the binary state of the 3 I/O pins.

(Not available in the MULTI Debugger **Target** and **Command** windows)

gpincfg *pin_config*

Sets or displays the configuration of the front panel I/O pins.

The parameter *pin_config* specifies the pin configuration, where:

- `1/2/3` specifies the I/O pin to configure
- `I` configures the pin as `INPUT`
- `D` configures the `INPUT` pin to never trigger
- `R` configures the `INPUT` pin to trigger on `RISING` edge
- `F` configures the `INPUT` pin to trigger on `FALLING`
- `O` configures the pin as `OUTPUT`
- `A` configures the `OUTPUT` pin with `ACTIVE` drive (`+5v/GND`)
- `G` configures the `OUTPUT` pin with `GROUNDING` drive (`float/GND`)

Examples:

`gpincfg 1OG3OG`: Configures pins 1 and 3 as `OUTPUTS` with `GROUNDING` drive.

(Not available in the MULTI Debugger **Target** and **Command** windows)

help [*group* | *topic*[*subtopic*]]

Displays help messages. If no arguments are specified, then this command displays a brief overview of the most commonly used commands, as well as what *groups* are available.

If a *group* is specified, all help topics within that group are displayed. If a *topic* (and optional *subtopic*) is specified, detailed help on that topic is displayed.

help is a probe-specific help command that currently differs from **help** through **MPserv**. For information on the **MPserv help** command, see the table in “Generic Debug Server Commands” on page 38.

Examples:

`help bs`: Displays help on the breakpoint set command.

`help set ip`: Displays help on setting the IP address.

info

Displays basic probe information.

jd *bits* [b | l] *data* [*rti*]

Performs a JTAG data scan. The state of the target CPU's internal JTAG state machine is moved to the `Shift-DR` state, and then *bits* bits of *data* are scanned in. Then the CPU's JTAG state machine is moved back to `Select-DR-Scan` after spending *rti* cycles in `Run-Test/Idle`. The default value for `rti` is `0`.

The optional arguments `b` and `l` specify that data scanning starts with the most or least significant bit respectively. This command prints the bits scanned out from the target in hexadecimal.

ji *instruction* [*rti*]

Performs a JTAG instruction scan. The CPU's internal JTAG state machine is moved to the `Shift-IR` state, and then the least significant bits of *instruction* are shifted in. Then the JTAG state machine is moved back to `Select-DR-Scan` after spending *rti* cycles in `Run-Test/Idle`. The default value for *rti* is `0`. This command prints the bits scanned out from the target in hexadecimal.

Examples:

`ji 0xff`: Selects the `BYPASS` chain on most targets.

jp [on | off | *pin_name*1 | 0]

Displays and controls JTAG pins. If no arguments are specified, then all pins and their current status are displayed.

Specifying `on` or `off` enables or disables (tristates) all interface pins. When the interface pins are disabled you can safely connect a different target board to the probe.

Specifying a *pin_name* together with a `1` or `0` drives individual pins high or low to test for shorts, or forces unspecified pins to a board-specific value.

Examples:

`jp`: Lists the current status of all JTAG pins.

`jp off`: Tristates JTAG pins.

`jp on`: Enables all JTAG pins.

`jp NRST 0`: Pulls NRST low.



Note The `USER` button emulates **jp on** and **jp off**. See "Step Three: Connecting the Probe to Your Target Board" on page 6.

jr

Resets the JTAG TAP controller.

ma *start_address end_address* [*permission*] [*match*] [*access_list*]

Defines debug memory access permissions starting at *start_address* (inclusive), and ending at *end_address* (inclusive). These permissions are used by all memory reads and writes (including software breakpoints), and can be used to disallow unintentional or incorrect memory accesses to memory and memory mapped peripherals. A dash, '`-`', can be used in place of a *start_address* to indicate that nothing is allowed. The default behavior is to disallow all access to the memory region.

The optional argument *permission* specifies any combination of `r`, `w`, and `x` to allow reading, writing, and executing.

The optional argument *match* is specified in the form: *value*/*mask*. The *mask* is ANDed with the memory access address and this result is compared with the *value*. If the comparison fails, no access is allowed. The default *match* is `0x0/0x0`.

The optional argument *access_list* specifies a comma-separated list of allowable access sizes in bits. An asterisk, '`*`', is used to indicate that all access sizes are permissible, while a 'b' is used to allow block accesses (which may use any access size).

Examples:

`ma`: Displays current debug memory access permissions.

`ma 0xz 0xh`: Prevents any access.

`ma 0x80000000 0x8001ffff rwx`: Specifies the memory range as regular RAM.

`ma 0xbf000c00 0xbf000cff rw 8`: Specifies a device which has 8-bit registers.

`ma 0xbf000d00 0xbf000dff rw 0x0/0x7 16`: Specifies a device which has 16-bit registers but only at 32-bit aligned address.

`ma 0xbf000e00 0xbf000eff rw 8,16`: Prevents 32-bit accesses.

md *start_address* [*length*]

Displays 64 bytes of target memory, beginning at *start_address*. The optional argument *length* can be used to vary the length of the memory read.

Examples:

`md 0xa0000000`: Reads 64 bytes starting at `0xa0000000`.

`md 0x1000 200`: Reads 200 bytes starting at `0x1000`.

mr [*access_size*] *start_address*

Performs a sized memory read, beginning at *start_address*

The optional parameter *access_size* specifies the size, in bytes, of the memory access. The default value is 4. If this parameter is specified then you must ensure that the *start_address* is aligned in a compatible manner.

Examples:

`mr 0xa0000000`: Reads 4 bytes of data from `0xa0000000`.

`mr 1 0x1000`: Reads 1 byte of data from `0x1000`.

mw [*access_size*] *start_address value*

Performs a sized memory write of the value, *value*, beginning at *start_address*.

The optional parameter *access_size* specifies the size, in bytes, of the memory access. The default memory *access_size* is 4. If this parameter is specified then you must ensure that the *start_address* is aligned in a compatible manner.

Examples:

`mw 0xa0000000 0x23`: Writes `0x00000023` to `0xa0000000`.

`mw 1 0x1000 0xfd`: Writes `0xfd` to `0x1000`.

reboot

Reboots the probe.

(Not available in the MULTI Debugger **Target** and **Command** windows)

restore

Restores the current configuration settings from non-volatile memory.

rr [*group* | *name*]…

Reads target registers. If no parameters are specified, then all the most commonly accessed registers are displayed. Alternatively, you may specify a particular register *name* (for example, `r0` or `pc`), or one of the following register groups:

- `gp` (general purpose)
- `spl` (special purpose)
- `fp` (floating point)
- `sys` (system)
- `db` (debug)
- `*` (all)

Examples:

`rr`: Displays all the most commonly accessed registers.

`rr fp`: Displays all floating point registers.

`rr *`: Displays all registers.

`rr r0 r1 r2 r3 pc`: Displays registers `r0`, `r1`, `r2`, `r3`, and the `pc`.

rw *name value*

Writes the value, *value*, to the register, *name*.

Examples:

`rw pc 0xa0000000`: Sets the `PC` to `0xa0000000`.

`rw r1 0x0`: Sets `R1` to `0x0`.

.

save

Saves the current configuration settings to non-volatile memory.

set [*option*[*value*]]

Queries and sets configuration options by name. If no parameters are given then all options and their current values are listed.

If only an *option* is specified, the value of the option is displayed.

If an *option* and a *value* are specified, the option is set to that value.

Use the **setup** command (below) to be guided interactively through setting all common options.

Examples:

`set`: Displays all the options.

`set ip`: Shows the value of the IP address.

`set netmask 255.255.255.0`: Sets the netmask to `255.255.255.0`.

setup [*group*]

Interactively guides you through setting the probe configuration options. If no parameters are given, the most common options are configured.

If you specify a *group*, then only the options in that group will be configured. The configuration groups are as follows:

- `net` (network)
- `trg` (target specific)

Examples:

`setup`: Configures the default group. This is the easiest way to configure a new probe out of the box.

(Not available in the MULTI Debugger **Target** and **Command** windows)

t *core_id*

Changes the current core id to *core_id*. All subsequent commands apply to the new core, until another **t** command is executed.

tc

Continues running the target from the current PC (program counter).

th

Halts a running target.

ti

Displays current target status information.

tr [d | r]

Resets the target system using JTAG reset pins.

If no parameters or d is specified, the target remains in debug mode immediately after the reset. For targets which do not support this, a halt request is immediately sent.

If `r` is specified, the target system runs freely after the reset.

Examples:

`tr`: Resets the target and keeps it in debug mode.

`tr r`: Resets the target and lets it run.

ts

Single steps one instruction on the target. On some architectures this may result in more than one instruction being executed. For instance, the branch delay slot may also be executed when single stepping a branch instruction on a MIPS target.

vb [*count*]

Verifies that the JTAG bypass register can be scanned properly. This test will scan several predefined patterns and then continue to scan in *count* number of random patterns. The default value of *count* is 1000.

