Green Hills Debug Probes User's Guide

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

DISCLAIMER

PubID: ghprobe-525955
Branch: http://toolsvc/branches/probe-branch-5.2
Date: February 19, 2015

Contents

Preface

Contents

This preface discusses the purpose of the manual, the MULTI documentation set, and typographical conventions used.

About This Book

This book contains reference information for the Green Hills Probe and SuperTrace Probe. For installation information, see the *Getting Started* book for your probe. For instructions that explain how to use the MULTI Debugger, see the *MULTI: Debugging* book.

(SuperTrace Probe users only) For specific information about using the TimeMachine debugging tools, which are licensed separately from the MULTI Debugger, see the documentation about analyzing trace data with the TimeMachine tool suite in the *MULTI: Debugging* book.

Note

New or updated information may have become available while this book was in production. For additional material that was not available at press time, or for revisions that may have become necessary since this book was printed, please check your installation directory for release notes, **README** files, and other supplementary documentation.

Conventions Used in the MULTI Document Set

All Green Hills documentation assumes that you have a working knowledge of your host operating system and its conventions, including its command line and graphical user interface (GUI) modes.

Green Hills documentation uses a variety of notational conventions to present information and describe procedures. These conventions are described below.

Convention	Indication	Example
bold type	Filename or pathname	**C:\MyProjects**
	Command	**setup** command
	Option	**-G** option
	Window title	The **Breakpoints** window
	Menu name or menu choice	The **File** menu
	Field name	**Working Directory:**
	Button name	The **Browse** button
italic type	Replaceable text	**-o *filename***
	A new term	A task may be called a *process* or a *thread*
	A book title	*MULTI: Debugging*
`monospace` type	Text you should enter as presented	Type `help` *`command_name`*
	A word or words used in a command or example	The **wait** [-global] command blocks command processing, where `-global` blocks command processing for all MULTI processes.
	Source code	`int a = 3;`

	Input/output	`> print Test`
		`Test`

	A function	`GHS_System()`
ellipsis (...) (in command line instructions)	The preceding argument or option can be repeated zero or more times.	**debugbutton** [*name*]...

greater than sign (>)	Represents a prompt. Your actual prompt may be a different symbol or string. The > prompt helps to distinguish input from output in examples of screen displays.	`> print Test`
		`Test`

pipe (|) (in command line instructions)	One (and only one) of the parameters or options separated by the pipe or pipes should be specified.	**call** *proc* | *expr*

Convention	**Indication**	**Example**
square brackets ([ ]) (in command line instructions)	Optional argument, command, option, and so on. You can either include or omit the enclosed elements. The square brackets should not appear in your actual command.	`.macro` *`name`* `[`*`list`*`]`

The following command description demonstrates the use of some of these typographical conventions.

gxyz [*-option*]... *filename*

The formatting of this command indicates that:

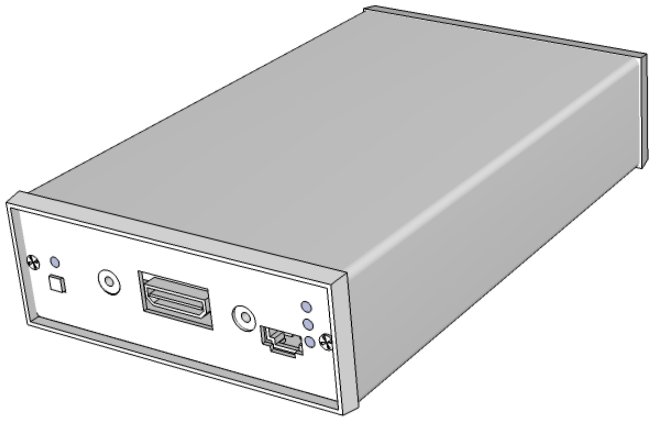
- The command **gxyz** should be entered as shown.
- The option *`-option`* should either be replaced with one or more appropriate options or be omitted.
- The word *`filename`* should be replaced with the actual filename of an appropriate file.

The square brackets and the ellipsis should not appear in the actual command you enter.

Green Hills Probe

The Green Hills Probe is a high-speed, network-enabled, stand-alone debugging instrument that:

- Allows downloading at speeds exceeding 5 MBps.
- Gives you extensive visibility and control of your target through Ethernet or USB host interface ports. (Support via USB is available only on Windows XP or newer.)
- Can be re-targeted to support different CPU families by changing the target adapter and updating your firmware.
- Allows debugging of large multiple-core systems.
- Includes a telnet server and USB connectivity to allow convenient configuration and control of your target.



SuperTrace Probe

The SuperTrace Probe is a network-enabled hardware debugging instrument that provides unprecedented speed and the largest trace memory available in a hardware debugging interface. The SuperTrace Probe shares the basic capabilities of the Green Hills Probe, but can also collect trace data. The SuperTrace Probe:

- Can collect up to four gigabytes of trace data.
- Can collect trace data at speeds up to 300 MHz.
- Can be used with the Green Hills TimeMachine tools to vastly expand the trace and debugging features of the MULTI Debugger.

PARALLEL
TRIGGER
OUT
IN
POWER
HOST
DEBUG
CPU

Chapter 1

Administering Your Probe

Contents

Status LEDs And The User Button

The front of your probe has three status LEDs that indicate various statuses for the probe and your target. This section describes what these LEDs indicate, and how to use the User button on the front your probe. It also covers the status LEDs on included trace pods and active pods.

Green Hills Probe v2 and SuperTrace Probe v1 Status LEDs

The Green Hills Probe v2 and SuperTrace Probe v1 have three single-color status LEDs on the right-hand side. From top to bottom, they are:

Label	Color	Indication When Lit
RST	Red	The target is halted after a reset.
HALT	Amber	The target is in debug mode because it has been halted or is stopped at a breakpoint.
RUN	Green	The target is running.

All three LEDs blink while the probe is tri-stated (for more information, see “The User Button” on page 5). Multiple LEDs light up during system operations (such as firmware updates).

These probes also have a power LED that is solid green when the probe is powered.

Green Hills Probe v3 and SuperTrace Probe v3 Status LEDs

The Green Hills Probe v3 has three LEDs on the right-hand side. From top to bottom, they are:

Label	State	Indication
Host	Solid Green	A host is connected to the probe via Ethernet.
	Solid Amber	A host is connected to the probe via USB.
	Blinking Green	The probe is communicating over Ethernet.
	Blinking Amber	The probe is communicating over USB.
Debug	Blinking Green	The probe is communicating with the target. The speed of the blinks roughly correlates with the density of communication.

Label	State	Indication
CPU	Solid Green	The target is running.
	Solid Amber	The target is in debug mode (for any reason).
Power	Solid Green (STP)	The probe is on.
	Solid Red (STP)	The power supplied to the probe is inadequate.
	Blinking Green (STP)	The probe is booting.
	Solid Green (GHP)	The probe is on.
	Other (GHP)	Contact Green Hills support.
Parallel (STP)	Solid Green	The probe is ready and properly connected to the trace pod over the parallel interface.
	Solid Orange	The probe is partially configured but not ready. May also mean that no adapter is connected.
	Solid Red	The probe is ready, but is unable to communicate to the trace pod. It may not be properly connected.
Serial (STP)	Solid Green	The probe is ready and properly connected to the target over the serial (HSST) interface.
	Solid Orange	The probe is partially configured but not ready.
	Solid Red	The probe is ready, but is unable to communicate to the target. It may not be properly connected.

The **Debug** and **CPU** LEDs both blink while the probe is tri-stated. While updating the probe's firmware, all three LEDs blink and cycle through different colors.

TraceEverywhere Target Adapter LEDs

TraceEverywhere (TE) kits have an adapter attached to the target end of the TE trace pod. This adapter has a single status LED. If the LED is green, the adapter is properly connected to the probe. If it is red or off, there is a problem with the connection.

Note

The LEDs on the TE trace pod are described in the following section.

TraceEverywhere Pod LEDs

TraceEverywhere (TE) kits for the SuperTrace Probe contain a TE pod that has five LEDs. The LED on the top of the pod nearest the ribbon cable indicates whether or not the pod has power. The other four LEDs on the top and bottom of the pod indicate one of the following statuses:

Color	Indication
Orange	No power is detected or the outputs are disabled (the probe is tri-stated).
Green	No trace clock is present. If the light is flickering, there is JTAG activity.
White	The trace clock is present. If the light is flickering, there is JTAG activity.

ColdFire Active Pod LEDs

The ColdFire active pod has the following status LEDs:

Label	Indication When Lit
V	The target is powered. This LED corresponds to the `TVcc` signal from the target.
H	There is activity going in or out the BDM port.
T	The Green Hills Probe is powered.

Trace Pod Status LEDs

Trace pods for SuperTrace probes have several status LEDs that indicate the state of the probe's connection to your target. These LEDs are located on the backside of the trace pod, opposite the heat sink. While different LEDs are different colors, each LED has only one color. The following table describes what each LED indicates when lit:

Order	Label	Indication When Lit
1	**PROBE POWER**	The trace pod is powered.
2	**READY**	The SuperTrace Probe can communicate with the trace pod.
3	**TRACING**	The SuperTrace Probe is collecting trace data.

Order	Label	Indication When Lit
4	**TRIGGER**	The SuperTrace Probe has encountered a trigger event and will stop collecting trace data when the specified percentage of the trace buffer has filled.
5	**RUN CTL**	There is JTAG traffic. This LED blinks to indicate traffic, but the traffic is often so fast that the LED looks solid.
6	**STATUS 1**	The trace clock is present, or, if you are using a phase locked loop (PLL), the PLL is locked.
7	**STATUS 2**	This LED is reserved for future use and is never lit.
8	**STATUS 3**	The trace clock is present. If you are not using a PLL, this light indicates the same thing as `STATUS 1`.

Note

If your trace pod's LEDs are not labeled, the previous table lists them from top to bottom, where the top LED is closest to the Green Hills Software logo.

The User Button

The **User** button is located on the front left side of the probe. You can use the **User** button to:

- Toggle the target outputs on (normal operation) and off (tri-stated). This is equivalent to issuing the **jp on** and **jp off** commands. When outputs are tri-stated, the status LEDs blink, and you can connect and disconnect the target adapter while the probe is still powered and/or the target is running. Never connect a powered probe to a running target unless the outputs are off.
- Perform a serial boot. This is helpful if you need to update firmware on a probe that cannot boot or has a bad firmware image. For more information, see "Updating Probe Firmware to a Working Probe" on page 17.

Ports Used by the Probe

Green Hills Debug Probes use the following TCP/IP ports:

- 23 — Telnet

- 5000 — Communication with MULTI, probe administration utilities, and **mpserv**.

The Graphical Probe Administrator

The Green Hills Graphical Probe Administrator allows you to easily configure your Green Hills Probes. You can also update the firmware for your probes, in addition to viewing probe status and opening consoles to the probes.

Starting the Graphical Probe Administrator

To start the Graphical Probe Administrator, either:

- From the MULTI Launcher, select **Utilities** → **Probe Administrator**; or
- Run **gpadmin** from a command prompt.

Command Line Options

When you run the Graphical Probe Administrator from the command line, you can specify command line options to allow you to open a **Probe Administration** window on a single probe. Additionally, you can specify that you want to update the firmware for a single probe.

Opening a Probe Administration Window

To open a **Probe Administration** window for a single probe connected over Ethernet, enter:

gpadmin *probe_ip*

where *probe_ip* is either the hostname or IP address of your probe.

To open a **Probe Administration** window for a single Probe connected via USB, enter:

gpadmin -usb

For more information about the **Probe Administration** window, see “Using the Probe Administration Window” on page 12.

Updating Probe Firmware

To update the probe firmware for a probe connected via Ethernet from the command line, use the following syntax:

gpadmin -update *probe_ip* [*firmware_file*]

where *probe_ip* is used to specify the probe's IP address or hostname. *firmware_file* is an optional parameter to specify the firmware file to flash to your probe.

To update the probe firmware for a probe connected via USB from the command line, use the following syntax: **gpadmin** -update -usb [*firmware_file*]

where *firmware_file* optionally specifies the firmware file to flash to your probe.

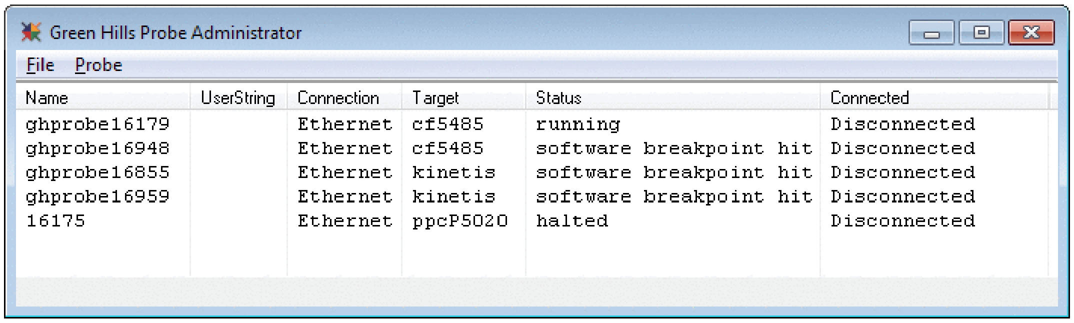
If you do not specify a firmware file, the **Update Probe Firmware** dialog box appears to allow you to select the firmware file to flash to your probe.

If you do specify a firmware file, the file is automatically flashed to your probe.

For more information about updating probe firmware, see “Updating Probe Firmware to a Working Probe” on page 17.

Using the Probe List Window

The primary interface of the Graphical Probe Administrator is the **Probe List** window, shown next.



Name	UserString	Connection	Target	Status	Connected
ghprobe16179		Ethernet	cf5485	running	Disconnected
ghprobe16948		Ethernet	cf5485	software breakpoint hit	Disconnected
ghprobe16855		Ethernet	kinetis	software breakpoint hit	Disconnected
ghprobe16959		Ethernet	kinetis	software breakpoint hit	Disconnected
16175		Ethernet	ppcP5020	halted	Disconnected

Managing the Probe List

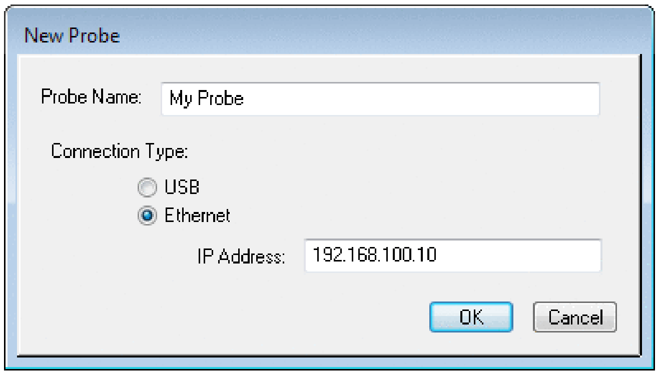
The **Probe List** window displays information about probes you have added.

Adding Probes to the Probe List

To manually add probes to the **Probe List** window:

- Select **File** → **New Probe**; or
- Right-click in the **Probe List** window, and select **New Probe**.

These commands will open the **New Probe** dialog box that allows you to enter information about the new probe so that the Graphical Probe Administrator can connect to it. The **New Probe** window is shown in the following.



Probe Name: My Probe

Connection Type:

USB [✓ Ethernet]

IP Address: 192.168.100.10

1. To add a new probe, enter a name to identify the probe. This name is only used to help you identify the probe while using the Graphical Probe Administrator.
2. Select the method that the Graphical Probe Administrator should use to connect to the probe.
    - If you have a probe connected via USB, select the **USB** radio button.
    - If you have a probe connected over Ethernet, select **Ethernet**. Enter the IP address for this probe, which can be entered as either a numeric IP address or using the hostname assigned to the probe.

Modifying Probes in the Probe List

Once a probe appears in the **Probe List**, you can edit its name and connection method by either:

- Double-click the probe, and select **File → Edit Probe**; or
- Right-click a probe, and select **Edit Probe**.

This opens the **Edit Probe** dialog box, which allows you to modify the name and connection method for the probe.

Removing Probes from the Probe List

You can remove a probe from the Probe List by either:

- Double-click the probe, and select **File → Remove Probe**; or
- Right-click the probe and select **Remove Probe**.

Removing a probe has no effect other than displacing it from the **Probe List**. Once a probe has been removed, you can add it back later if you need access to it.

Probe List Data

For each probe in the **Probe List** window, the Graphical Probe Administrator opens a connection to the probe and retrieves some information. It then collects some information from the probe and disconnects from it.

For each probe in the **Probe List**, the following information is displayed:

Column	**Description**
Name	A user-defined name to uniquely identify this probe.
User String	The user string for this probe. For more information about the user string, see "Setting the User String" on page 85.
Connection	The connection type that the Graphical Probe Administrator uses to connect to this probe (USB or Ethernet). If the probe is connected by Ethernet, this column displays the Ethernet address in brackets.
Target	The target type that is configured for this probe.

Column	Description
Status	The status of the target connected to this probe. This status could be running, halted, stopped at a breakpoint, or one of several other states.
Connected	The state of the Graphical Probe Administrator's connection to this target. This column is useful so that you can see whether it is currently connected to a probe, trying to connect to a probe, or if it is unable to connect to a probe.

Refreshing Probe Data

After this information is collected from the target, the Graphical Probe Administrator disconnects from each probe. The information might become stale after a period of time. You can force the **Probe List** to refresh a single probe by right-clicking it, and selecting **Refresh** from the context menu. The Graphical Probe Administrator connects to the probe and updates the information in the **Probe List**.

Additionally, you can refresh the information for all probes in the **Probe List** by selecting **File → Refresh All**. The Graphical Probe Administrator then connects to each of the probes and updates the information in the **Probe List**.

Maintaining Probe Connections

You can also use the **Probe List** to monitor the status of the processors that are connected to each probe. You can tell the **Probe List** to stay connected to a probe. When the Graphical Probe Administrator is connected to a probe, the **Status** column updates as the target status changes. To tell the Graphical Probe Administrator to maintain a connection to a probe:

- Double-click the probe, and select **Probe → Stay Connected**; or
- Right-click the probe, and select **Stay Connected**.

When you select **Stay Connected**, the Graphical Probe Administrator connects to the probe and maintains a connection so that the target status stays up to date. In addition, a check mark appears next to the **Stay Connected** menu item.

To disconnect from a probe that has **Stay Connected** enabled, select **Stay Connected** again. The Graphical Probe Administrator disconnects from the probe (unless a

Probe Administration window is still open) and the **Stay Connected** menu item is disabled.

Probe List Colors

The probes that appear in the **Probe List** are color coded so that you can quickly tell the status of the connection to each probe. The various colors are:

- Black — Indicates that the Graphical Probe Administrator was able to connect and retrieve the status, but has since disconnected from this probe.
- Green — Indicates that the Graphical Probe Administrator is actively connected to this probe.
- Blue — Indicates that the Graphical Probe Administrator is actively attempting to establish a connection to this probe.
- Red — Indicates that the Graphical Probe Administrator was unable to connect to this probe.

Probe List Operations

From the **Probe List** window, you can configure the settings for a probe, upgrade the firmware for a probe, and reboot a probe.

Opening a Probe Administration Window

To configure the settings for a probe or to access a terminal to it, do one of the following:

- Highlight a probe, and then select **Probe → Configure Probe;**
- Right-click a probe, and select **Configure Probe**; or
- Double-click a probe in the **Probe List**.

This opens a **Probe Administration** window. For more information about the **Probe Administration** window, see "Using the Probe Administration Window" on page 12.

Updating Probe Firmware

To update the firmware for a probe, either:

- Highlight a probe, and select **Probe → Update Firmware**; or
- Right-click a probe, and select **Update Firmware**.

For more information about updating the firmware for your probe, see “Updating Firmware with the Probe Administrator” on page 17.

Rebooting a Probe

You may want to reboot a probe so that certain settings that require a reboot can take effect. To reboot a probe, either:

- Highlight the probe in the **Probe List** window, and select **Probe → Reboot Probe**; or
- Right-click a probe in the **Probe List**, and select **Reboot Probe**.

This sends a reboot command to the probe and it reboots. The **Probe List** indicates that the selected probe is rebooting, waits for a short time, and then attempts to reconnect to the probe. After reconnecting, it reads information from the probe and displays it in the **Probe List**.

Any **Probe Administration** windows that are open for the selected probe close when a reboot is requested.

Note

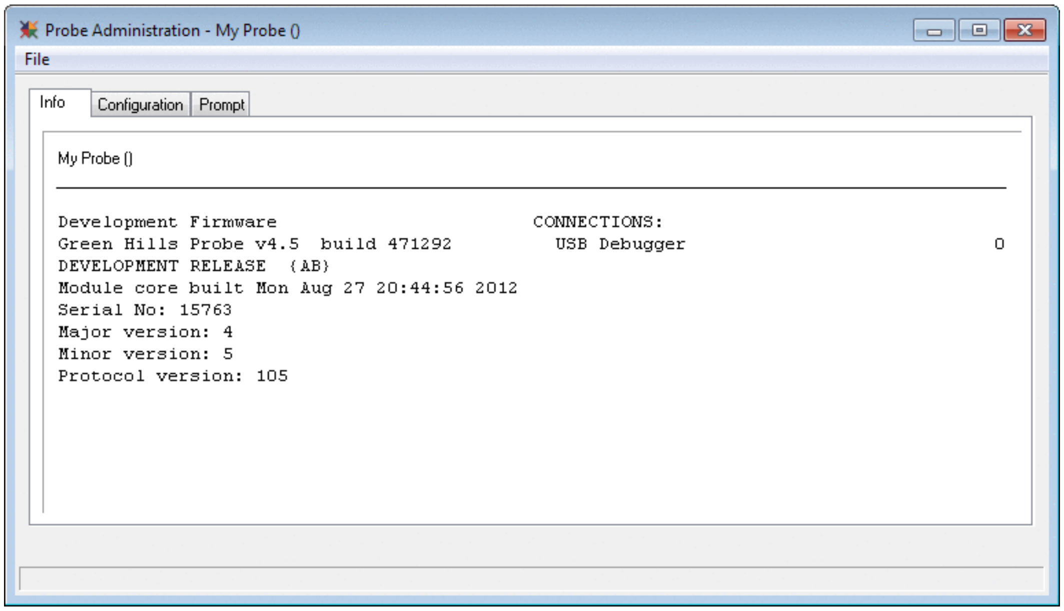
The **reboot** command is not supported in the **Prompt** pane of the **Probe Administration** window.

Using the Probe Administration Window

The **Probe Administration** window allows you to configure the options for your probe, and offers a command prompt that can be used to interact with the probe.

Viewing Probe Information with the Info Pane

The **Info** pane displays some general information about your probe, including the firmware version, its serial number, and a list of the current connections to the probe. The **Info** pane is displayed below:



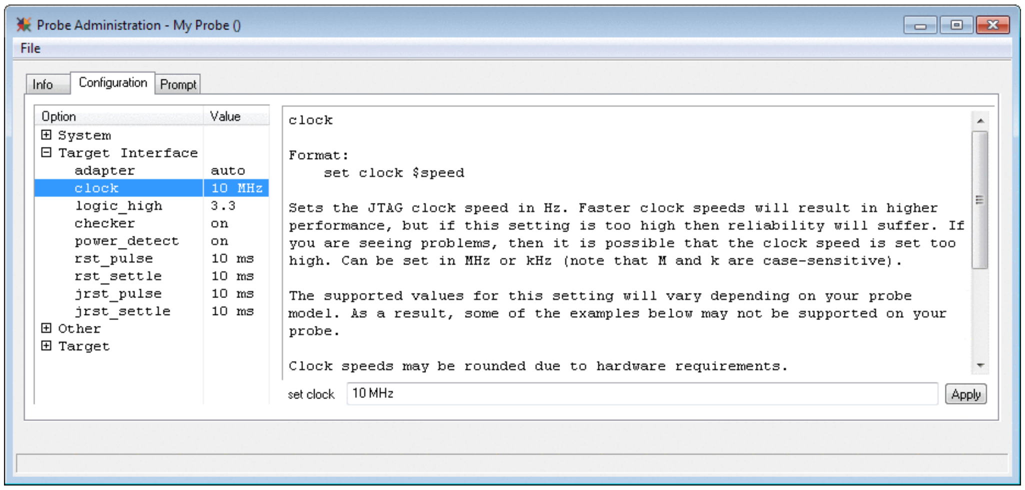
```
Development Firmware                        CONNECTIONS:
Green Hills Probe v4.5  build 471292          USB Debugger                         0
DEVELOPMENT RELEASE  (AB)
Module core built Mon Aug 27 20:44:56 2012
Serial No: 15763
Major version: 4
Minor version: 5
Protocol version: 105
```

The left side of the **Info** pane displays the firmware version, the probe serial number and other information about the firmware.

The right side of the **Info** pane displays a list of the connections and the number of seconds since there was communication over each connection.

Setting Probe Options with the Configuration Pane

The **Configuration** pane allows you to view and modify probe options. The **Configuration** pane is shown below.



```
Option               Value
⊞ System
⊟ Target Interface
    adapter          auto
    clock            10 MHz
    logic_high       3.3
    checker          on
    power_detect     on
    rst_pulse        10 ms
    rst_settle       10 ms
    jrst_pulse       10 ms
    jrst_settle      10 ms
⊞ Other
⊞ Target
```

```
clock

Format:
    set clock $speed

Sets the JTAG clock speed in Hz. Faster clock speeds will result in higher
performance, but if this setting is too high then reliability will suffer. If
you are seeing problems, then it is possible that the clock speed is set too
high. Can be set in MHz or kHz (note that M and k are case-sensitive).

The supported values for this setting will vary depending on your probe
model. As a result, some of the examples below may not be supported on your
probe.

Clock speeds may be rounded due to hardware requirements.
```

set clock 10 MHz

The left side of this pane contains a list of all of the current probe options as well as their current values. The options that appear in this list depend on what type of probe you are connected to, what version of probe firmware you are using, and what target you have configured.

When you select an option, documentation for this option is displayed on the right side of this window. Additionally, there are controls that allow you to change the value of the selected option.

There are several controls that may appear to change the current value of an option depending on the type of option. The possible controls are:

- A text field, for options that require a string to be entered. To change the value of these options, type a new value and click **Enter**.
- A pull down control, for options that have a fixed set of values. To alter the value of these options, change the selection in the pull down control. Once a new value is selected, it is automatically changed on the probe.

For more information about probe options, see Chapter 4, “Probe Option Reference” on page 65.

Saving and Restoring Probe Options

You can save and restore the options configured in the **Configuration** pane. This allows you to save the probe configuration for a specific target and restore it later. This is very useful if you have two or three different targets that you connect to regularly. It is also convenient to copy the settings from one probe to another, and enables sharing probe settings across an entire development group.

To save the current set of options, select **File → Save Options to File**, and then select a filename. The options are saved to this file.

To load the options from a previously saved file, select **File → Load Options from File**. The options saved in the file are loaded to the probe, and the **Configuration** pane is updated to reflect the new option values.

Note

Configuration files ending in **.ghpcfg** are not supported in the Graphical Probe Administrator. To load them, see "Loading and Saving Configuration Files" on page 20.