This is a good first step to ensure that you have set up your system correctly and that the probe can communicate with the target.

Examples:

`vb 512`: Scans a test sequence and an additional 512 random values.

(Not available in the MULTI Debugger **Target** and **Command** windows)

vbp *start_address* [*count*]

Tests software execution breakpoint control by writing a simple test program to the target and executing it. This test verifies register and memory reads and writes between every breakpoint.

The program is written to *start_address*, a `1Kb` area of memory which can be read, written, and executed from and, if *count* is specified, will execute *count* times.

Examples:

`vbp 0xa0000000`: Tests breakpoints on a test program loaded at `0xa0000000`.

(Not available in the MULTI Debugger **Target** and **Command** windows)

vc *start_address* [*count*]

Tests single instruction stepping, breakpoints, and asynchronous run/halt control by writing a simple test program to the target and stepping through it. This test verifies register and memory reads and writes between every operation.

The program is written to *start_address*, a `1Kb` area of memory which can be read, written, and executed from and, if *count* is specified, will execute *count* times.

Examples:

`vc 0xa0000000`: Tests single instruction stepping, breakpoints, and asynchronous run/halt control on a test program loaded at `0xa0000000`.

(Not available in the MULTI Debugger **Target** and **Command** windows)

vlb

Verifies that the internals of the Target Interface Unit are working correctly.

Examples:

`vlb`: Runs the loop back test.

(Not available in the MULTI Debugger **Target** and **Command** windows)

vm [*access_size*] *start_address length*

Tests memory by writing a random sequence of values to the specified area, beginning at *start_address*, and continuing for *length*, and then reading it back, comparing the values.

The optional argument *access_size* specifies the size, in bytes, of memory access. The default value is the target's general purpose register size (which, in most cases, is 4).

Examples:

`vm 0xa0000000 0x10000`: Tests memory `0xa0000000` to `0xa0010000`.

`vm 1 0x0 0x8000`: Tests byte access `0x0` to `0x8000`.

(Not available in the MULTI Debugger **Target** and **Command** windows)

vrh *start_address* [*count*]

Verifies asynchronous run/halt control by writing a simple test program to memory and executing it. This test verifies register and memory reads and writes between every run and halt sequence.

The program is written to *start_address*, a `1Kb` area of memory which can be read, written, and executed from and, if *count* is specified, will execute *count* times.

Examples:

`vrh 0xa0000000`: Tests run/halt control on a test program loaded at `0xa0000000`.

`vrh 0x8000 200`: Tests run and halt through a test program loaded at `0x8000`, 200 times.

(Not available in the MULTI Debugger **Target** and **Command** windows)

vsi *start_address* [*count*]

Tests single instruction stepping by writing a simple test program to the target and stepping through it. This test verifies register and memory reads and writes between every step.

The program is written to *start_address*, a `1Kb` area of memory which can be read, written, and executed from and, if *count* is specified, will execute *count* times.

Examples:

`vsi 0xa0000000`: Single steps through a test program at `0xa0000000`.

vta

Verifies that the correct Target Adapter is connected and that all required pins can be controlled and monitored. You may have to disconnect the ribbon cable from the target system for accurate results.

Examples:

`vta`: Tests the adapter pins.

(Not available in the MULTI Debugger **Target** and **Command** windows)

w

Displays a list of all active connections to the probe. For each connection, the following information is shown:

- the connection type
- if it is an ethernet connection, the IP address of the client
- the number of seconds since the user last used the connection

(Not available in the MULTI Debugger **Target** and **Command** windows)

Generic Debug Server Commands

The following commands are available to most Green Hills debug servers. In addition to these generic commands, **MPserv** also accepts the commands listed in “Green Hills Probe-Specific Commands” on page 24.

addressof *symbol*

Prints and returns the address of *symbol*. This command requires that an image be loaded in the MULTI Debugger.

amask [*mask*] [*value*]

If *mask* and *value* are specified: Sets the download mask and value to *mask* and *value*, respectively. When **MPserv** downloads a program, it will bitwise AND *mask* and bitwise OR *value* with the download addresses, as follows:

```
address = (address & mask) | value
```

This command is useful for shifting download target addresses without relinking a program. However, the program being downloaded will still retain its original relocations and may not run correctly at its new destination address without further support on the target.

If no arguments are specified: Displays the current settings of the download mask and value. The default settings of *mask* and *value* if they are not specified are `0xffffffff` and `0`, respectively.

close *fd* Closes the specified file descriptor.
debug *n* Sets the debugging bit flags.
echo [on | off] If **on** or **off** is specified: Toggles printing of commands executed in a script. If no argument is specified: Prints and returns the current echo mode.
fprint *fd string* Prints the string to the specified file with script variable expansion.
fprintb *fd integer* Prints the integer to the file in binary mode.
fread *fd identifier* Reads one line of text from the specified file. The line is stored as a variable with the specified identifier. The number of bytes read is returned. If there is an error, `-1` is returned.
freadb *fd identifier* Reads an integer from the specified file in binary mode. The integer is stored as a variable with the specified identifier. The number of bytes read is returned. If there is an error, `-1` is returned.
getenv *envName identifier* Stores the value of the environment variable *envName* in the script variable with the specified identifier.
halt Halts execution of the target CPU and forces it into debugging mode.

help [*command* | *group*]

If no argument is specified: Prints general help information and instructions for finding more detailed and specific help.

If an argument is specified: Prints help information about the specified *command* or *group* of commands. Valid command groups include **target**, **server**, and **scripting**. For a command group, **help** prints the commands in the group with information on the arguments each command takes. For a specific command, **help** prints more detailed information about that command.

Example 1:

```
help server
help [<command> | <group>]
debug <n>
playdead
```

Example 2:

```
help help
help [<command> | <group>]
Prints help information
```

listvars

Prints all variable identifiers in no particular order.

Example:

```
str1="foo"
str2="bar"
i=100
listvars
i
str1
str2
```

load [all | text | data | bss]

If no argument is specified: Displays the current **load** settings.

If an argument is specified: Specifies which sections to include in host-to-target downloads. You can combine this command with the **noload** command.

`.text` sections contain code, `.data` sections contain initialized variables, and `.bss` sections contain uninitialized data.

Setting the `all` option includes all of these sections in the download. Note that `.bss` sections are cleared by all programs compiled with Green Hills tools, so you do not need to download them.

Standard Example:

```
load all
Download Options: text data bss
```

Advanced Example:

```
load all noload bss
Download Options: text data
```

m [-d*size*] *address* [=*val*]

If *val* is not specified: Reads the specified memory address on the target.

If *val* is specified: Writes *val* to the specified memory address on the target.

Memory addresses and values must be specified in hexadecimal (with or without preceding `0x`). The **-d** argument can be used to set the access size to byte (**-d1**), short (**-d2**), or long (**-d4**). The default is **-d4**.

Examples:

```
m 1000
7ca62b78
m 1000=12345678
m -d2 1000
1234
```

nofail *command*

Executes the specified command and always returns success.

noload [all | text | data | bss]

If no arguments are specified: Displays the current **noload** settings.

If an argument is specified: Specifies which sections to exclude from host-to-target downloads.

`.text` sections contain code, `.data` sections contain initialized variables, and `.bss` sections contain uninitialized data.

Use the command **noload bss** to exclude `.bss` sections from downloads. These sections are cleared to all zeros by programs compiled with Green Hills tools, so downloading them to the target is usually unnecessary.

open *file*

Opens the specified file for writing and returns a file descriptor.

pid [*pid*]

If *pid* is specified: Sets the PID that will be used for subsequent direct commands. This command is useful for addressing direct commands to a particular core on a multi-core system.

If *pid* is not specified: Displays the PID that will be used for subsequent direct commands.

print *string*

Prints the specified string with script variable expansion.

Example:

```
print Test
Test
```

random *max*

Generates and returns a pseudo-random number between `0` and *max*`-1`.

regnum *reg* [=*val*]

If *val* is not specified: Reads the specified register.

If *val* is specified: Writes *val* to the specified register.

Registers are specified by MULTI register number and values and should be entered in hexadecimal with no leading `0x`.

Example:

```
regnum 0 = deadbeef
regnum 0
Register 0 = deadbeef
```

run [*address*]

If *address* is specified: Sets the PC to *address* and then runs the processor.

If *address* is not specified: Runs the processor from the current PC.

Use this command very carefully. It is possible to run a program on the board when MULTI thinks it is halted, which causes unpredictable results. A common use for this command is to run a ROM monitor program so that the monitor may set up the board properly before MULTI downloads a program.

script *file*

Executes the commands in the specified script file as if they were typed in line by line.