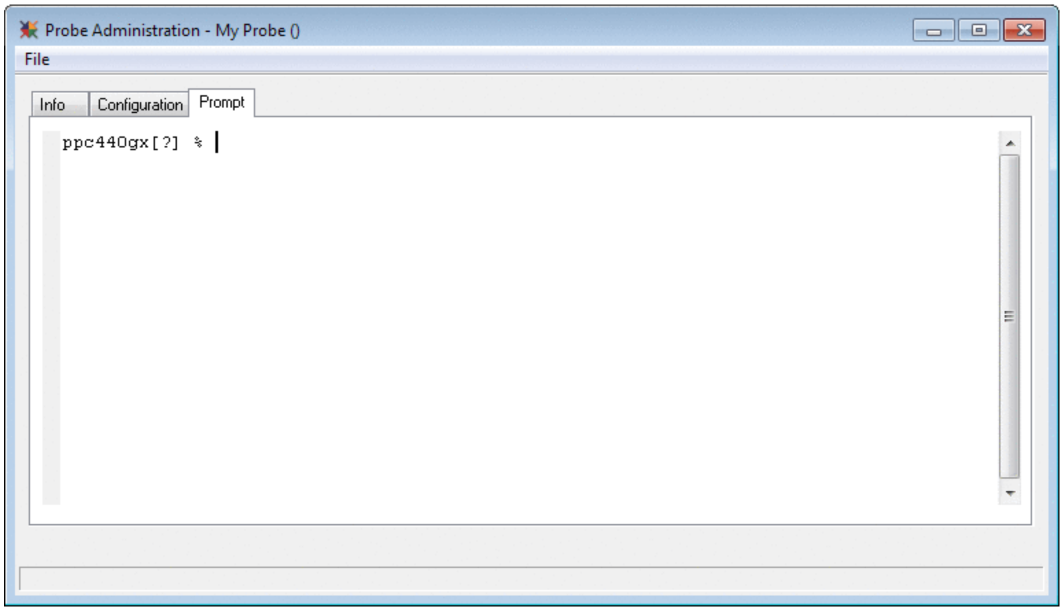
Refreshing the Configuration Pane

If probe options are changed through an interface other than the Graphical Probe Administrator (that is, through a telnet console or a connection through MULTI), you can refresh the values in the **Configuration** pane.

To refresh the **Configuration** pane, select **File → Refresh**. All configuration options are then read from the probe and the display is updated.

The Prompt Pane

The **Probe Administration** pane offers a command pane so that you can interact directly with the probe. The **Prompt** pane is shown below.



```
ppc440gx[?] %
```

From this pane, you can issue nearly all of the probe commands, including run-control commands, register commands, memory read and write commands, and many others. For more information about the commands you can use to interact with this prompt, see Chapter 5, “Probe Command Reference” on page 173.

Updating Firmware with the Probe Administrator

You can update the firmware for your probe using the Graphical Probe Administrator.

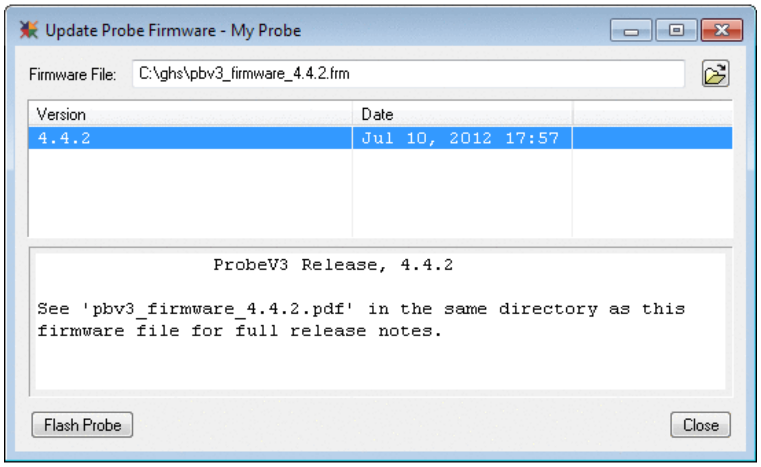
If your probe is functioning normally, then you can update the firmware over USB or Ethernet using the steps outlined in “Updating Probe Firmware to a Working Probe” on page 17.

However, if your probe firmware has been corrupted, you must recover your probe by downloading new firmware over the serial port. For more information about how to do this, see “Recovering a Probe that Has Corrupt Firmware” on page 18.

Updating Probe Firmware to a Working Probe

If your probe is communicating successfully over Ethernet or USB, you can update the firmware by performing the following:

1. Select the probe you want to update in the **Probe List** window. After selecting the probe, either select **Probe** → **Update Firmware** or right-click it and select **Update Firmware**. This opens the **Update Probe Firmware** dialog box:



Firmware File: C:\ghs\pbv3_firmware_4.4.2.frm

Version	Date
4.4.2	Jul 10, 2012 17:57

```
                 ProbeV3 Release, 4.4.2

See 'pbv3_firmware_4.4.2.pdf' in the same directory as this
firmware file for full release notes.
```

2. Select the firmware file you want to load onto your probe. You can either type the full path to the firmware file in the **Firmware File** text field or click the button. Clicking the button opens a file chooser so that you can select the firmware file.

3. After you select a firmware file, the release notes are displayed so that you can read about the new firmware, and verify that you want to flash this firmware image.
4. Click **Flash Probe** and the firmware flash process begins. The **Reloading Probe Firmware** dialog box displays the download and flash status.
5. The probe is rebooted and the new firmware is now loaded.
6. Click **Done** to dismiss the progress dialog.

Recovering a Probe that Has Corrupt Firmware

If your probe has invalid firmware loaded to it and is unable to communicate over Ethernet or USB, you can use the following steps to load a new firmware image over the serial port.

Warning

If you have special feature keys installed by Green Hills Support, they will be removed during recovery and must be re-installed.

1. Turn your probe off, and connect it to your host machine through the serial port using a null modem cable.
2. On your host machine, run MULTI. In the MULTI Launcher, select **Utilities → Probe Administrator**.
3. In the **Probe Administrator** window, select **Probe → Recover Probe**. A dialog box opens to confirm that you want to update your firmware. Click **OK**.
4. The **Update Probe Firmware** window opens. Enter the full path to a firmware file in the **Firmware File** box, or click the button and select it in the file chooser. Click the **Flash Probe** button.
5. The **Serial Boot** dialog box opens. If the correct serial port is not already displayed, type the name of the serial port to which your probe is connected. Click **OK**.
6. A dialog box opens. Turn your probe on while holding down the **User** button (as instructed by the dialog). After turning on your probe, click the **OK** button.
7. The **Reloading Probe Firmware** dialog box opens and displays the status of the update. When it is finished, click **Done**.
8. Power cycle your probe. The new firmware is now loaded and running.

The Command Line mpadmin Utility

mpadmin is a command line utility program with many administrative functions. You can use it to set or check your configuration settings, to update your firmware, to load a configuration file, and to test your connection.

mpadmin Syntax

To invoke **mpadmin**, enter:

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

where:

- ***connection*** is required and specifies the type of connection and, if applicable, the IP address of the target, as listed in the following.
  - For Green Hills Probe Ethernet connections, *connection* is the hostname or IP address of the target.
  - For Green Hills Probe USB connections, *connection* is **-usb [*serial_number*]**, where *serial_number* is optional and can be used to specify a particular probe if multiple probes are connected to a single host. If multiple probes are connected over USB, we recommend using a serial number; if you do not, the probe is selected in a non-deterministic way.
  - For Green Hills serial connections, *connection* is **-serial *port***, where *port* specifies the serial port connected to the probe.
- ***firmware_file*** is required and specifies the firmware image file supplied by Green Hills Software.

  The *firmware_file* argument must follow the *connection* argument.
- *options* can be any non-conflicting combination of the options listed in the following section.

mpadmin Options

mpadmin has the following options:

Option	Effect
-cfgload	Loads configuration information from a configuration file into the probe. Configuration files may set probe feature keys, may set or clear probe xswitches, and may set probe options. Loading a configuration file may require the probe to reboot one or more times.
-cfgsave	Saves configuration information from the probe into a configuration file.
-full_format	Fully format the probe's flash on update.
-list	Lists contents of firmware file (no connection).
-reset	Restores all probe settings to factory defaults
-setup	Prompts the user for common settings to configure the probe, such as IP address and target type.
-update	Updates the probe firmware.
-v	Runs the update in verbose mode.

Running mpadmin's Interactive Setup Utility

mpadmin includes a setup utility that prompts you for common probe settings and reboots the probe. To run the utility and configure a probe, type the following at the command line:

```
> mpadmin -setup connection
```

For example, to configure a Green Hills Probe over USB, use:

```
> mpadmin -setup -usb
```

Loading and Saving Configuration Files

You can use **mpadmin** to load and save configuration files that store your probe's settings. To save a configuration file ***mysetup*.ghpcfg**, type the following at the command line:

```
> mpadmin -cfgsave connection mysetup.ghpcfg
```

To load the configuration file ***mysetup*.ghpcfg**, type the following at the command line:

```
> mpadmin -cfgload connection mysetup.ghpcfg
```

For example, to save the settings from a probe on the network with the hostname `probe_1` into **config.ghpcfg** and load them into a probe connected to your host over USB, type the following commands:

```
> mpadmin -cfgsave probe_1 config.ghpcfg
> mpadmin -cfgload -usb config.ghpcfg
```

-cfgsave and **-cfgload** are not supported over serial port connections.

Loading a **.ghpcfg** file may reboot the probe, depending on the contents of the file.

Note

You can load legacy **.cfg** files using **mpadmin** in the same manner as **.ghpcfg** files. Loading a **.cfg** file never reboots the probe, so you may need to do so manually to make sure all settings take effect.

Updating Firmware with mpadmin

When new versions are available, you can update your Green Hills Probe firmware. Firmware updates usually include new features, add new CPU support, and/or enhance the performance of the probe. To update the probe's firmware through the serial, USB, or Ethernet port, type the following at the command line:

```
> mpadmin -update connection firmware.frm
```

To update the probe over an Ethernet or USB connection, a valid firmware image must already be present and running on the probe, and the probe must be able to boot correctly. Otherwise, you must update the firmware through the serial port.

For example, if your probe is booting properly and you want to update the firmware to the image **new_firmware.frm** over USB, type the following at the command line:

```
> mpadmin -update -usb new_firmware.frm
```

If the probe does not boot correctly and you need to update the firmware over the serial port `com1`:

- Type the following command:

  ```
  > mpadmin -update -serial com1 new_firmware.frm
  ```

- Follow the instructions displayed by **mpadmin** by turning the probe off and on while holding down the **User** button. This causes the probe to boot from code received on the serial port, which allows it to be reprogrammed even if there is not a valid image present on the probe.
- After the programming is completed, reboot the probe.
- Verify that the update did not change or erase probe settings using **mpadmin** or a probe console. Make sure to check your settings before attempting to debug any programs with your probe.

Chapter 2

Configuring Target Resources

Contents

Every Top Project contains a **Target Resources** project (**tgt/resources.gpj**) that holds the files that help MULTI connect to and configure your target board, including:

- a *board setup script* — MULTI uses this file to initialize your target before downloading and debugging a program. The default board setup script is **mpserv_standard.mbs**.
- *linker directives files* — The linker uses one of these files to determine how how to link your executable and load it into memory. By default, your program is linked to and executes out of RAM, using the information in **standalone_ram.ld**.

If your target board is not listed in the **Project Wizard** or you experience problems when downloading and debugging programs, you may need to customize these files. This section provides customization guidelines and diagnostics that you can use to make sure that your target resources are configured correctly.

Customizing MULTI Board Setup Scripts

A board setup script is a file that MULTI uses to initialize your target before downloading and debugging a program. The default board setup script is **mpserv_standard.mbs**, located in your target resources project.

Setup scripts in MULTI use the following commands and conventions:

- The MULTI scripting language described in the documentation about MULTI scripts in the *MULTI: Scripting* book.
- Commands described in the *MULTI: Debugging Command Reference* book. You can find an overview of useful board setup script commands in “Useful Commands for MULTI Board Setup Scripts” on page 29.
- Additional probe commands listed in Chapter 5, “Probe Command Reference” on page 173. When specifying any of these additional commands, you must prepend the **target** command.

Note

If you want to use a legacy script generated by MULTI 4 or older, see “Using Legacy (.dbs) Setup Scripts When Connecting to Your Target” on page 56.

To edit a MULTI board setup script to make it suitable for your system:

1. Obtain your board and processor's documentation. You will need it to gather information about memory resources, registers, interrupts, etc.
2. Double-click **mpserv_standard.mbs** in your target resources project to open it in an editor.
3. Determine whether your board can initialize itself, and then proceed as follows:
    - If your target does not have a valid ROM image that initializes the target upon reset, skip to the next step.
    - If your target has a valid ROM image that initializes the target upon reset, comment out the contents of the MULTI board setup script you are editing, using the hash character ('`#`'). Replace the script with the command sequence shown in the following (or an equivalent command sequence).

    ```
    // Reset and halt the board
    reset
    // Let the ROM image run the target
    c
    // Give the ROM image 3 seconds to set up the board
    wait -time 3000
    // Halt the board to get ready for debugging
    halt
    ```

    Note

    You may need to alter the **wait** command, depending on how long your board takes to set itself up.

    If you need to initialize your board further, continue to the next step. Otherwise, save the setup script and skip to "Customizing Linker Directives Files" on page 31.
4. Verify that your setup script begins with a command that resets the target, such as the MULTI Debugger **reset** or debug server **target tr** commands. You must halt any target before beginning any debugging activity, or the state of the target might be unpredictable.
5. Disable any interrupt sources that can disrupt the setup or destabilize the board's memory. The following steps provide a general procedure for disabling interrupt sources:

   a. Determine whether your processor has any interrupt sources that might disturb your debugging session.
   b. Using your processor's documentation, determine which registers affect interrupt sources. Then, determine the values those registers must have to disable the interrupt sources.
   c. Enter the commands necessary to configure your memory resources into your setup script (see “Useful Commands for MULTI Board Setup Scripts” on page 29 for more information).

6. Configure your target's memory controller based on your board's memory resources.

   If your memory controller and memory resources are already properly configured, skip to item number 6. The following are general steps for configuring memory using a setup script:

   a. Determine what memory resources your board has by answering the following:
      - How fast and how 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 Synchronous DRAM?
      - 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.
   c. Using your processor's documentation and memory resources, determine which memory-related registers you must set. Additionally, determine what values those registers must have to properly configure your memory resources.

   d. Enter the commands necessary to configure your memory resources into your setup script (see “Useful Commands for MULTI Board Setup Scripts” on page 29 for more information).

7. Save your setup script. For information about specifying and running the script, see the documentation about specifying setup scripts in the *MULTI: Debugging* book.

Additional Considerations for High-Speed Serial Trace

If your target uses high speed serial trace (HSST), and is not available in the New Project Wizard, you may need to update your setup script further to initialize trace properly. A setup script for an HSST target should be organized as follows:

```
reset

// configure memory and peripherals

target set hsst_rx_lanes lanes
target tracereg aurora_linerate=rate
target tracereg aurora_rx_reset=1

// reset and bring up the target side of the HSST link
```

where:

- *lanes* is the number of `rx` lanes in the connection
- *rate* is the data rate of each lane in Mbps

This information, along with the information about how to start the target side of the link, should be available in the documentation for your hardware.

Useful diagnostic functions for HSST are included in your probe installation at ***comp_install*****/ghprobe/hsst_debug.rc**. To include these functions in your setup script for troubleshooting problems, put the following line at the top of the script:

```
< comp_install/ghprobe/hsst_debug.rc
```

For more information about these functions, see the comments in **hsst_debug.rc**.

Note

MULTI tests the lane and line rate settings immediately after target reset, and disables trace if the Aurora link is not functional. If you previously had trace enabled, you may need to re-enable trace after running your board setup script.

Testing Individual Commands

If you have connected MULTI to your target (see Chapter 3, “Probe Connection Reference” on page 39), you can use the MULTI Debugger's **cmd** pane to confirm the success or failure of each command individually instead of trying to debug an entire setup script.

Some commands cannot be tested individually because they must be executed within a certain time period in relation to other commands. In this case, put the relevant commands into a small script and run the script from the Debugger's **cmd** pane using the **<** command, or type them in the same command line, separated by semicolons (`;`).

Useful Commands for MULTI Board Setup Scripts

You can find a complete list of MULTI commands in the *MULTI: Debugging Command Reference* book. However, only a small subset of these commands are useful for most setup scripts. The list below outlines these commands.

addhook Adds a hook to a Debugger action.
c Continues a stopped process.
clearhooks Removes hooks.
eval Evaluates an expression without printing the result.
halt Halts the current process.
memread Performs a sized memory read from the target, and prints the result.
memwrite Performs a sized memory write to the target.
reset Resets the target.

target
 	
Transmits commands directly to the debug server, **mpserv**.
 	
The commands available to **mpserv** are described in Chapter 5, “Probe Command Reference” on page 173. For example, to pass the **tr** command to **mpserv** using the Debugger, type:
 	
`target tr`
 	
To pass the output of a MULTI command to a debug server command, use the following syntax:
 	
`target command %EVAL{multi_command}`

wait Blocks command processing.

`$register = value`
 	
Sets a register named *register* on your target board to *value*. For example:
 	
`> $ivor0 = 0x10`
 	
`$register.field = value`
 	
Sets a field in *register* to *value*. For example:
 	
`> $CPSR.F = 1`
 	
If the name you specify for *register* is not a named register, MULTI creates a new variable using the name provided. For example, the following command creates a new variable called `$FOO` and sets its value to `6`:
 	
`> $FOO = 6`

You can also use C-style expressions for complex memory manipulation. For example: `> *((unsigned int *) 0x8000) |= 0x10`

Customizing Linker Directives Files

A linker directives (**.ld**) file controls how the linker links your executable and loads it into memory. When you create a project for a target running Stand-Alone programs using the **Project Wizard**, it creates several linker directives files and places them in the target resources project (**tgt/resources.gpj**). The linker links any project you add to your Top Project using one of these files, depending on that project's **Program Layout** setting.

By default, the **Project Wizard** sets the **Program Layout** setting for new Stand-Alone program projects to **Link to and Execute out of RAM**. Because of this setting, the linker links the program using the **standalone_ram.ld** linker directives file in your target resources project. There is another reference to this file as **tgt/standalone_ram.ld** inside your program's project.

Note

You can change the **Program Layout** for your project by right-clicking the project and choosing **Configure**. If you change the **Program Layout**, you will need to edit a different linker directives file.

To edit your linker directives file:

1. In the Project Manager, double-click the linker directives file in your program's project to open it in an editor.
2. Modify and save the edited linker directives file.
3. Select the project (**.gpj**) file in the Project Manager and click 🛠 to rebuild it.

For more information about linker directives files, see the *MULTI: Building Applications* book for your processor family.

Testing Target Resources

After you have created your new setup script, open your project in the MULTI Debugger. At the bottom of the Debugger window is the **cmd** pane, which you can use to send commands to MULTI. To run your project's default setup script:

- In the **cmd** pane, type `setup`.

After running this command, test that your target is correctly initialized by performing the diagnostics in the following sections.

Testing Register Access

To test the probe's ability to access your target's registers:

1. In the Debugger's **cmd** pane, enter the following command to halt your target:

   ```
   > target th
   ```

   `target` is required because `th` is a debug server command, meaning that it is sent directly to your probe and is not interpreted by MULTI.
2. Run the register access test by entering the following command:

   ```
   > target vr
   ```

   This command writes to all general purpose registers and reads back the values to verify that they have changed. If the test is successful, you should see the following message:

   ```
                  Testing registers. [100%]
   Test passed.
   ```

   The test was not successful if you see a number lower than `100%`.
3. Perform a JTAG stress test by repeating the test 100 times with the following command:

   ```
   > target vr 100
   ```

   If the test does not pass, you may need to lower your JTAG clock speed. If you continue to have problems, it is likely that there is a problem with your debug connection.
4. If register access is working, enter the following command to make sure that you can read all named registers:

   ```
   > target rr *
   ```

   If the output from this command is reasonable, the test was successful. If all registers have a value of `0`, your target may be held in reset.

After running the **vr** diagnostic, you should also test that you can access an explicit register. To test your ability to access a register:

1. Select a general purpose register (for example, `r1`).
2. Read the register:

   ```
   > $r1
   ```

3. Write a different value to the same register:

   ```
   > $r1=0xdeadbeef
   ```

4. Read the register again and see if it has changed to the new value. If the new value is incorrect, but some of the bits match, check the definition of that register. It is possible that some of the bits in the register are reserved and the test was successful.

Testing Memory Access

To test the your ability to access the target's memory:

1. If you have not run your setup script, enter `setup` in the Debugger's **cmd** pane.
2. Select a location in memory where you plan to download a program (for example, `0x8000`).
3. Read the memory at this location by entering the following command:

   ```
   > memread 4 0x8000
   ```

4. Write a different value to the same memory location:

   ```
   > memwrite 4 0x8000 0xdeadbeef
   ```

5. Read the memory location again and see if it has changed to the new value. If it has, the debugging interface is successfully accessing your target's memory.

If you can successfully read and write memory at a single address, test a greater range of addresses using the **vm** diagnostic. For example, use the following command to test 16 KB of memory starting at `0x8000`, using 4-byte accesses:

```
target vm 4 0x8000 0x4000
```

If **vm** passes, you should see `test passed` in its output. Try repeating the test with different access sizes (1, 2, 4, and 8). If it fails, use the following steps to determine if it is the read or the write that is causing the failure:

1. Read the same range of memory multiple times using the **md** command and compare the results. For example, enter the following command multiple times:

   ```
   > target md 0x8000 0x4000
   ```

   If the test output is different each time, the read is failing.
2. If the data is the same each time, it is likely that the write is failing. To determine if certain writes fail, write to the memory using the **mf** command and read it back using **md**. Try writing both zero and non-zero values. For example:

   ```
   > target mf 0x8000 0x100 0x0
   > target md 0x8000 0x100
   > target mf 0x8000 0x100 0xffffff
   > target md 0x8000 0x100
   ```

   If you find that accesses to memory ending in `0x0` to `0x3` work, but accesses ending in `0x4` to `0x7` are incorrect, you may need to configure the **32_bit_bus** option. For more information, see “PowerPC” on page 124.
3. If the writes are all failing, try writing just `0x20` bytes at a time. If that works, double this number until it fails. If this happens, it may indicate that there is a stuck address bit or a memory controller initialization problem that causes the same data to be read back from multiple addresses.

Testing Run Control

After confirming that you can access your target's registers and memory, confirm that run control is functioning by performing the following tests:

1. Make sure that exceptions and external interrupts are disabled on your target.
2. In the Debugger's **cmd** pane, type the following command:

   ```
   > target vc address
   ```

where *address* is a location in memory to which **vc** can download a small test program. This command runs several different tests to verify that run control works.

3. If **vc** passes, run control is probably working. If it fails, try running each individual test to determine which part of the test is failing. These test commands, documented in "Test Commands" on page 212, are:
    - `ve` *`address`* — verifies endianness.
    - `vrh` *`address`* — verifies the probe's ability to run and halt the target.
    - `vsi` *`address`* — verifies the probe's ability to single step the target.
    - `vbp` *`address`* — verifies the probe's ability to set software breakpoints.
    - `vbph` *`address`* — verifies the probe's ability to set hardware execute breakpoints.

If these tests seem to fail randomly, the JTAG clock speed may be set too high. Try lowering the clock speed using the **set clock** command and running the tests again.

Note

If your target does not have hardware breakpoints, **vc** does not test them. If your target has only one hardware breakpoint that is reserved to enable software breakpoints, **vc** does not test hardware breakpoints. If you disable software breakpoints on one of these targets (for example, with `disable_swbp`), **vc** tests hardware breakpoints but not software breakpoints.

Troubleshooting Run Control Problems

This section covers common problems you may encounter when testing run control.

All Run Control Tests Pass Except vbp and vbph

If **ve**, **vrh**, and **vsi** pass, but **vbp** and **vbph** do not, try turning off the instruction cache and putting the data caches into write-through mode.

vc Causes Exceptions On Some Power Architecture Targets

If you get an exception when running **vc** on an 85xx, 55xx, 56xx, P1xxx, or P2xxx, you may need to set the `MSR[SPE]` bit.

If your target is a 55xx or 56xx, the address you specify for **vc** must be located in memory where VLE is disabled.

vc or vrh Issues Error 13

If you get an error similar to the following when running **vc** or **vrh**:

```
ERROR 13 (test failed): PC (0x00000500) < base address
(0x00009000)
```

the probe is indicating that the program counter is outside the expected range. This happens when an exception takes place, causing the processor to jump to the exception vector. It may also indicate that the registers do not match, because they were overwritten by the exception vector:

```
ERROR 13 (test failed): Mask register (0) was corrupted.
It should be 0xf8d0f1a1 but it is 0x000032d8.
```

If you receive either of these errors, disable external interrupts and try running the test again.

Tip

On Power Architecture, external interrupts are enabled by the `MSR[EE]` bit (`0x8000`).

Using Timestamps to Collect Real World Timing Information

When you collect trace data from your target using the SuperTrace probe, MULTI trace analysis tools normally display the duration of function calls in terms of the number of instructions executed for each call. While this information is very useful for uncovering some inefficiencies, it does not provide real world time measurements that show when one event happens relative to another. *Timestamps* provide timing information for each event, including time where the target is sleeping or halted.

Because they describe events in terms of when they happen, timestamps are especially useful for:

- debugging performance problems with asynchronous control flow
- determining time intervals between specific events

For example, timestamps might help you determine how long your target is left waiting on an interrupt, or whether a specific interrupt is happening in regular or irregular intervals.

Timestamp information is accurate to about 1 microsecond, but varies by target. It is collected using the probe's trace clock, which has a crystal error of +/- 50 ppm and may cause visible differences in data collected over very long periods of time. It is displayed in the **Trace List** in two columns: the amount of time elapsed during the event, and the total time elapsed since the probe began collecting trace data. On average, trace data with timestamps uses 15-20% more of the trace buffer than trace data without timestamps.

To enable timestamps:

1. Open the **Trace Options** window, select the **Collection** tab, and click the **Target Specific Options** button.
2. Select the **Timestamps** option and click **OK**.

The timestamp clock resets each time it begins collecting trace data.

Targets usually group instructions and buffer trace data. Grouped instructions are all stamped with the same time. In the trace list, the first instruction in the group displays the time difference from the previous instruction, and the remaining instructions in the group all display a difference of `0`. For this reason, timestamps are not particularly helpful for determining the specific run time of individual instructions; they are much better suited for finding the wall-clock time between bigger picture events.

Note

Some ETM implementations keep a single byte of trace data in their internal buffers instead of outputting it over the trace port. This behavior results in inaccurate timestamps near points where trace is enabled and disabled. When using trace filters, there are many points at which trace is enabled or disabled, which increases the likelihood of inaccurate timestamps. Reducing the port size to 8 bits may help, but in general, using timestamps with filters may be unreliable on these ETM implementations.

Chapter 3

Probe Connection Reference

Contents

Before you can begin debugging with MULTI, you must first connect MULTI to the target through a *debug server* that runs on the host. The debug server that supports Green Hills Probe and SuperTrace Probe connections is called **mpserv**. **mpserv** is provided in your MULTI installation. This chapter explains how to connect MULTI to your target using **mpserv**.