Example:

```
script crc32.scr
```

setup *file*

Specifies a script file to be automatically run immediately prior to every host-to-target download.

Example:

```
setup mbx800.dbs
```

sleep *seconds*

Suspends the debug server for the specified number of seconds.

status

Prints and returns the current status of **MPserv** using the following status codes:

0 Running

1 Stopped by breakpoint

2 Single step completed

3 Exception

4 Halted

5 Process exited

6 Process terminated

7 No process

8 (This code is unassigned.)

9 Stopped by hardware breakpoint

10 Failure

11 Process ready to run

12 Host system call in progress

13 Target reset

step

Single steps the target CPU from the current program counter location.

undef *variable*

Removes the specified variable and releases any memory associated with it.

Example:

```
x=5
undef x
print $x
Error: variable undefined!
```

MPserv Scripting

In addition to the commands listed in the previous section, a full scripting language is available from the **MPserv** command line. You can run scripts by:

- Entering scripts one line at a time at the command prompt. When typing a script line by line, nothing will be executed until the top-level enclosing **while** or **if** statement is terminated.
- Storing commands in a script file and then running the file using the **script** command. You can also use the **-setup** option to ensure that a script file is automatically executed immediately prior to every host-to-target download.

General Notes

When using the **MPserv** scripting language, keep the following in mind:

- There cannot be more than one statement per line.
- Any line that has the # character as the first non-whitespace character is treated as a comment.
- Variables do not need to be declared before they are used.
- Variable types are determined automatically. For example, an identifier that is bound to an integer variable can later be assigned to a string variable.
- Variable and function names are not case sensitive.

Scripting Syntax

The following sections describe and give examples of some features of **MPserv** scripting.

Expressions

Expressions in the **MPserv** scripting language are similar to C-language expressions. The following table contains, in order of precedence, the valid operators you can use in expressions. Note that a string is treated like an integer if it contains a string representation of an integer.

Operator	Integer Function	String Function
(	Groups operators to ensure desired evaluation	Groups operators to ensure desired evaluation
*	Multiplication	Invalid
/	Division	Invalid
%	Modulus	Invalid
+	Addition	Concatenation
-	Subtraction	Invalid
<<	Bitwise left shift	Invalid
>>	Bitwise right shift	Invalid
<	Less than	Alphabetic less than
<=	Less than or equal to	Alphabetic less than or equal to
>	Greater than	Alphabetic greater than or equal to
>=	Greater than or equal to	Alphabetic greater than or equal to
==	Equality test	Equality test
!=	Inequality test	Inequality test
&	Bitwise AND	Invalid
^	Bitwise XOR	Invalid
|	Bitwise OR	Invalid
&&	Logical AND	Invalid
||	Logical OR	Invalid

Assignments

The syntax for an assignment is:

identifier = expression

The expression is evaluated and the result is stored as a variable with the given identifier. String, integer, and array variables are supported. Identifiers can contain alphanumeric characters and the _ character, but are not permitted to begin with a number.

Arrays

Arrays are indexed lists of variables. Each cell in an array can contain a string, integer, or array. Array indexing begins with the index `0`. An array can be created by assigning an entire array to an identifier or by assigning a string, integer, or array to one cell of an array. To reference a cell, follow the array identifier by the index contained in square brackets. If an array cell contains another array, the elements in the second array are accessed by appending an additional index in square brackets. The following example demonstrates the various methods of array access.

```
endl="\n"
bar[3] = 42
foo = { bar, 7, "hello" }
print $foo[2] world.$endl
if(foo[0][3]==bar[2+1])
    print Array indexing works.$endl
endif
```

Arrays are dynamically allocated in a sparse fashion. For example, making an assignment to `foo[0]` and then to `foo[100]` only allocates two array cells, and no space is used for the undefined array cells 1 through 99. Once an array element has been allocated, it cannot be deallocated without using the **undef** command on the top-level array identifier.

Conditionals

The following table explains the syntax for conditionals.

Syntax	Effect
if *expression* *statements* **endif**	If the expression evaluates to zero, nothing happens. Otherwise, the block of statements between the **if** and **endif** lines will be executed.
if *expression* *statements* **else** *statements* **endif**	If the expression evaluates to zero, **MPserv** will execute the block of statements between the **else** and **endif** lines. Otherwise, the block of statements between the **if** and **else** lines will be executed.
if *expression* *statements* **elif** *expression* *statements* **endif**	If the expression in the **if** statement does not evaluate to zero, **MPserv** will execute the block of statements between the **if** and **elif** lines. Otherwise, if the expression in the **elif** statement does not evaluate to zero, **MPserv** will execute the block of statements between the **elif** and **endif** lines. If both expressions evaluate to zero, **MPserv** will not execute anything.

Loops

The following table explains the syntax for loops.

Syntax	Effect
while *expression* *statements* **endwhile**	The statements between the **while** and **endwhile** lines will be executed as long as the expression in the **while** statement does not evaluate to zero. The expression is evaluated before the loop is executed for the first time. Be careful to avoid infinite loops. If an infinite loop occurs, you must shut down and restart **MPserv**.

Variable Expansion

To use script variables as arguments to **MPserv** commands, you must prepend the variable name with one or two $ characters.

- To pass a variable to a command in its default text representation, prepend the variable name with a single $ character. This passes a decimal string for integer variables and passes the string itself for string variables. An entire array cannot be given as an argument to a command.

- To pass an integer variable to a command as a hexadecimal string, prepend the variable name with two `$` characters. Use this method with commands such as **m** that require arguments in hexadecimal form.

Variable expansion must be done in an unambiguous way. The script parser attempts to use the longest legal identifier name following the `$` character. In the following example, the user has attempted to print the string `bar` after the expansion of the variable `foo`. The parser interprets this as printing the value of the variable `foobar` and reports that the variable is not defined:

```
foo="foo"
print $foobar
Error: variable undefined!
```

Example Scripts

You can use the following examples of the **MPserv** scripting language as a guide when writing your own scripts.

Example 1.

For this example, assume that the file **test.ascii** contains the following text:

```
This is a test of script file access in ascii mode.
This should print 3 lines of which this is the second.
And this is the third.
```

The following script commands access the file **test.ascii**:

```
file = open test.ascii
filecontents=""
totalchars=0
while(numlinechars=fread file line)
filecontents=filecontents+line
totalchars=totalchars+numlinechars
endwhile
close file
endl="\n"
print Read: $filecontents$endl
print Total of $totalchars characters read.$endl
```

The output of this example script on a UNIX host is:

```
Read: This is a test of script file access in ascii mode.
This should print 3 lines of which this is the second.
And this is the third.

Total of 130 characters read.
```

Example 2.

The following script is another example of accessing a file:

```
i=100
file = open temp.bin
while(i>0)
fprintb file $i
i=i-1
endwhile
close file
file = open temp.bin
sum=0
while(freadb file i)
sum=sum+i
endwhile
close file
endl="\n"
print The numbers between 1 and 100 sum to $sum!$endl
```

The output of this example script is:

```
The numbers between 1 and 100 sum to 5050!
```

Example 3.

The following script calculates the CRC32 value of the memory range `0x010000` - `0x010100` and prints the result in the **Target** window. This script demonstrates the use of loops, conditional statements, expressions, variables, and other debug server scripting constructs in a real application.

```
# Change the following values to specify the memory
# range you want to calculate a CRC32 for.
# Note: locations from memstart to memend-1 are used
# to compute the CRC32 value.
memstart=0x010000
memend=0x010100
# This is the CRC32 polynomial. This is the same as
# is used in ethernet packets.
p=2+4+16+32+128+256+1024+2048+4096+65536+4194304+8388608+67108864
r=0
ptr=memstart
while(ptr<memend)
currbyte=m -d1 $$ptr
currbit=128
while(currbit)
test=r&(1<<31)
r=r<<1
r=r|(currbyte&currbit)
if(test)
r=r^p
endif
currbit=currbit>>1
endwhile
ptr=ptr+1
endwhile
# This loop is for the 32 zeros appended to the
# original memory contents
i=0
while(i<32)
test=r&(1<<31)
r=r<<1
if(test)
r=r^p
endif
i=i+1
endwhile
# Now the resulting 32 bit CRC is in r
# Many of the same ASCII control codes that are used
# in C are supported in debug server scripts.
endl="\n"
print CRC32 = $$r$endl
```

Chapter 5

Advanced Configuration

This Chapter Contains:

- Introduction
- Setting the IP Address
- Setting the Netmask
- Setting the Gateway
- Setting the Adapter Type
- Configuring Your Target
- Setting the Serial Communication Speed (Baud Rate)
- Turning the Status Checker On or Off
- Setting the JTAG Clock Speed
- Setting Debug Message Generation
- Setting Override of Sanity Checks On or Off
- Turning Target Power Detection On or Off
- Setting the User String
- Setting the Reset Pulse Length
- Setting the JTAG TAP Reset Pulse Length
- Setting the Reset Settle Length
- Setting the JTAG TAP Reset Settle Length
- Setting Reset Handshaking Timeout
- Setting the Endianness
- Target-Specific Options

Introduction

This section describe configuration options that can be changed to customize your debug requirements. Most of these options can be configured interactively by executing the **setup** command from a terminal (see “Step Four: Configuring the Probe” on page 6 in Chapter 2, “Quick Start Guide”).