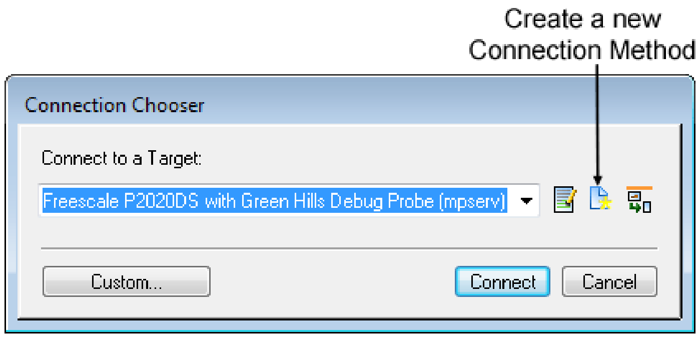
MULTI allows you to create and save *Connection Methods* that correspond to your particular host, target systems, and your desired debugging options. Once you have created at least one Connection Method, you can connect to a target by selecting a Connection Method from a list of available methods in the **Connection Chooser** or the **Connection Organizer**.

For a full explanation of how to create and work with Connection Methods, see the documentation about connecting to your target in the *MULTI: Debugging* book. The following sections describe how to configure the specific settings available for connections to Green Hills Debug Probes.

Creating a Standard Connection Method for Green Hills Debug Probe (mpserv) Connections

To create a Connection Method for your Green Hills Debug Probe:

1. Click from the MULTI Launcher and select **Connect** to open the **Connection Chooser**:

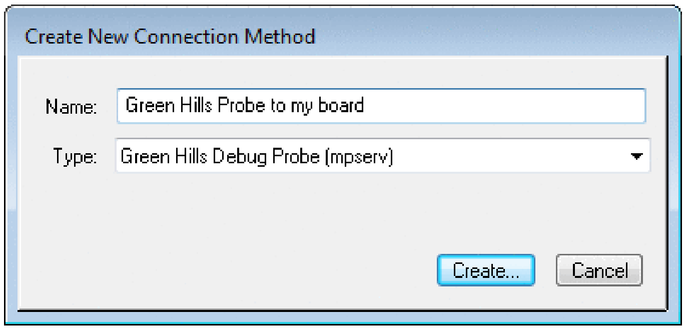


Freescale P2020DS with Green Hills Debug Probe (mpserv)

Note

You can also open the Connection Chooser from the MULTI Project Manager or Debugger. For instructions, see the documentation about connecting to your target in the *MULTI: Debugging* book.

2. Click to open the **Create New Connection Method** dialog box.



Name: Green Hills Probe to my board
Type: Green Hills Debug Probe (mpserv)

3. Enter a name for your Connection Method (for example, `Green Hills Probe to my board`).
4. Select **Green Hills Debug Probe (mpserv)** from the drop-down list of connection types.
5. Click **Create**.

The **Green Hills Debug Probe (mpserv) Connection Editor** for your new Connection Method opens. You must use the fields and settings described in the next section to edit your Connection Method before you use it for the first time.

The Green Hills Debug Probe (mpserv) Connection Editor

A **Connection Editor** is a target-specific GUI that allows you to configure and edit Connection Methods. The **Connection Editor** window for Standard Connection Methods is divided into the following four sections:

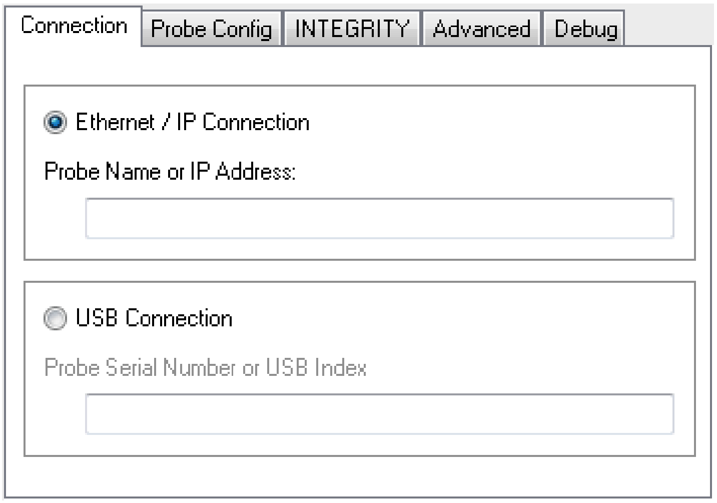
- General information
- Target-specific fields
- Command line viewer
- Action buttons

The command line viewer, the action buttons, and the fields that appear in the general information section, are consistent across all **Connection Editors**, and are described in detail in the documentation about the Connection Editor in the *MULTI: Configuring Connections* book.

In addition to these generic fields that appear on all Connection Editors for Standard Connection Methods, the **Green Hills Debug Probe (mpserv) Connection Editor** includes that provide settings and options specific to your target and host operating systems.

When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method can be used.

Green Hills Debug Probe (mpserv) Connection Settings



● Ethernet / IP Connection

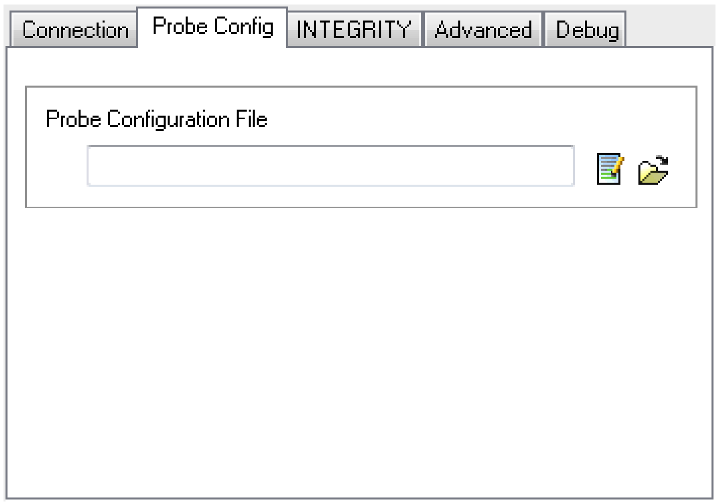
Probe Name or IP Address:

○ USB Connection

Probe Serial Number or USB Index

Ethernet/IP Connection	Specifies an Ethernet/IP connection. If you select an Ethernet connection, you must provide a hostname or IP address in the **Probe Name or IP Address** text field.
Probe Name or IP Address	Specifies the hostname or IP address of your Green Hills Probe. This field is only available if you select the Ethernet/IP radio button.
USB Connection	Specifies a USB connection. This button is only enabled if you are running Windows.
Probe Serial Number	Specifies the serial number of your probe to determine which probe to use if multiple probes are connected to your computer over USB.

Green Hills Debug Probe (mpserv) Probe Config Settings

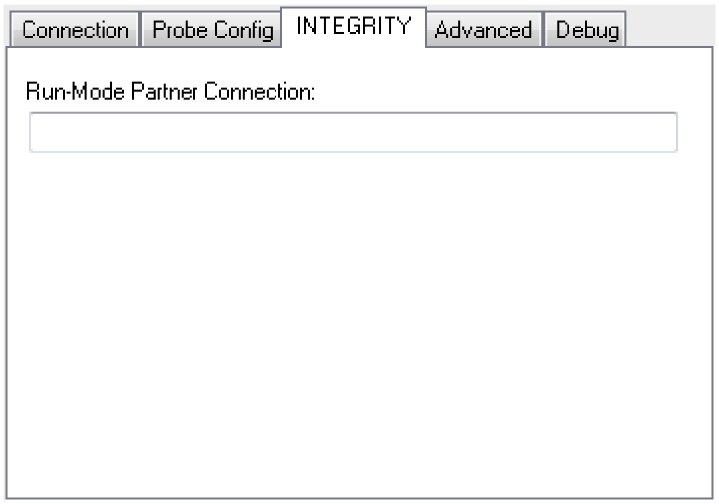


Probe Configuration File

Probe Configuration File	Specifies the probe configuration file to load onto the probe when the connection starts. **Note:** Some probe configuration files require rebooting the probe. Loading these files at connection is only supported with MULTI 7 and later.

For more information about probe configuration files, see “Loading and Saving Configuration Files” on page 20.

Green Hills Debug Probe (mpserv) INTEGRITY Settings



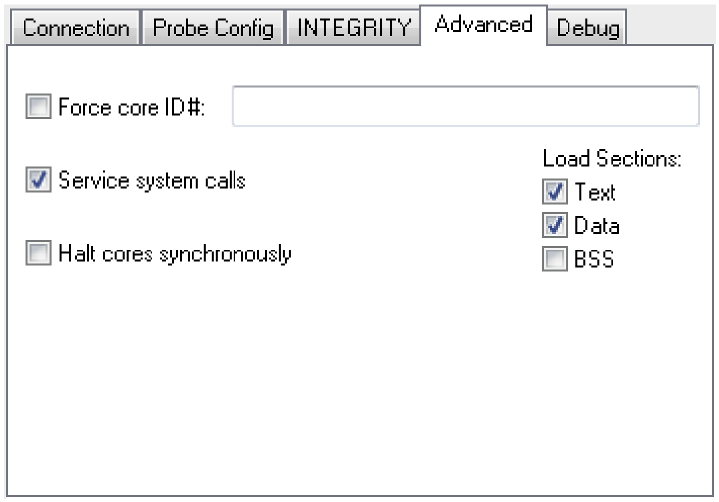
Run-Mode Partner Connection:

Run-Mode Partner Connection	Specifies the run-mode connection that the Debugger automatically attempts to establish when you boot an INTEGRITY kernel via the current freeze-mode connection.

For more information about this connection mode for an INTEGRITY system, see the documentation about automatically establishing run-mode connections in the *MULTI: Debugging* book.

For information about Probe Run Mode, a feature that allows you to create a run-mode partner when your target does not have Ethernet or serial drivers, see “Probe Run Mode” on page 59.

Green Hills Debug Probe (mpserv) Advanced Settings



- [ ] Force core ID#:
- [✓] Service system calls
- [ ] Halt cores synchronously

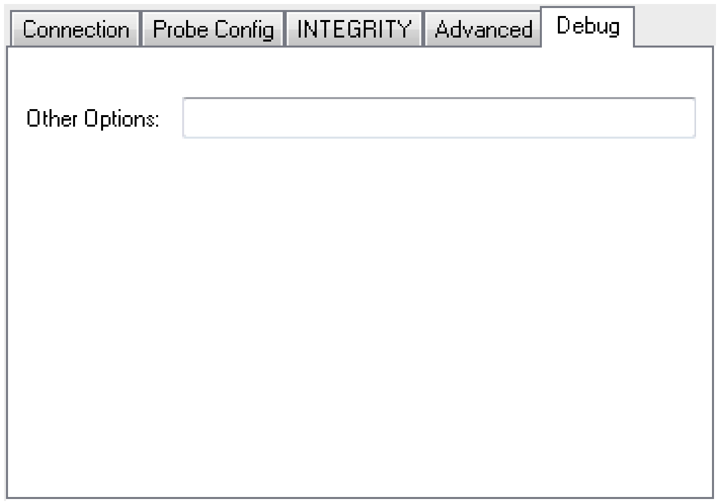
Load Sections:
- [✓] Text
- [✓] Data
- [ ] BSS

Warning

Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

Force core ID#	Limits the collection of debug information to a subset of cores. This option allows you to use multiple instances of MULTI to connect to different subsets of cores in a multi-core system, including connection to a single core. If you select this option, you must enter core ID(s) in the text field. For information about specifying cores ID(s) see **-force_coreid** in “Options for Custom Connection Methods” on page 49. Use the probe command **tl** to view a list of cores and their corresponding IDs. This option is only supported by the Green Hills Probe.
Service system calls	Enables host-based system calls. If this box is selected, system calls are enabled and are implemented with a single software breakpoint. This box is checked by default. If you are running from ROM and software breakpoints cannot be used, it may be desirable to disable host-based system calls by clearing this box.
Halt cores synchronously	If this box is selected, the probe will enforce that the cores are either all running or all halted, which is useful for debugging SMP and other multicore systems. Run and halt will affect all cores, and a breakpoint hit on one core will cause all other cores to halt. We recommend selecting this option if you plan to use software breakpoints in code that is shared by multiple cores. This feature is only supported with MULTI 7 and later. The feature is currently only supported with ARM Cortex-A and POWER e500mc, e5500, and e6500 targets. This option will be ignored if connecting to a single core. Use the **Force Core ID** field to limit this behavior to a subset of cores.
Load Sections	Controls which sections of your program will be downloaded to the target, as follows: • **Text** — If this box is checked, the `.text` (code) sections of your program will be downloaded to the target. This box is checked by default. • **Data** — If this box is checked, the `.data` (initialized data) sections of your program will be downloaded to the target. This box is checked by default. • **BSS** — If this box is checked, the `.bss` (uninitialized data) sections of your program will be cleared. This box is not checked by default.

Green Hills Debug Probe (mpserv) Debug Settings



Warning

Do not change the settings on the **Debug** tab unless you are instructed to do so by Green Hills Technical Support.

Other Options	Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.

Using Your New Connection Method to Connect

After you have specified your desired settings for your new Connection Method, click **OK** to apply your changes and save the Connection Method. The **Connection Editor** closes and returns you to the **Connection Chooser**, which displays the name of your new connection. Click **Connect** to connect to your target.

For information about other ways to connect to your target, as well as information about temporary connections and how to use the **Connection Chooser** and **Connection Organizer** to edit and manage your Connection Methods, see the documentation about connecting to your target in the *MULTI: Debugging* book.

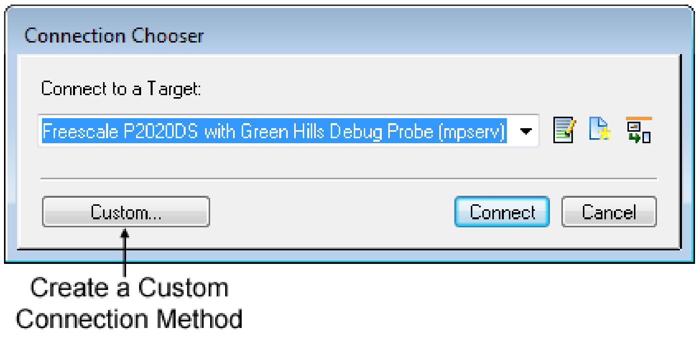
Using Custom Connection Methods

Advanced users and/or users of earlier versions of MULTI who prefer to use command line options rather than the GUI interface can create *Custom Connection Methods*. Like Standard Connection Methods, Custom Connection Methods can be saved, invoked quickly using the **Connection Chooser**, and managed using the **Connection Organizer**. General instructions for using Custom Connection Methods are given in the documentation about Custom Connection Methods in the *MULTI: Debugging* book.

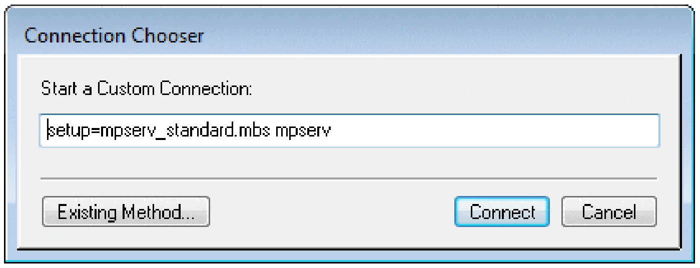
This section gives the specific commands and syntax for **mpserv** Custom Connection Methods.

To create a Custom Connection Method for your Green Hills Debug Probe:

1. Click from the MULTI Launcher and select **Connect** to open the **Connection Chooser**:


Connection Chooser
Connect to a Target:
Freescale P2020DS with Green Hills Debug Probe (mpserv)
Custom...
Connect
Cancel
Create a Custom Connection Method

2. Click **Custom**. The dialog box displays **Start a Custom Connection** and a text field.



setup=mpserv_standard.mbs mpserv

3. Enter the following command:

[setup=*filename*.mbs] mpserv *connection* [*options*]...

where:

- **setup=*filename*.mbs** is optional and specifies the target setup script. This argument is optional because not all targets require setup scripts.

  Note

  This option can only be used to specify an **.mbs** setup script. If you are using an older **.dbs** script, you must use the **-setup** option, which must precede the **mpserv** command. See Chapter 2, "Configuring Your Probe" in the *Getting Started* book for your probe for more information about setup scripts.

- ***connection*** is required and specifies the type of connection and, if applicable, the IP address of the target, as listed below.
  - For Green Hills Probe Ethernet connections, *connection* is the hostname or IP address of the target.
  - For Green Hills Probe USB connections, *connection* is **-usb [*serial*]**, where *serial* is optional. It can be used to specify a particular probe if multiple probes are connected to a single host. If *serial* is not specified, the probe is selected in a non-deterministic way.
- *options* can be any non-conflicting combination of the **mpserv** options listed in "Options for Custom Connection Methods" on page 49.

4. Click **Connect**.

MULTI will connect to your target and save your Custom Connection Method. In the future, the new method appears in the list of available Connection Methods in the **Connection Chooser** and the **Connection Organizer**. The name of your Custom Connection Method is the command line you entered. You can change this name using the **Custom Connection Editor**. See the documentation about Custom Connection Methods in the *MULTI: Debugging* book for more information.

Options for Custom Connection Methods

Options	Action
-adi	Blackfin only. Forces **mpserv** to treat the downloaded program as one compiled with the ADI VDSP compiler. This affects the way system calls are handled by the debugger.
-attach	Sets passive attachment mode, which allows **mpserv** to attach to a running target without disturbing the target.
-bss	Sets downloading of `.bss` (uninitialized data) sections. By default, `.bss` downloading is disabled because the default Green Hills startup code clears `.bss` sections. This option allows you to clear `.bss` sections upon download.
-cfgload	Specifies the probe configuration file to load onto the probe when the connection starts. Note that some probe configuration files require rebooting the probe. Loading these files at connection is only supported with MULTI 7 and later.
-data	Sets initialized data downloading. By default, initialized data (`.data`) sections are always downloaded, so this switch should never be used. This option is documented for backward compatibility only.

Options	Action
-force_coreid ***cores***	Limits the collection of debug information to a subset of cores specified by *cores*. *cores* is a comma separated list of core groups. A core group can be specified as follows: • `*` — selects all cores. • A single decimal number — selects one core. • `startcore..endcore` — selects a contiguous group of cores. For example, both `1..3` and `3..1` select cores 1, 2, and 3. Additionally, only one of the bounds is required, so `1..` selects cores 1, 2, 3, etc., and `..5` selects cores 0, 1, 2, 3, 4 and 5. There should be no whitespace within the *cores* string. Core groups may overlap, and it will not result in MULTI seeing the same core twice. Non-debuggable cores in core groups are ignored. This option allows you to use multiple instances of MULTI to connect to different subsets of cores in a multi-core system, including connection to a single core. When you use this option, MULTI will act only on the cores specified, except that **groupaction** on the `@All` group will still affect all cores, even those not included in the *cores* subset. To direct **groupaction** commands to only the tasks listed in the Debugger, first create a task group corresponding to those cores, and then use it as the `@task_group` parameter to the **groupaction** command. For more information, see "groupaction" in Chapter 18, "Task Group Command Reference" in *MULTI: Debugging Command Reference*. Use the probe command **tl** to view a list of acceptable values for *cores*. The **-force_coreid** option only applies to Green Hills Probe connections.
-hwbpsc	Specifies that the probe uses a hardware breakpoint (instead of a software breakpoint) to catch system calls made by the program running on the target. This option requires that the target has at least one execute hardware breakpoint available, and that execute hardware breakpoints are precise. This argument may be useful when the system call function is in memory where software breakpoints do not work (such as read-only memory).

Options	**Action**
-loadall	Enables downloading of all program sections. By default, all sections are loaded except for uninitialized data (`.bss`) sections, which are cleared by the default Green Hills startup code. The **-loadall** option allows you to download all sections, including `.bss`.
-load_pheader_translate	Adds support for ELF files created by the ARM Ltd. toolchain. Use this option only if your program was compiled or linked using this toolchain.
-log	Creates a log of actions performed by **mpserv** and stores it in a file named **db.log**. If you have problems using **mpserv** with your target, this log file can help Green Hills Technical Support find a solution.
-nexus_trace_coreid *n*	Specifies the core from which to collect trace data on Nexus targets. The default is 0. To specify the second core, pass the option `-nexus_trace_coreid 1`.

Options	Action

-no_trace_registers	Disables access to ARM trace registers until the **trace_registers on** command is used.
	
	When a probe is first connected to the target, MULTI attempts to read some registers on the target to determine what trace capabilities the target has. If your target requires setup before those registers can be safely accessed, you can use the **-no_trace_registers** command line argument and the **trace_registers** command to do that setup.
	
	Create a setup script with the following structure:
	
	`MBS_OPT="early"`
	
	`define trace_register_setup() {`
	`    // Put any necessary trace setup here`
	`    ...`
	`    target trace_registers on`
	`}`
	
	`clearhooks`
	`addhook -after connect {`
	`   trace_register_setup()`
	`}`
	`addhook -before reset {`
	`   target trace_registers off`
	`}`
	`addhook -after reset {`
	`   trace_register_setup()`
	`}`
	
	Then, pass the **-no_trace_registers** argument to **mpserv**. This will ensure that your target's trace registers are enabled on connect and after a reset, and will prevent MULTI from trying to read the trace registers prior to initialization. Using the **trace_registers** command outside a setup script hook like the one described is not supported and may result in incorrect trace capture.
	
	Supported on ARM only.
	
	For more information, see the documentation for the **trace_registers** command in "Other Commands" on page 207.

-noadi	Blackfin only. Forces **mpserv** to treat the downloaded program as one compiled with the Green Hills Compiler. This affects the way system calls are handled by the debugger.

Options	Action
-nobss	Disables uninitialized data (`.bss`) clearing by **ocdserv**. This option is enabled by default and is documented for backward compatibility only.
-nodata	Disables downloading of initialized data (`.data`) sections.
-noload	Disables downloading of any of the program sections in an executable file. This option is useful when debugging an executable that is already loaded on the target. For example, use this option when debugging an executable in ROM or flash memory. This option does not disable stack setup processes, such as setting the stack pointer, writing the stack to memory, or setting the system call breakpoint.
-nosyscalls	Disables system calls. By default, **mpserv** places a breakpoint on the `.syscall` section to service system calls on the target. However, some applications may place the `.syscall` in read-only memory, while others may implement their own system call mechanism with `.syscall`. The **-nosyscalls** switch prevents conflicts between **mpserv** and such applications.
-notext	Disables downloading of program sections containing executable code (`.text` sections).
-prm_coreid *core*	Specifies the core used to start Probe Run Mode. Use the probe command **tl** to view a list of acceptable values for *core*. Specifying an invalid *core* prevents Probe Run Mode from starting. This option overrides any checks that the probe would normally make to test your target for live memory access. It overrides these checks so that you can start on one core while dedicating a different core to live memory access (such as on x86).

Options	Action
-profileon *filename*	Specifies that you are using profiling on the executable *filename*. Use this option if you have enabled profiling and at least one of the following is true: • You are using the **-attach** option. • You used the **Program already present on target** or **Program Flash ROM** option in the **Prepare Target** dialog box. • Your linker directives (**.ld**) file allocates program text to ROM, whether or not startup code copies it from ROM to RAM.
-set *option*=*value*	Sets the configuration option, *option*, to *value* before beginning the current session. Any configuration changes made using this option is saved to `EEPROM`. See Chapter 4, “Probe Option Reference” on page 65 for a complete description of the available configuration options.
-setup *filename*	Specifies a setup script file, *filename*, to be run by the **mpserv** command interpreter before each download. This option can only be used to specify **.dbs** setup scripts. See Chapter 2, “Configuring Your Probe” in the *Getting Started* book for your probe for information about specifying **.mbs** setup scripts.
-synccores	Specifies a synchronous run control environment, where all cores on the target are always resumed and halted at the same time. For more information, see the documentation for **Halt cores synchronously** in “Green Hills Debug Probe (mpserv) Advanced Settings” on page 44.
-trace_triggers	Enables trace triggers for PowerPC 405, 440, and 460. Passing this option has the same effect as issuing the following command in the MULTI command pane: `target trace_triggers on` For more information, see the documentation for the **trace_triggers** command in “Other Commands” on page 207.

Options	Action
-usb [*serial*]	Connects to a Green Hills Probe using the USB port. The optional *serial* parameter can be used to specify a particular probe by serial number if multiple probes are connected to a single host. If multiple probes are connected and no *serial* is specified, the probe is selected in a non-deterministic way.

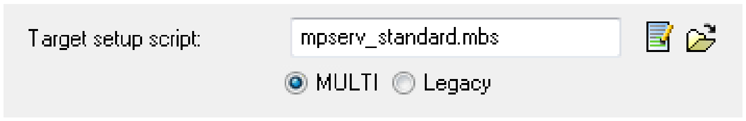
Specifying Setup Scripts

After you have a working setup script, you should run it prior to every download to ensure that your target is clean, stable, and properly configured. The following sections explain how to specify and run board setup scripts depending on how you are connecting to your target and whether you are using a MULTI (**.mbs**) or legacy (**.dbs**) board setup script.

Using MULTI (.mbs) Setup Scripts When Connecting to Your Target

The method for specifying an **.mbs** setup script at the time of connection varies depending upon the procedure you use to connect MULTI to your target:

- To run an **.mbs** setup script every time you connect using a particular Standard Connection Method, specify the filename of the script in the **Target Setup script** field of the **Connection Editor** for the Connection Method and select the **MULTI** radio button immediately below the field.

  Target setup script: mpserv_standard.mbs

  [✓ MULTI] Legacy

  If you are using a default Connection Method created by the **Project Wizard**, the necessary setup script file for your processor-board combination (if applicable) is specified automatically and the **MULTI** button will be selected.

- To run an **.mbs** setup script and connect to your target using a Custom Connection Method:
  - If you are editing the Custom Connection Method using the **Connection Editor**, specify the filename of the target setup script in the **Target Setup script** field.

  - ◦ If you are entering the connection command using the **Start a Custom Connection** field of the **Connection Chooser**, precede the debug server command with the option **setup=*filename*** (where *filename* is the **.mbs** setup script filename). Click **Connect** to continue.

- To run an **.mbs** setup script when connecting from the Debugger command pane, use the following syntax:

  connect setup=*filename*.mbs *dbserv* [*args*]... [*opts*]...

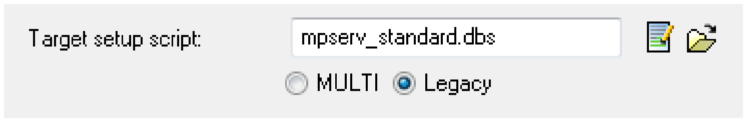
  where ***filename*.mbs** is the setup script filename, *dbserv* is the name of the debug server to be used, and *args* and *opts* are appropriate arguments for your debug server and target.

  For more information about the **connect** command, see the documentation about general target connection commands in the *MULTI: Debugging Command Reference* book and the documentation about connecting to your target in the *MULTI: Debugging* book.

Using Legacy (.dbs) Setup Scripts When Connecting to Your Target

The method for specifying a legacy (**.dbs**) setup script at the time of connection varies depending upon the procedure you use to connect MULTI to your target. The various methods are:

- To run a legacy setup script every time you connect using a particular Standard Connection Method, specify the filename of the target setup script in the **Target Setup script** field of the **Connection Editor** for the Connection Method. Select the **Legacy** radio button immediately below the field.



Target setup script: mpserv_standard.dbs
MULTI [✓ Legacy]

- To run an **.dbs** setup script and connect to your target using a Custom Connection Method:

  - ◦ If you are editing the Custom Connection Method using the **Connection Editor**, include the **-setup *filename*.dbs** debug server option in the **Arguments** field.

- ◦ If you are entering the connection command using the **Start a Custom Connection** field of the **Connection Chooser**, include the **-setup *filename*.dbs** debug server option in the command you enter and click **Connect**. See the appropriate debug server chapter later in this book for more information about connecting to your specific target this way.