All these options can be displayed and configured by the **set** command. (see Chapter 4, “Green Hills Probe Command and Scripting Reference”).



Note Some options, such as those relating to network configuration, require the probe to be rebooted before they will take effect.

Setting the IP Address

Format:

`set ip` *ip*

Sets the IP address of the probe to *ip*, which is specified in dotted quad notation. This setting does not take effect until the next reboot.

Example 1.

```
set ip 192.168.0.5
```

Sets the IP address to `192.168.0.5`.

Setting the Netmask

Format:

`set netmask` *netmask*

Sets the netmask for the probe’s ethernet connection to *netmask*, which is specified in dotted quad notation. Ask your system administrator if you are unsure what this value should be. This setting does not take effect until the next reboot.

Example 2.

```
set netmask 255.255.255.0
```

Sets the netmask to `255.255.255.0`.

Setting the Gateway

Format:

`set gateway` *gateway*

Sets the gateway for the probe's ethernet connection to *gateway*, which is specified in dotted quad notation. This setting does not take effect until the next reboot.

Example 3.

```
set gateway 192.168.0.1
```

Sets the gateway to `192.168.0.1`.

Setting the Adapter Type

Format:

`set adapter` *adapter_type*

Sets the JTAG adapter currently plugged into the probe. Different CPU Adapters are needed to meet the electrical specifications of the wide variety of JTAG/BDM interfaces on target boards. The default setting of **auto** automatically detects what kind of adapter is plugged in. You should not need to change this setting.

The following table lists the main CPU adapter types and the targets for which they are typically used:

adapter_type	**Target**
auto	The probe will automatically determine the adapter type. This is the recommended default.
arm-20	ARM targets using the 20-pin debug connection.

adapter_type	**Target**
arm-14	ARM targets using the older 14-pin debug connection.
ejtag25	MIPS targets using the 14-pin EJTAG v2.5 or v2.6 connection.
ejtag20-20	MIPS targets using the 20-pin EJTAG v2.0 or v1.53 connection.
ejtag20-12	MIPS or Lexra targets using the 12-pin EJTAG v2.0 or v1.53 connection.
cop	PowerPC targets using the 16-pin COP connection. Examples include the PowerPC 82*xx* series CPUs.
eonce	Star*Core targets using the 14-pin EOnCE connection.

Configuring Your Target

For single-core operation, enter:

```
set multicore off
set target device_name
```

For multi-core operation, enter:

```
set multicore on
set chain device_name [device_name] [device_name]
```

where, for either operation, *device_name* consists of any one of the following target-specific devices:

Target Family	Target	Supported Device Name
MIPS Family	MIPS32, MIPS64	**mips32_4kc**, **mips32_4kp**, **mips32_4km**, **mips32_4kec**, **mips32_4kep**, **mips32_4kem**, **mips32_4ksc**, **mips64_5kc**, **mips64_5kf**, **mips64_20kc**, **ejtag20**, **ejtag25**, **ejtag26**
	IDT323xx	**idt323xx**, **idt32355**, **idt32334**, **idt32364**
	Broadcom 1250	**bcm1250**
	Broadcom 6352/3345	**bcm3345**, **bcm6352**
	Philips 3940/1900	**pr3940**, **pr1900**
	LSI Logic	**lsi4102**
	Alchemy/AMD	**au1000**, **au15000**

Target Family	Target	Supported Device Name
ARM Family	ARM	**arm7tdmi**, **arm720**, **arm9tdmi**, **arm920**, **arm940**, **arm966**
	XScale	**xscale**
Lexra Family	Lexra	**lx4280**
PowerPC Family	PowerPC 82xx	**ppc8240**, **ppc8260**
	PowerPC 8xx/555	**ppc8xx**, **ppc860**, **ppc855**, **ppc823**, **ppc555**
Star*Core Family	Star*Core	**msc8101**
Wintegra Family	Wintegra	**winpath**
ZSP Family	LSI Logic	**zsp**

Setting the Serial Communication Speed (Baud Rate)

Format:

`set baudrate` *baudrate*

Sets the serial communications speed to *baudrate*. This setting takes effect immediately. If this is typed from a serial console, you must reconfigure your communications software immediately to the new baud rate.

Turning the Status Checker On or Off

Format:

`set checker on|off`

Turns the status checker on or off. When turned on, the probe will continually poll the target's status and report changes.

If this option is not set, the **ti** command can be used to request the current target status. MULTI is not dependent on this configuration option.

Setting the JTAG Clock Speed

Format:

`set clock` *speed*

Sets the JTAG clock speed in Hz. Faster clock speeds will result in higher performance. If this setting is too high then reliability will suffer. If you are seeing unexplained problems, the clock speed may be set too high.

Example 4.

```
set clock 10000000
```

Sets the JTAG clock speed to 10MHz.

Setting Debug Message Generation

Format:

`set debug` *group*

Specifies the *group* of debug messages that are generated. *group* is a bit mask which may have the following values:

- 1: protocol
- 2: target
- 4: communications
- 8: FPGA
- 16: verbose protocol
- 32: error explanations
- 64: Probe configuration

By default, debug messages are output to the serial port. To redirect them, use the **debug** command.

Setting Override of Sanity Checks On or Off

Format:

```
set override on|off
```

Turns sanity checking on or off. When override is set on, the probe will execute most commands as given and not check for unexpected situations such as trying to read registers from a running target. This should remain set to **off** during normal operation.

Turning Target Power Detection On or Off

Format:

```
set power_detect on|off
```

Turns target power detection on or off. When turned on, the status checker will try to determine whether the target has power or not.

Setting the User String

Format:

`set user_string` *string*

Sets the user string of the probe to *string*. The user string is used to name and identify a probe, and appears on the startup screen and web page.

Setting the Reset Pulse Length

Format:

```
set rst_pulse
```

Specifies the minimum time in milliseconds the nRST line will be held low during a reset. This may need to be adjusted from the default if your target needs a longer result pulse to reset the target. See the option **rst_handshake_timeout** option in “Setting Reset Handshaking Timeout” on page 60.

Setting the JTAG TAP Reset Pulse Length

Format:

`set jrst_pulse` *milliseconds*

Specifies how long the nTRST line will be held low during a JTAG TAP reset. This is used exclusively to reset the JTAG TAP controller and not the target This may need to be adjusted from the default if your target needs a longer result pulse to reset the JTAG TAP controller.

Setting the Reset Settle Length

Format:

`set rst_settle` *milliseconds*

Specifies how long to wait after a nRST pulse is issued before scanning any commands into the target. You may need to adjust this if your target needs more time to stabilize before issuing commands.

Setting the JTAG TAP Reset Settle Length

Format:

`set jrst_settle` *milliseconds*

Specifies how long to wait after a nTRST pulse is issued before scanning any commands into the target. You may need to adjust this if your target needs more time to stabilize before issuing commands.

Setting Reset Handshaking Timeout

Format:

`set rst_handshake_timeout` *timeout*

Newer MIPS targets include a handshaking protocol for performing reset. This option sets the amount of time by which the standard reset sequence may be extended, waiting for this handshake to complete. Setting this option to zero results in no handshake attempted.

The **rst_pulse** and **rst_settle** times will always be kept as minimum times for all targets. To disable the reset sequence, set the **rst_settle**, **rst_pulse**, and

the **rst_handshake_timeout** to zero; never pulse the nRST line when the **rst** command is issued.

Setting the Endianness

Format:

`set endianness` *mode*

Indicates to the Probe the target's endianness, where *mode* is either **big** for Big endian or **little** for Little endian. See the **de** and **detect** commands in "Detailed Syntax and Examples" on page 26.

Target-Specific Options

ARM

The following commands apply to arm7tdmi, arm720, and arm9tdmi.

Option	Description
abort_check	When turned on, the Probe will do extra work before and after each debug mode memory access to attempt to determine if a data abort (access to invalid memory) occurred during the access. Turning this off can speed memory access marginally, but disables the Probe's ability to check for successful completion of memory reads and writes. The default is on.
catch_abort, **catch_fiq**, **catch_irq**, **catch_prefetch**, **catch_reset**, **catch_swi**, **catch_undef**	For ARM9 and XScale. These parameters control the exception vector catching mechanisms built into the ARM9, ARM9E, and XScale processors. Specifically, these control the catching of the data abort, fast interrupt exceptions, interrupt exceptions, prefetch abort, reset, software interrupt, and undefined instructions, respectively. When turned on, the processor will stop at the exception vector, after switching to the appropriate exception mode, whenever one of these exceptions is triggered, which enables relatively easy debugging of the exception handler. When turned off, the vectors will be taken without interruption.

Option	Description
fast_dl	When turned on, the Probe will assume that the target can complete memory writes as fast as it can issue them. Turning this setting on will increase memory write speed by up to 300% in some cases, but can cause incorrect operation of targets that have slow CPUs or slow memory. The default is `off`.
handler_base	For XScale only. The base address of the debug handler used to debugging the XScale. The Probe requires a small section of memory approximately 2 Kbytes in size, aligned on a 2 Kbyte boundary. The starting address of this section is specified by the **handler_base** setting. The address does not have to correspond to any real memory in the system, but it should not overlap with any addresses used, either for code or data, by your application, code in flash ROM, or any memory-mappedI/O devices in the system.

MIPS

The following commands apply to ejtag20, ejtag25, idt323xx, and lsi4120.

Option	Description
bp_in_delay_slots	Sets the behavior for breakpoints in delay slots. When turned on, breakpoints in delay slots are handled the same as any other breakpoint. The PC will appear on the branch before the delay slot. When turned off, breakpoints in delay slots are ignored. When the Probe hits one it will be removed, and the programming will keep running. This is necessary to avoid confusion when hitting a breakpoint in a delay slot.

dbscratch_addr, **dbscratch_size**	For idt323xx only. IDT323xx processors currently need a scratch area to successfully debug code in cached memory. The scratch area needs to be at least 16 bytes and 4-byte aligned. You need to set the size and location of the scratch area with the **dbscratch_addr** and **dbscratch_size** configuration options. For example:
	
	`idt323xx[h] % set dbscratch_addr 0x80800000`
	`idt323xx[h] % set dbscratch_size 0x10`
	
	Care must be taken that your application does not use the debug scratch memory.

Option	Description
fast_dl	As described for the ARM **fast_dl** option above. The common case for MIPS targets where **fast_dl** must be off is for those targets whose core is synthesized into an FPGA or other programmable logic device. The default is on (which is unsafe.)
step_ints	Sets single stepping behavior when interrupts are enabled. When turned on, interrupts will be disabled on the target during a single step.

PowerPC

The following commands apply to ppc8240 and ppc8260.

Option	Description
immr_base	For ppc8260 only. The base address of the internal memory-mapped register set upon reset of the target. To find this address, consult your board or processor manual. This setting is essential for proper operation of PowerPC 8260 targets.

Option	Description
step	Sets the step method, using one of the following three arguments: **swbp**: sets a software breakpoint at the next address. This works as long as the code you are debugging is in memory that can be written. It will fail if the code resides in ROM or flash. **trace**: uses the chip's built-in single-step functionality. This does not properly single-step over all instructions. It fails for instructions that change the **MSR** and **mtctr** instructions. **hwbp**: sets a hardware breakpoint at the next address. This fails in certain cases just after a branch has been taken, but does allow ROM debugging.
swbp_type	For PowerPC 88x/555. Sets the software breakpoint method. There is no single interrupt that is guaranteed to be available to the Probe for software breakpoints, so you should choose one that does not cause a conflict with the application running on the target. Specifically, this setting directly affects some of the bits in the DER register. **swbp_type** uses one of the following three arguments: **illegal**: Uses the software emulation interrupt to cause software breakpoints by writing an **illegal** instruction to memory where software breakpoints are used. Sets SEIE in DER. **syscall**: Uses the system call interrupt to cause software breakpoints by writing the **syscall** instruction to memory where software breakpoints are used. Sets SYSIE in DER. **trap**: Uses the program interrupt to cause software breakpoints by writing a **trap** instruction to memory where software breakpoints are used. Sets PRIE in DER.

Star*Core

The following commands apply to the msc8101.

Option	Description

Option	Description
immr_base, **sypcr_write_enable**, **sypcr_value**	The MSC8101 processor includes an internal watchdog timer, and in the default configuration on the ADS8101 board (the most common evaluation board for this processor) debugging is difficult, unless the watchdog timer is disabled on reset. However, disabling the watchdog timer involves writing to the SYPCR system protection register, which can only be written to once after each target reset. These three configuration values enable the probe to find the SYPCR register and write a value to it just after reset. The default configuration (`immr_base=0xf0000000`, `sypcr_write_enable=on`, and `sypcr_value=0xffffffc1`) causes the Probe to disable the ADS8101 watchdog timer just after target reset. If **sypcr_write_enable** is turned on, the Probe will write the value specified in **sypcr_value** to `immr_base+0x1004` after waiting at least **rst_settle** milliseconds following the rising edge of NRST to release the 8101 from reset.
accelerated_download	When turned on, the probe uses a small target-resident agent to speed up downloads to 8101 targets by a factor of about 10. Although this almost never causes problems on the target, setting it to `off` will disable this behavior. The default is on.

Chapter 6

Web Interface Reference

This Chapter Contains:

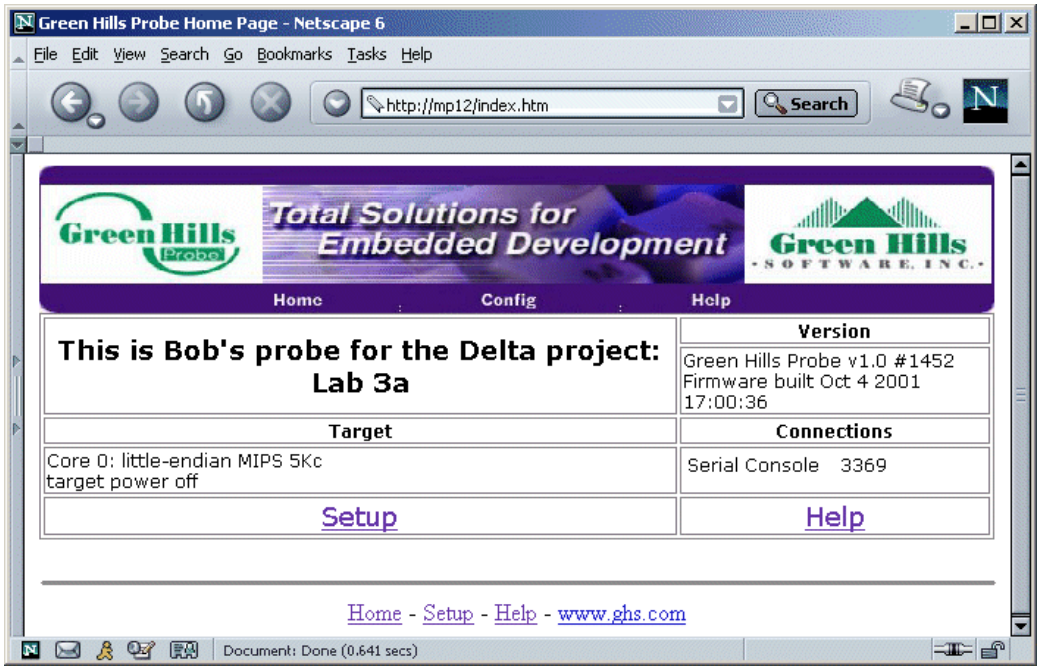
- Introduction
- Main Page
- Setup Pages
- Help Pages

Introduction

The Green Hills Probe provides a web interface for probe configuration, as well as status and visibility of probe, connection, and target values. This interface is accessible through the Ethernet connection using popular web browsers. The web interface provides a graphical alternative to much of the functionality available in the terminal interfaces.

The web pages shown below represent the probe's initial release. The appearance of web pages from your probe is likely to differ somewhat. At a minimum, there will be a main page, pages for probe setup and configuration, and pages for online help.