  To run a legacy setup script and connect to your target using a Custom Connection Method, include the **-setup *filename*.dbs** debug server option in the command you enter into the **Start a Custom Connection** field of the **Connection Chooser** and then click **Connect**. See the appropriate debug server chapter later in this book for more information about connecting to your specific target this way.

- To run a **.dbs** setup script from the Debugger command pane, use the following syntax:

  connect *dbserv* -setup *filename***.dbs** [*args*]... [*opts*]...

  where ***filename*.dbs** is the setup script filename, *dbserv* is the name of the debug server to be used, and *args* and *opts* are appropriate arguments for your debug server and target.

Running Setup Scripts Manually

In addition to running setup scripts as part of the connecting process, you can also run setup scripts manually at other times using any of the following methods:

- (MULTI **.mbs** scripts) Run your setup script file manually from the Debugger command pane using the **<** command.
- (MULTI **.mbs** scripts) Use the MULTI **setup *filename*.mbs** command. If this command is used with a connection command, the setup script runs prior to downloading and debugging. The **setup** command can also be used without a specific script name if you are connected and specified a setup script when you connected. The script you specified for the connection is run if you issue the **setup** command with no specified file. See MULTI: Debugging Command Reference for more information about the **setup** command.
- (Legacy **.dbs** scripts) Use the **target script *filename*.dbs** command from the Debugger command pane.

Early MULTI Board Setup Scripts with Debugger Hooks

Ordinarily, MULTI (**.mbs**) board setup scripts are run every time you download a program to your target, just before the download begins. However, in certain circumstances and for certain targets (such as multi-core boards), it is not appropriate to re-initialize the entire board every time you download a program to a given CPU on that board.

If you write a setup script with a comment on the first line containing the marker `MBS_OPT="early"`, the entire board setup script will be run once immediately after you connect to your target instead of every time just before you download a program. The comment should look similar to the following:

```
// MBS_OPT="early"
```

When combined with the **Hook Commands** described in the documentation about scripting commands in the *MULTI: Debugging Command Reference* book, the "early" MULTI board setup script mechanism gives you fine-grained control over how and when MULTI will set up your target. For example, you can install hooks in your setup script to reinitialize the entire board upon reset by using reset hooks to cause certain initializations to take place before reset and certain others to take place afterwards.

```
addhook -before reset { /* Before reset work /* }
addhook -after reset -core 0 { /* After reset, core 0 work */ }
addhook -after reset -core 1 { /* After reset, core 1 work */ }
```

You can also add a hook to reset the board upon connecting, which causes your reset hooks to run whenever you connect to the target with MULTI.

```
addhook -after connect { reset }
```

Lastly, you might decide to reinitialize a selected subset of the board circuitry every time you download a program to one of the CPUs, while leaving the other CPUs alone.

```
addhook -before download -core 0 { /* Core 0's initialization commands */ }
addhook -before download -core 1 { /* Core 1's initialization commands */ }
```

Probe Run Mode

Traditionally, INTEGRITY run-mode connections are made by connecting the host machine to the same network as the target. INTEGRITY then sends information over Ethernet to the host machine for use by the Debugger. This technique requires that your hardware has Ethernet or serial drivers. Probe Run Mode allows you make both a freeze-mode and run-mode connection to your target using just the probe's debug connection. Probe Run Mode requires the following:

- Live memory access support on the target
- An INTEGRITY BSP that supports Probe Run Mode (GipcTarget)
- The probe must be connected to the host using Ethernet; Probe Run Mode connections are not supported over USB

INTEGRITY 11 and earlier provide BSP support through patches. To see if your target is supported, contact Green Hills support. For newer versions of INTEGRITY, see the BSP notes.

Most of the time, probe run-mode is handled properly by the Debugger's automatic run-mode partner feature. To connect to Probe Run Mode manually, use your probe's IP address or hostname and port `2222` for the INDRT2 connection. For example:

```
rtserv2 192.168.1.101:2222
```

Similarly, you can connect to the MULTI **ResourceAnalyzer** using the probe's IP address or hostname.

If you are using MULTI 6.1.4 or earlier, and your target is capable of both Probe Run Mode and run mode over Ethernet, the automatic run mode partnering feature picks one in a non-deterministic way.

At this time, only one Probe Run Mode Proxy can run at a time on a probe, even if you are running independent programs on multiple cores of a multi-core system. The last Probe Run Mode-capable kernel that is loaded is always the active one.

Connecting to a Running Kernel

Follow these steps to connect to a running kernel (either downloaded previously or started from some other means such as a boot loader):

1. Optionally use the **prm subscribe all** command to monitor the progress of the connection.
2. Use the **prm start** command (for more information, see “Other Commands” on page 207) with one of the following methods:
   - If you are attached to the kernel in MULTI and halted in a kernel context (such as the Idle loop or a Kernel Task) you can simply run `target prm start` and the address will be automatically determined.
   - If the target is running or not halted in the kernel, you need to determine the physical address of the kernel's `.gipctarget` section and run `target prm start` *`address`*`p` where *`address`* is the address of the `.gipctarget` section. Note that the `p` suffix is required on the address to denote it as physical. One way to determine the proper address is to connect to PRM automatically the first time, run `target prm status`, and use the address on the `Base Address` line.
3. Resume the kernel (if it was halted in or before step 2).
4. Wait for **Probe Run Mode Communication Established**. If Step 1 was skipped, run `prm status` to check the current status.
5. Establish the rtserv2 connection to the target manually with a command such as `rtserv2` *`probe_ip`*`:2222`, where *`probe_ip`* is your probe's hostname or IP address.

See also the documentation for **prm_ignore_disconnect** and **prm_ignore_reset** options in “Generic Configuration Options” on page 66, which can be useful for situations involving an externally loaded kernel.

Serial Port Forwarding

Green Hills Debug Probes can forward information from the RS-232 serial port to a telnet session. This feature allows you to connect your target to your probe using the serial port, and initiate a serial terminal session from your host machine over Ethernet. To forward the serial port:

1. Make sure the **serial_terminal** option is set to `off`.
2. Connect your target to the probe's RS-232 serial port using a null modem cable or straight-through cable, depending on your target.
3. On your host machine, open a telnet session to your probe.

4. In the telnet session, type `pterminal serial -baud` *`rate`*, where *rate* is a baud rate compatible with your target. You may need to specify additional options for this command (see “System Commands” on page 188).

The telnet session now behaves like a serial terminal to your target. You should now see serial output in the session running **pterminal**. To quit the forwarding session, press the following three keys, in order:

1. Enter
2. Tilde (~)
3. Period (`.`)

Using External Trace Triggers

The SuperTrace Probe v3 has SMB connectors on the front panel for external trace trigger input and output that are always enabled. External trace triggers are useful for observing the relationship between code running on the CPU and a set of signals on the target. Generally:

- To observe what signal behavior causes or is caused by particular code, connect the probe's **TRIGGER OUT** connector to an external device that is monitoring the signals on the target, and configure MULTI to trigger on the code.
- To observe what code causes or is caused by particular signal behavior, connect the probe's **TRIGGER IN** connector to an external device that can trigger the probe when that behavior occurs.

External trace triggers are supported for all trace architectures except PowerPC 405.

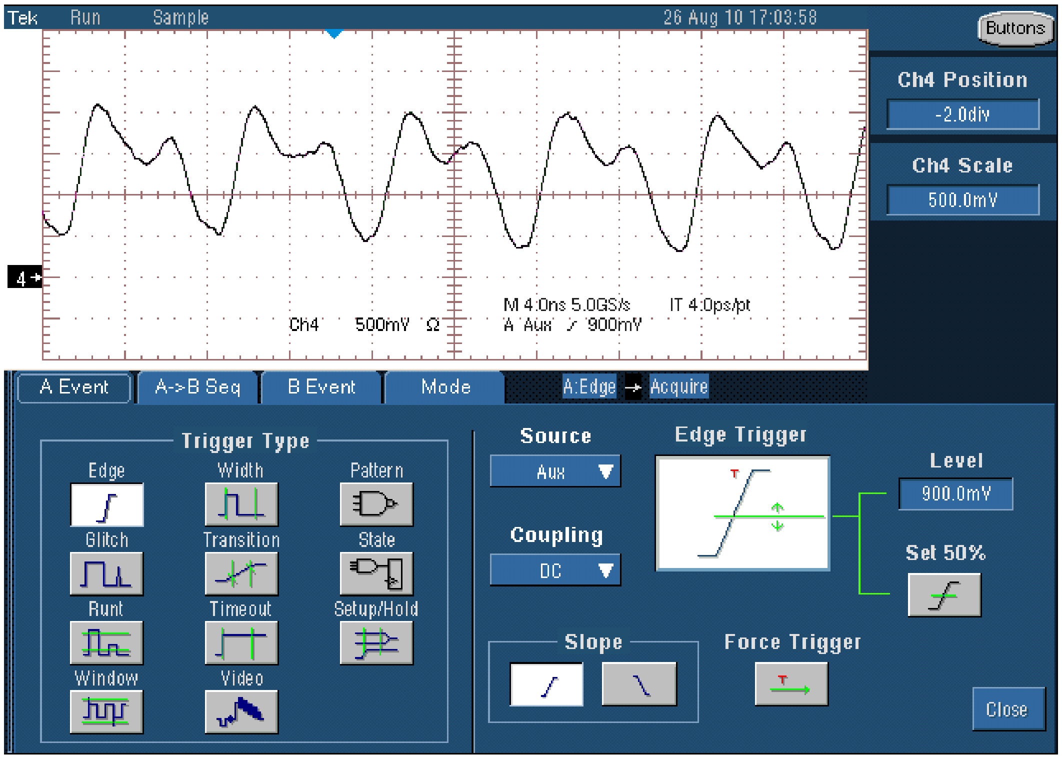
When configuring a device to accept trace trigger input from your probe's **TRIGGER OUT** connector, use the following settings:

- Edge — rising
- Voltage — 900 mV
- Coupling — DC

When configuring a device to send trace triggers to the probe's **TRIGGER IN** connector, make sure the triggers have the following characteristics:

- Voltage — 1.5 V
- Length — > 100 ns

For example, the following image shows a Tektronix TDS5104 configured to capture data when it receives a trace trigger from the probe.



Ch4 Position -2.0div
Ch4 Scale 500.0mV
Ch4 500mV Ω
M 4.0ns 5.0GS/s
IT 4.0ps/pt
A Aux ∕ 900mV
Source Aux
Coupling DC
Level 900.0mV
Slope ∕

The probe emits a pulse on the **TRIGGER OUT** connector for any trace trigger event, including an external input trigger. Whenever the probe detects a pulse on the **TRIGGER IN** connector, it triggers trace collection. To stop using the probe's external trace trigger in or out functionality, disconnect the appropriate cable.

For information about configuring trace triggers based on program events, see the documentation about configuring trace collection in the *MULTI: Debugging* book.

Note

Due to buffering on your target and trace pod, when the probe acts on an external trigger from the **TRIGGER IN** port, there may be some skew between the trigger event in the trace list and the code or signal that happened at that time. The skew is dependent on your target; it is typically about 200 nanoseconds, but may be as much as 1 microsecond.

General Purpose IO Port Drive Characteristics

All probe models except the SuperTrace Probe v3 have general purpose IO (GPIO) pins. The electrical drive characteristics of these pins are 3.3 V, 16mA. The connector should be similar to the Molex 50-57-9404 (4 position .100 with latch).

Chapter 4

Probe Option Reference

Contents

This chapter provides descriptions of options that you can use to customize your debugging configuration.

Setting Configuration Options

The Green Hills Debug Probe configuration options described in this chapter are divided into two groups:

- The options listed in "Generic Configuration Options" on page 66 are available for all target architectures (unless otherwise noted).
- The options listed in "Target-Specific Configuration Options" on page 92 work only for certain target architectures (as indicated in the tables in that section).

To display or configure any of the options listed in this chapter, use the Probe Administrator. For more information about the Probe Administrator, see "The Graphical Probe Administrator" on page 6.

Alternatively, you can use the **set** command:

- In the MULTI Debugger **Target** pane.
- In the MULTI Debugger **Command** pane (if prefixed with the **target** command).
- In a telnet or serial terminal window.

For a description of the **set** command, refer to the table in "Green Hills Debug Probe Commands" on page 174.

Note

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

Generic Configuration Options

This section briefly describes configuration options that are applicable to most targets. Unless otherwise indicated, these options work independently of both the target architecture type and the type of Green Hills Debug Probe being used. If an option is not supported for all probe configurations, use the following abbreviations determine which configurations the option applies to:

- **STP** — SuperTrace Probe
- **V3** — Green Hills Probe v3.
- **TEP** — TraceEverywhere Pod

For more details about the syntax and possible settings for each of these general configuration options, and for further information about the features controlled by each option, follow the cross references provided in the table.

For additional configurations options that are available for your particular target, see "Target-Specific Configuration Options" on page 92.