Main Page

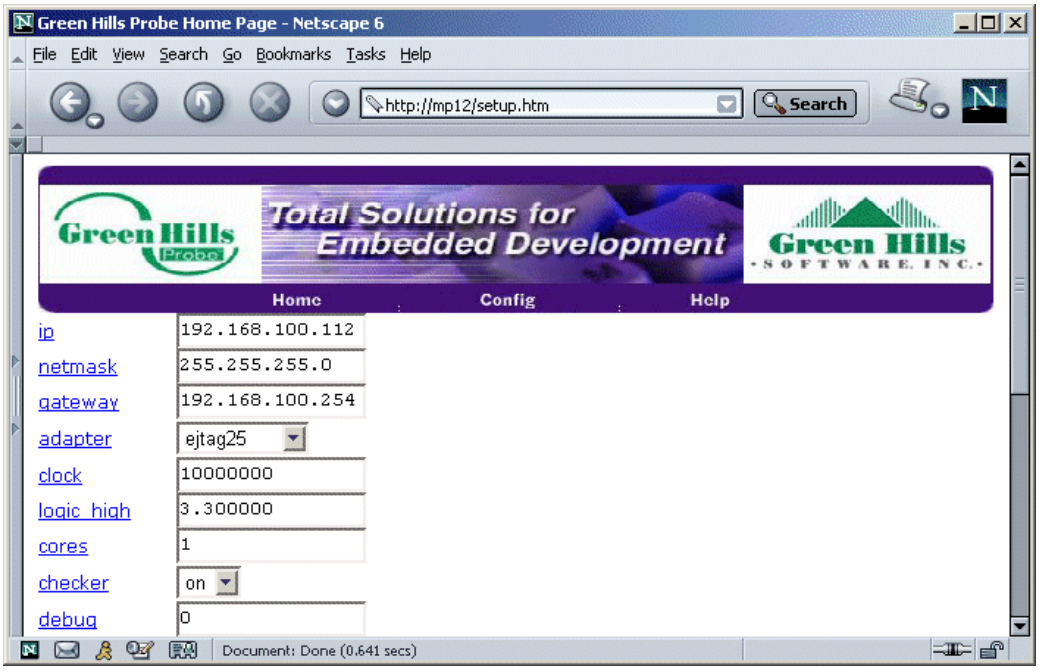


This is an example of the main (**index.htm**) page presented by the probe when you point a web browser at its IP address. This page displays the following information:

- The probe's firmware revision and date
- The user banner (this can be used to personalize your probe)
- The status and type of the target CPU

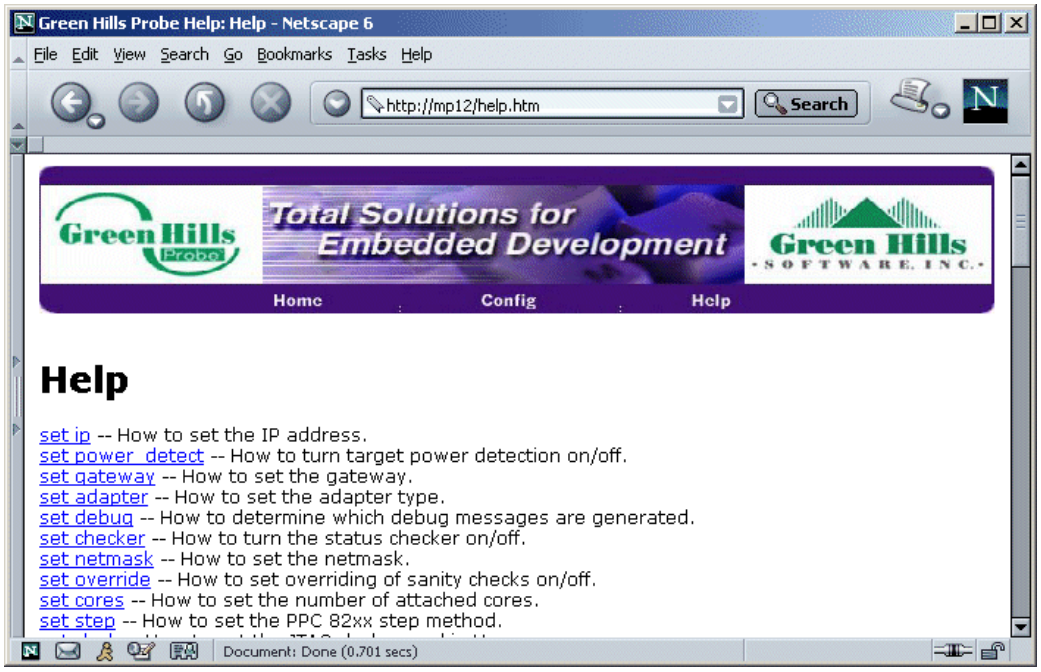
- Links to the setup and help pages, and to the Green Hills home page.

Setup Pages



This page represents a graphical alternative to some of the commands available through the terminal interface. Values can be edited and saved to non-volatile probe memory by pressing the **Submit** button. Help on any individual configuration item can be reached by clicking the item name in the web page.

Help Pages



This page shows a help index for the probe. Detailed help on any command can be displayed by clicking the item of interest in the web page display.

Chapter 7

Utilities and Troubleshooting

This Chapter Contains:

- Updating the Probe Firmware
- Troubleshooting

Updating the Probe Firmware

The Green Hills Probe firmware can be updated to provide new CPU support, new features, and enhanced performance. The firmware can be updated quickly through the Serial, USB, or Ethernet ports by the **MPadmin** utility application.

To update the probe by Ethernet or USB, a valid image must be present and running on the probe to accept the update request. If for some reason the probe does not boot, such as if power was cycled while programming the boot flash, the firmware can still be reprogrammed through the serial port. **MPadmin** will prompt you to power the Green Hills Probe with the USER button pressed. (For more information on the USER button, see “Step Three: Connecting the Probe to Your Target Board” on page 6. As soon as the amber light appears on the front of the probe you may release the button. This will cause the probe to boot from code received on the serial port, allowing the unit to be programmed even if there is not a valid image present on the probe. When the programming is completed, power cycle the probe.



Note After updating the firmware and before attempting to connect through **MPserv**, you should ensure that this update did not change or erase the probe settings. You can check your probe settings with the **MPadmin** Utility or through a probe console.

The MPadmin Utility

The **mpadmin** utility is invoked from the command line.

Format:

install_dir/**mpadmin** [*options*] *connection* [*firmware_file*]

Where *options* are one or more of the following:

Option	Meaning
-update	Updates the probe firmware.
-v	Runs the update in verbose mode.
-cfgload	Load configuration information from a file into the probe.
-cfgsave	Save configuration information from the probe into a file.

The *connection* parameter should be specified as follows:

Connection Type	Enter
Ethernet	Enter your probe name or IP address (in dotted quad form).
USB	**-USB** [*usb index*]. [usb index] = 1 if not specified. Multiple probes are enumerated in the order they were attached to the system.
Serial	**-serial** *port* [**-baud** *baudrate*]

The *firmware_file* is the firmware image file supplied.by Green Hills Software for the probe.

Example 1. Using mpadmin

The command line:

```
mpadmin -update ghs_probe3 firmware.frm
```

performs an ethernet update to the probe with ethernet host name `ghs_probe3`.

The command line:

```
mpadmin -update -usb firmware.frm
```

updates the first enumerated USB probe.

The command line:

```
mpadmin -update -usb 3 firmware.frm
```

updates the third enumerated USB probe.

The command line:

```
mpadmin -update -serial com1 firmware.frm
```

loads the firmware file `firmware.frm` into a probe connected to the host's COM1 serial port.

The command line:

```
mpadmin -setup 192.168.17.12
```

runs an interactive setup with the probe at ethernet address.`192.168.17.12`

The command line:

```
mpadmin -cfgload probe11 mysetup.cfg
```

loads the probe configuration file `mysetup.cfg` into the probe with ethernet hostname `probe11`.

Troubleshooting

My Green Hills Probe is not booting properly

Is the Power LED Illuminated?

Ensure the power supply is connected to the probe and the power switch is in the On position. When power is supplied, the green power LED on the front of the unit should illuminate.

If the light does not illuminate check the power switch position, turn it off and then back on again. If still unsuccessful you should try a different power outlet and/or supply. The supply should be between 7 and 15 Volts, center positive or negative, providing at least 500mA.

If you are still unable to get the light to come on you should contact Green Hills Support for further instructions.

Does the Probe Pass the First Stage of the Boot Sequence?

When the Probe Boots from a power cycle, the **RST**, **HALT**, and **RUN** LEDs on the front panel should come on for about two seconds and then go off again. If this does not happen, check the power supply and ensure it meets the requirement mentioned above.

Does the Probe Pass the Second Stage of the Boot Sequence?

Shortly after the first stage, the probe will print `GHS:Starting` to the serial port at `9600-8n1`, and illuminate the **RST** and **HALT** light for about a second. If nothing comes out of the serial port, the probe might not have a valid firmware image loaded.

Try:

- Using a different COM port on your host computer.
- Using a different serial cable (this must be a NULL modem cable like the one that came with your probe).

- Loading a new firmware image (see “Updating the Probe Firmware” on page 72).

If the banner still does not appear, contact Green Hills Support.

My Green Hills Probe is not Communicating Properly with my Host Machine via:

RS-232 Serial Port

Is the Banner only Partially Printed?

If only part of the banner is printed this most likely means that the power on self test fatally failed. Try to load a new firmware image (see “Updating the Probe Firmware” on page 72).

If the problem persists, contact Green Hills Support.

Is Random Data Printed to your Terminal after the Correct Serial Port Banner?

The Probe’s baud rate can be changed to support the requirements of different boards. It will always boot at 9600 and allow you to press the space bar before changing the baud rate. It will display which baud rate it will change to before setting it. You should change the baud rate of your terminal program to match that of the probe.

Can You Type Commands into the Serial Terminal?



Note If you are connecting with HyperTerminal: Certain versions of this application exhibit problems communicating with target systems in general. Try to restart the HyperTerminal application. If this is not successful try a different Host machine, upgrade to a newer version of HyperTerminal, or try a different terminal application.

Try using a different serial cable. This must be a NULL modem cable like the one that came with your probe. All lines must be present on your cable. Also check that serial flow control is turned off within your terminal application.

Ethernet Port

Does Your Probe Respond to Ethernet Requests?

The easiest way to determine if your probe ethernet settings are correct is to attempt to **ping** it from your host machine, by typing the following at a command prompt:

`ping` *ProbeIP*|*ProbeName*

A successful **ping** will print a report like the following:

```
$ ping mp11

    Pinging mp11.ghs.com [192.168.100.111] with 32 bytes of data:

    Reply from 192.168.100.111: bytes=32 time<10ms TTL=29
    Reply from 192.168.100.111: bytes=32 time=10ms TTL=29
    Reply from 192.168.100.111: bytes=32 time<10ms TTL=29
    Reply from 192.168.100.111: bytes=32 time<10ms TTL=29

    Ping statistics for 192.168.100.111:
        Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
       Minimum = 0ms, Maximum =  10ms, Average =  2ms
```

An unsuccessful **ping** will print a report like the following:

```
$ ping mp11

    Pinging mp11.ghs.com [192.168.100.111] with 32 bytes of data:

    Request timed out.
    Request timed out.
    Request timed out.
    Request timed out.

    Ping statistics for 192.168.100.111:
        Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum =  0ms, Average =  0ms
```

If your **ping** is successful, continue to the next section.

If your **ping** is unsuccessful, ensure that an ethernet cable is plugged in, and that the green connection light on the ethernet port is lit.

If it is not lit, there is a problem with your ethernet connection. Try a different cable, or a different port on the hub you are connecting to.

If the light is on, you should periodically see the orange light flash indicating that data is being sent over the ethernet. If you are still unsuccessful, you may need to configure your IP address. (see “Setting the IP Address” on page 54)

Can you Ping your Probe, but not get a Response to other Commands?

Try unplugging the ethernet cable from the probe and **ping** it again. If the **ping** is still successful, there are probably two devices with the same IP address. Check with your systems administrator to obtain a unique IP address.

Otherwise, try to open a telnet or Web Server connection from a different Host machine. If problems persist, contact Green Hills Support.

USB Port

Does a Hardware Configuration Wizard Appear When you Plug in your USB Cable?

This indicates that the Probe drivers were not installed correctly. Re-install them by running the installation application **setup.exe** on the root level of the Probe CD. This will place a **ghs_usb.sys** file in the **windows/system32/drivers/** directory and a **ghs_usb.inf** file in the **windows/inf/** directory.

After you have installed the drivers the probe should be automatically recognized. If problems persist, contact Green Hills Support.



Note USB is only supported on Windows 2000/98/ME/XP

Does Your USB Link LED Illuminate?

The USB link LED should illuminate briefly when the host machine and the probe have both been booted.

If this does not happen:

- Check that you have properly installed the USB cable between your host machine and the probe.
- Unplug the USB cable from the probe and plug it back in again.
- Unplug the USB cable from the probe, re-install the drivers, and plug it back in again.

If problems persist, contact Green Hills Support.

My Green Hills Probe is not communicating properly with my target device

In general you should verify the following when you are experiencing problems connecting with your target devices:

- Make sure you have selected the correct CPU Adapter type for your target device in the probe setup facility. (see “Setting the Adapter Type” on page 55).
- Ensure that the ribbon cable is securely connected, with the proper pin orientation, to the CPU Adapter and the target device JTAG/BDM header.
- Ensure that the correct CPU type is selected in the probe setup facility.
- Ensure that the target system is powered.
- Ensure that the target system is known to be working.

Target Connection Troubleshooting

Verify the probe configuration

Connect a terminal to the probe (either through the probe’s RS-232 port or telnet to the probe’s IP address) and type in:

```
set
```

The results of the set command will show most of the probe’s configuration settings, for example:

```
$set
ip                  = 192.168.17.12
netmask             = 255.255.255.0
gateway             = 192.168.17.254
adapter             = auto[arm-20]
clock               = 10000000
logic_high          = 3.300000
multicore           = off
target              = arm7tdmi
checker             = on
debug               = 32
override            = off
power_detect        = on
register_cache      = on
user_string         =
baudrate            = 9600
rst_pulse           = 100 ms.
rst_settle          = 300 ms.
jrst_pulse          = 100 ms.
jrst_settle         = 300 ms.
endianness          = auto
```

The important items to look for are:

- **target** Make sure it's set correctly for the target CPU type
- **clock** JTAG clock frequency. Try reducing this value if problems persist (nominal value = 10000000).
- **logic_high** This should be set to the CPU core voltage, often will be 3.3V
- **multicore** Set this parameter to `off` if debugging a single-CPU core target
- **endianness** Target endianness is determined automatically by the probe, but can be set manually with this value.

If the probe configuration appears correct, try controlling the target.

Can the target's JTAG port be placed in BYPASS Mode?

Use the **vb** (verify bypass) command to test the CPU target JTAG port in BYPASS mode. If the test is successful, the result will be:

```
Test passed.
```

If not successful, the result will look like:

```
Test failed: in=0x1, out=0xffffffff
```

Verify bypass is a low-level diagnostic; if this does not work, but the target is known-good, use the **vta** (verify target adapter) command to test the CPU adapter on the front of the probe

Test the Probe's front-panel CPU adapter

The **vta** (verify target adapter) command to test the Probe's front-panel CPU adapter. This test can be run while connected to a target system, but it's better to also test while disconnected from the target (in case there's a shorted pin, for instance.) The results should be similar to:

```
vta
Testing arm-20 adapter pins:
                      VTREF passed. [100%]
                    VSUPPLY passed. [100%]
                      NTRST passed. [100%]
                        GND passed. [100%]
                        TDI passed. [100%]
                        GND passed. [100%]
                        TMS passed. [100%]
                        GND passed. [100%]
                        TCK passed. [100%]
                        GND passed. [100%]
                       RTCK passed. [100%]
                        GND passed. [100%]
                        TDO passed. [100%]
                        GND passed. [100%]
                      NSRST passed. [100%]
                        GND passed. [100%]
                      DBGRQ passed. [100%]
                        GND passed. [100%]
                     DBGACK passed. [100%]
                        GND passed. [100%]
```

If any pin has a connection problem, it will be displayed as:

```
[any pin] was not initially low.
```

Make certain the target CPU adapter type shown in the results correctly matches the adapter on the probe's front panel. If this test fails AND the adapter type is correct, there is some problem with the CPU adapter on the Probe.

- Try repeating the test with the target cable unplugged from the adapter.
- Make sure the adapter is plugged in securely, and the screws are snug.
- Look for signs of damage or contaminants on the CPU adapter.
- Try removing the adapter, inspecting for damage, then re-installing the adapter and repeating the test.

If these steps still yield negative results, the probe or CPU adapter might be damaged. Contact Green Hills Software for Technical Support.

If this test passes, check the status of the CPU interface pins when connected to the target using the **jp** (JTAG pins) command.

Check the status of the debug interface connector using the jp command

The **jp** (show JTAG pins) command will read and display the current state of the pins on the target connector that are used by the probe for debugging. The important things to watch for are:

- Output Pins: are they enabled, or tri-stated?.
- Adapter type: is it correct/does it match the CPU adapter on the probe?
- *RST (reset) pins: are the RESET pins held in a reset state (usually LOW)?
- VSUPPLY: Is there POWER on the target board?

Does the target system correctly RESET when you type the tr (target reset) command?

The **tr** (target reset) command will momentarily assert a JTAG and system RESET to the target for a duration specified with the **rst_pulse**, **rst_settle**, **jrst_pulse**,and **jrst_settle** configuration settings. You may need to review your target's schematics as a reset from the JTAG/BDM does not always do the same thing as a reset initiated elsewhere.

If the command is successful, you should see something like:

```
Core 0 running
```

If the command is not successful, check the `rst_pulse` and `rst_settle` times using the **set** command. 200 mS `rst_pulse` and 300 mS `rst_settle` times will work on most targets, but might need to be extended for your application, particularly if a supervisory IC (which generates CPU reset) is used -- extend the `rst_settle` time to accommodate the reset and startup delay of the target.

Another option is to manually RESET the target system using its own reset circuitry.

Does the target system correctly HALT when you type the th (target halt) command?