For general information about setting configuration options, see "Setting Configuration Options" on page 66.

adapter Specifies which type of target adapter is attached to your probe. See "Setting the Target Adapter Type" on page 74.
baudrate Sets the serial communication speed. See "Setting the Serial Communication Speed" on page 82.
checker Turns the status checker on or off. See "Enabling or Disabling the Status Checker" on page 82.
clock Sets the JTAG clock speed. See "Setting the JTAG or SWD Clock Speed" on page 82.
debug_type Selects the protocol to use for target communication. See "Selecting the Target Communication Protocol" on page 71.
dhcp (Ethernet connections only.) Allows you to use dynamically assigned IP addresses. See "Enabling and Disabling DHCP" on page 71.

dhcp_lease (Ethernet connections only.) Sets the lease time in days requested for a DHCP address. Set the lease time to 0 days for an infinite lease. The default setting is an infinite lease. See “Enabling and Disabling DHCP” on page 71.	
endianness Specifies the byte order of the target. See “Setting the Byte Order (Endianness)” on page 88.	
gateway Sets the gateway of the probe. See “Setting the Gateway” on page 73.	
hostname (Ethernet connections only.) Sets the hostname that DHCP requests send. The default setting is **ghprobe** with the probe's serial number appended. For example, a probe with a serial number of 4000 would use **ghprobe4000** as its default DHCP hostname. See “Enabling and Disabling DHCP” on page 71.	
hsst_rx_lanes Sets the number of lanes in the Aurora link from the target to the probe. See “Setting the Number of Receive Lanes” on page 73.	**STP v3 only**
ip Sets the IP address of the probe. See “Setting the IP Address” on page 70.	
jrst_pulse Specifies how long the nTRST line will be held low. See “Setting the JTAG TAP Reset Pulse Length” on page 86.	
jrst_settle Specifies how long to wait after an nTRST pulse. See “Setting the JTAG TAP Reset Settle Length” on page 87.	
jtag_drive Specifies the JTAG signal drive strength for TraceEverywhere pods. See “Setting the Drive Strength of TraceEverywhere Pods (SuperTrace Probe)” on page 88.	**TEP only**
logic_high Specifies the I/O interface voltage. See “Setting the I/O Interface Voltage” on page 75.	
netmask Sets the netmask of the probe. See “Setting the Netmask” on page 73.	

power_detect Enables or disables power detection. See "Enabling or Disabling Target Power Detection" on page 83.	
power_on_settle Sets the time (ms) in addition to `rst_settle` that the probe waits after detecting power on, before attempting any other communication with the target. See "Setting the Reset Settle Length" on page 86.	
prm_ignore_disconnect Specifies whether or not the Probe Run Mode Proxy persists when the connection that started the Proxy exits. See "Configuring Probe Run Mode Behavior on Host Disconnect" on page 83.	
prm_ignore_reset Specifies whether or not the Probe Run Mode Proxy continues running when a target reset is initiated. See "Configuring Probe Run Mode Behavior on Target Reset" on page 84.	
rst_pulse Specifies how long the nRST will be held low during a reset. See "Setting the CPU Reset Pulse Length" on page 86.	
rst_settle Specifies how long to wait after an nRST pulse is issued. See "Setting the Reset Settle Length" on page 86.	
serial_terminal Turns the probe's serial terminal console interface on or off. See "Enabling or Disabling the Serial Terminal" on page 84.	**V3 and STP v3 only**
single_mpserv_only Specifies whether the probe allows multiple binary **mpserv** connections. See "Disabling Multiple Binary Connections" on page 85.	
target Configures your target. See "Specifying Your Target" on page 76.	
target_reset_pin Specifies to the probe how asserting the reset pin (nRST) affects JTAG TAP. See "Setting Reset Pin and JTAG TAP Interaction" on page 87.	
trace_clock_delay Adjusts the clock-to-data timing requirements of the SuperTrace Probe. See "Setting the Trace Clock Delay (SuperTrace Probe)" on page 90.	**TEP**

trace_clock_phase Adjusts the clock-to-data timing requirements of the SuperTrace Probe. See "Setting the Trace Clock Phase (SuperTrace Probe)" on page 89.	**STP**
trace_clock_source Controls clock-to-data timing used for capturing trace data. See "Setting the Trace Clock Source (SuperTrace Probe)" on page 88.	**STP**
trace_logic_high Specifies the trace input voltage on TraceEverywhere Trace Pods. See "Setting the I/O Interface Voltage" on page 75.	**TEP only**
trace_term_enable Controls the trace termination circuitry on TraceEverywhere pods. See "Setting Trace Termination Control on TraceEverywhere Pods (SuperTrace Probe)" on page 91	**TEP**
user_button Controls what happens when you press the **User** button on the probe's front panel. See "Setting the User Button Behavior" on page 91.	
user_string Sets the user string of the probe. See "Setting the User String" on page 85.	
verify_download Specifies whether a downloaded image is read back from the target to compare to the downloaded image, and if so, whether the full image is read back and compared, or a small subset of the downloaded image is read back and compared. See "Specifying Download Verification" on page 91.	

Setting the IP Address

To set the IP address for your Green Hills Probe or SuperTrace Probe, use the command:

```
set ip ip
```

where *ip* is the IP address, specified in dotted quad notation. This setting does not take effect until the next reboot.

Example:

```
set ip 192.168.0.5
```

This command sets the probe IP address to `192.168.0.5`.

Selecting the Target Communication Protocol

To select the protocol to use for target communication, use the command:

```
set debug_type protocol
```

where *protocol* is one of the following:

- **JTAG**— [default] IEEE 1149.1.
- **SWD** — ARM Single Wire Debug. In SWD mode, nTRST and TDI are unused, TMS becomes SWDIO and is bidirectional, and TDO becomes SWO and is unused.
- **cJTAG** — Compact JTAG IEEE 1149.7 in a 2-wire T4+ TAP.7 configuration. In cJTAG mode, TMS becomes TMSC and is bidirectional, TDI is unused, and TDO is unused. The probe uses either the OSCAN1 or the OSCAN2 scan format depending on target support.

This option is ignored when debugging BDM targets; use of BDM is determined by the connected adapter.

Note

Changing this option requires cycling the power on some targets. For more information, see "Changing the Target Communication Protocol" on page 323.

Enabling and Disabling DHCP

Green Hills Debug Probes support the Dynamic Host Configuration Protocol (DHCP), which allows you to use dynamically assigned IP addresses rather than assigning a permanent, static address for your probe. You must have a properly configured DHCP server on the same network as the probe for DHCP to function.

DHCP is enabled by default on new probes. If you have disabled DHCP on your probe and want to enable it again, open a serial terminal to the probe and enter the command:

```
set dhcp on
```

If your DHCP server supports hostname and lease requests, you can optionally specify your preferred hostname and lease setting to the probe:

```
set hostname myprobe
set dhcp_lease 5
```

The preceding example settings request a 5 day DHCP lease of an address assigned to the hostname **myprobe**. To request an infinite lease, set **dhcp_lease** to 0.

If a DHCP hostname is not specified, the default hostname request is **ghprobe** with the probe's serial number appended. For example, a probe with a serial number of 4000 has a default DHCP hostname of **ghprobe4000**. The default DHCP lease is an infinite lease.

The probe informs you that you must reboot the probe before your settings will take effect. To reboot the probe, either use the **reboot** command or power cycle the probe.

After the probe boots, it prints out the IP address that it received. You can use this IP address to telnet to the probe or connect with **mpserv**, **mpadmin**, etc.

The probe must acquire the DHCP IP address after the reboot. The probe cannot print the address immediately if there are delays on your network.

To disable DHCP, enter the command:

```
set dhcp off
```

Example 4.1. Enabling DHCP

```
arm7tdmi[h] % set dhcp on
NOTE: You have to reboot for this option to take effect.
arm7tdmi[h] % reboot
...
[*** BOOT MESSAGES ***]
...
Type 'info' to list probe information.
Type 'setup' to set the current configuration.
Type 'help' to list the online help.

DHCP IP address bound: 192.168.100.206
Netmask bound:    255.255.255.0
Gateway bound:    192.168.100.254
```

```
arm7tdmi[h] %
```

This probe is now ready to be accessed using the IP address 192.168.100.206.

Setting the Number of Receive Lanes

The SuperTrace Probe v3 supports high-speed serial trace communication over the Xilinx Aurora protocol. To configure the number of receive lanes used in the Aurora link from the target to the probe, use the command:

```
set hsst_rx_lanes n
```

where *n* is the number of transmit lanes provided by the target's Aurora block. This option is only applicable if you have a target and SuperTrace probe that support high-speed serial trace, and have the appropriate cable attached to your probe.

Setting the Netmask

To set the netmask for your Green Hills Probe or SuperTrace Probe Ethernet connection, use the command:

```
set netmask netmask
```

where *netmask* is the value of the netmask, specified in dotted quad notation. The default setting is appropriate in most cases. Ask your system administrator if you are unsure what this value should be. This setting does not take effect until the next reboot.

Example:

```
set netmask 255.255.255.0
```

This command sets the netmask to `255.255.255.0`.

Setting the Gateway

To set the gateway for your Green Hills Probe or SuperTrace Probe Ethernet connection, use the command:

```
set gateway gateway
```

where *gateway* is the value of the gateway, specified in dotted quad notation. This setting does not take effect until the next reboot.

Example:

```
set gateway 192.168.0.1
```

This command sets the gateway to `192.168.0.1`.

Setting the Target Adapter Type

Green Hills Debug Probes support a wide variety of target boards with different debug interfaces. To connect to your target, you must have a target adapter that meets the electrical specification of the particular debug interface on your target board. The **adapter** configuration option allows you to specify which target adapter is currently being used with the probe. The syntax for setting this option is:

```
set adapter adapter_type
```

where *adapter_type* represents the target adapter type (see “Green Hills Probe Target Adapter Types” on page 383 or “SuperTrace Probe Trace Pod Types” on page 385 for a list of appropriate values for your probe type). The default setting is `auto`, meaning that the probe detects what kind of adapter is attached.

If you are using TraceEverywhere cabling, `auto` is the only correct setting. Green Hills Probe v3 TraceEverywhere kits are supported with firmware 3.8 or newer. SuperTrace Probe TraceEverywhere kits are supported with firmware 5.0 or newer. If you are using legacy cabling, the probe can usually detect your adapter type when set to `auto`, but in some situations you may need to specify which target adapter you are using.

Setting the I/O Interface Voltage

Warning

You must set this value correctly for your target CPU type. Setting the value incorrectly can cause damage to the target system.

On a Green Hills Probe or SuperTrace Probe not connected to the Legacy Kit, first try the **dlh** command (see "Configuration Commands" on page 175) to automatically configure your logic levels.

On a SuperTrace Probe connected to the Legacy Kit (or if you need to enter a custom value), use **set logic_high** to specify the I/O interface voltage. For example:

```
set logic_high volts
```

where *volts* corresponds to your target system voltage (for most systems, this is 3.3 volts). The triggering level is 65% of the value of **logic_high**.

Note

Legacy kits can only accept the following four settings for **logic_high**: `1.5`, `1.8`, `2.5`, and `3.3` volts.

If you have a Green Hills Probe with a TraceEverywhere cable, there is an adapter that connects the cable to the probe. If this adapter has a serial number below `1001`, it is not recommended for use with targets that require a **logic_high** setting below 2.5 volts. If you have one of these cables and require this setting, contact Green Hills support.

On a TraceEverywhere Trace Pod, use the **trace_logic_high** option to specify the trace input voltage.

Format:

set trace_logic_high [logic_high | *voltage*]

- `logic_high` — Use the same I/O voltage as the `logic_high` setting. This is the default.
- *`voltage`* — Specifies an independent trace I/O logic level.

This option controls the voltage threshold for trace signals, conforming approximately to an LVCMOS standard at the given voltage (i.e., when set to 2.5, the target interface roughly conforms to a 2.5 volt LVCMOS standard). The actual voltage threshold used is further reduced internally when Trace Termination is enabled (see “Setting Trace Termination Control on TraceEverywhere Pods (SuperTrace Probe)” on page 91) to compensate for the dampened signals.

For example:

```
set trace_logic_high 1.2
```

Sets the trace I/O voltage reference corresponding to a 1.2 volt logic level (approximately 0.6 volts without termination, or 0.5 volts with termination).

Specifying Your Target

To detect and configure your probe for the processor cores on your target, run the **detect** command (see “Configuration Commands” on page 175 for more information).

If **detect** cannot detect the proper settings for your target, you must configure the target setting manually. To do so, enter the command:

set target *device_name ...*

where each *device_name* is one of the devices listed in “Supported Devices” on page 368. If your target has a BDM or SWD debug interface, specify one *device_name*. If your target has a JTAG debug interface, specify one *device_name* for each test access port (TAP) controller in your target's JTAG scan chain. List the devices in the same order as they appear in the JTAG scan chain, starting from the probe's TDI pin: list the device closest to the probe's TDI pin first, and the device closest to the probe's TDO pin last.

Specifying Multiple Cores and Multiple JTAG TAPs

The **detect** command can detect each TAP on a multiple-TAP target and configure your probe accordingly.

You can also manually specify multiple TAPs by using multiple *device_name* arguments for the **set target** command. For example, if there are two PPC750 processors on your target, use the following command:

```
set target ppc750 ppc750
```

If your target board uses one JTAG TAP controller for multiple cores, specify just one *device_name* argument for that TAP controller. For example:

```
set target ppc8572
```

For information about bypassing one or more of the devices on the JTAG scan chain, see the following section.

Note

Multi-core debugging is not currently supported for PowerPC 8xx/5xx and ColdFire targets.

Specifying Bypassed Devices

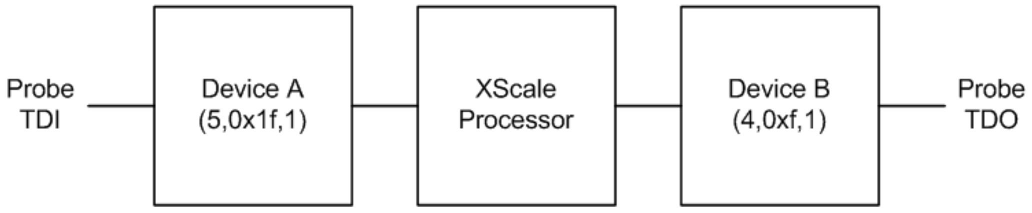
If there is a device on your JTAG scan chain that you do not want to or cannot debug (such as an FPGA or peripheral chip), bypass it by adding the following *device_name* to the **set target** command:

```
other(bypass_inst_length, bypass_inst, bypass_length)
```

where:

- *bypass_inst_length* — Specifies the length in bits of the device's JTAG bypass instruction.
- *bypass_inst* — Specifies the device's JTAG bypass instruction in hexadecimal. The