This command will attempt to halt a running CPU target. If successful, will indicate something similar to:

```
Core 0 halted; pc=0x00000100
```

If not successful:

- Check that the JTAG clock frequency is not set too high.
- Check that the target setting matches the target CPU type.
- Check that the CPU has been properly RESET.
- Try power-cycling the target system and repeating the above tests.

How can I monitor the communications between host, probe, and target?

Detailed visibility into the commands and data between the host and probe, and the probe and target is available through the probe's terminal windows by using the **debug** command. Type this command into the terminal window you will be using for debug monitoring (default is RS-232 port.) The amount of information displayed can be set with the **debug n** (set debug mask) command. Refer to the Command Reference for more information.

How can I test that the probe is working correctly with my target system?

The probe offers a series of built-in tests for verifying correct operation. The **vm** (verify memory) and **vc** (verify code) commands are useful for testing correct interaction with the target system, as well as measuring probe-to-target performance. Refer to the Command Reference for more information.

Does your Target Give you Meaningful Data when you Read and Write Registers?

If this is not the case, your target adapter may be damaged. Execute the **vta** command in the terminal window. All pins should be able to be brought high and low. If this command fails contact Green Hills support.

Attempt to reset the board and let it run for while. A reset should synchronize the probe with the target. Running it will allow the Boot ROM to configure the chip, and halting it will allow you to access the internals of the device.

```
tr r #
```

reset and run the target

```
th
```

If you still do not get valid data back from the probe, the electrical interface may not be configured correctly. You can quickly test this by executing the **vb** command. The target device does not need to be in any particular state when executing this command

```
vb
```

Test passed.

If this test fails, then the probe and the target system are not communicating correctly. You can narrow this down by executing low level JTAG commands.

```
mips_4kc[c] % jr
mips_4kc[c] % ji 0x1f
mips_4kc[c] % jd 32 0xffff0000
0x7fff8000
```

For most target systems you should get back what you typed in shifted over by one. In this case `0x7fff8000` or `0xffff8000`.

If you receive back all ones (`0xffffffff`) or all zeros (`0x00000000`) you should verify your connection, CPU adapter and Ribbon Cable. If you receive a random pattern, contact Green Hills Support.

Appendix A

Declaration of Conformity

Declaration of Conformity

according to ISO/IEC Guide 22 and CEN/EN 45014

Manufacturer's Name:	Green Hills Software, Inc.
Manufacturer's Address:	30 West Sola Street Santa Barbara, CA 93101 USA

Declares, that the product:

Product Name:	Green Hills Probe
Model Number(s):	
Product Options(s):	All options based on the above

Is in conformity with:

- EMC:
  - EN 55022 Class B
  - CISPR 22 Class B
  - EN 61000-3-2
  - EN 61000-3-3
  - EN 55024
- ANSI C63.4-1992 FCC Part 15, Subpart B Class B

Safety:

Additional Information:	This product was tested in a typical configuration.
Date:	May 14, 2001

Index

A

adapter type
    setting, 55
address command, 38
alias command, 25, 27
amask command, 38
arrays, 47
assignments, 46

B

base register
    setting, 12
baud rate
    setting, 57
bc command, 25, 27
bca command, 25, 27
bl command, 25, 27
bs command, 25, 28
`.bss` sections, 41 to 42
`.bss` section, reason for excluding, 18
Builder, MULTI, 15

C

chain
    option, 56
close command, 39
commands
    alias, 25, 27
    bc, 25, 27
    bca, 25, 27
    bl, 25, 27
    bs, 25, 28
de, 28
debug, 25, 29
detect, 29
exit, 25, 29
generic debugging, 38
gpin, 26, 29
gpincfg, 26, 30
help, 25, 30
info, 25, 30
jd, 26, 31
ji, 26, 31
jp, 26, 31
jr, 26, 31
ma, 25, 32
md, 25, 32
MPserv
    setup script, 12 to 13
mr, 25, 33
mw, 25, 33
reboot, 25, 33
restore, 24, 33
rr, 25, 34
rw, 25, 34
save, 24, 34
set, 25, 34, 54
setup, 25, 35, 54
t, 25, 35
tc, 25, 35
testing, 13
th, 25, 35
ti, 25, 35
tr, 25, 35
ts, 25, 35

vb, 26, 36
vbp, 26, 36
vc, 26, 36
vlb, 26, 36
vm, 26, 37
vrh, 26, 37
vsi, 26, 37
vta, 26, 38
w, 25, 38
conditionals, 47
configuring
target, 56

D

`.data` sections, 41 to 42
de command, 28
debug command, 25, 29, 39
debug message generation
setting, 58
detect command, 29
DRAM, 12

E

echo command, 39
EDO Mode, 12
example projects, 9
example scripts, 49
exit command, 25, 29
expressions, 45

F

Fast Page Mode, 12
firmware
updating, 72
-force_coreid *core*, 18
fprint command, 39
fprintb command, 39
fread command, 39
freadb command, 39

G

gateway
setting, 55
getenv command, 39
gpin command, 26, 29
gpincfg command, 26, 30
Green Hills Probe
advanced configuration options, 54
capabilities of, 2
command summary, 24
components of, 4
configuring, 6
configuring your target, 56
connecting to your host, 5
connecting to your target, 6
installing software for, 4
setting adapter type, 55
setting baud rate, 57
setting debug message generation, 58
setting gateway, 55
setting IP address, 54
setting JTAG clock speed, 58
setting JTAG TAP reset pulse length, 59
setting JTAG TAP reset settle length, 60
setting netmask, 54
setting reset pulse length, 59
setting reset settle length, 60
setting sanity checking, 58
setting status checking, 57
setting target power detection, 59
setting user string, 59
testing connections to, 8
troubleshooting, 75
updating firmware, 72
web interface, 68

H

halt command, 39
help command, 25, 30, 40

Index

I

info command, 25, 30
interrupt sources, 12
    disabling, 12
IP address
    setting, 54

J

jd command, 26, 31
ji command, 26, 31
jp command, 26, 31
jr command, 26, 31
JTAG clock speed
    setting, 58
JTAG TAP reset pulse length
    setting, 59
JTAG TAP reset settle length
    setting, 60

L

linker directives files
    editing, 14
listvars command, 40
load command, 41
-log *filename*, 18
loops, 48

M

m command, 41, 49
ma command, 25, 32
md command, 25, 32
memory
    configuring, 11
    testing, 15
MPadmin
    invoking, 72
    options for, 72
MPserv
    commands, 12 to 13
    startup options, 18
mr command, 25, 33
multi-core
    option, 56
multi-core operation
    configuring target, 56
mw command, 25, 33

N

netmask
    setting, 54
-bss, 18
-nodata, 18
nofail command, 41
-noload, 19
noload command, 42
-notext, 18
numbers
    methods for specifying, 22

O

open command, 42

P

pid command, 42
print command, 42
processors
    supported, 9

R

random command, 42
reads
    testing, 14
reboot command, 25, 33
registers
    reading, 14
    setting values, 12 to 13
    writing, 14
regnum command, 42
reset pulse length
    setting, 59

reset settle length
    setting, 60
restore command, 24, 33
rr command, 25, 34
rst command, 11
run command, 43
rw command, 25, 34

S

sanity checking
    setting, 58
save command, 24, 34
script command, 13, 43
scripting, 45
    arrays, 47
    assignments, 46
    conditionals, 47
    examples, 49
    expressions, 45
    loops, 48
    notes, 45
    syntax, 45
    variable expansion, 48
-set *option*=**c**, 19
set command, 25, 34, 54
setup command, 25, 35, 43, 54
-setup *file*, 19
setup files, 9
    editing, 11, 14
    example, 13
    testing, 14
-setup option, 14, 45
single-core operation
    configuring target, 56
sleep command, 43
SRAM, 12
status checking
    setting, 57
status command, 44
step command, 44

T

t command, 25, 35
target
    option, 56
target power detection
    setting, 59
target system
    configuring, 9
    connecting to, 10, 15, 18
    manually configuring, 11
    memory
        configuring, 11
    preparing for debugging, 20
tc command, 25, 35
testing
    registers, 14
    target memory, 15
`.text` sections, 41 to 42
th command, 25, 35
ti command, 25, 35
tr command, 25, 35
troubleshooting, 75
ts command, 25, 35

U

undef command, 44, 47
-usb [*index*], 19
USER button, 6, 31, 72
user string
    setting, 59

V

variable expansion, 48
vb command, 26, 36
vbp command, 26, 36
vc command, 26, 36
vlb command, 26, 36
vm command, 26, 37
vrh command, 26, 37
vsi command, 26, 37

vta command, 26, 38

W

w command, 25, 38
web interface, 68
    accessing, 68
    configuring probe with, 69
    help menu, 70
writes
    testing, 14

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP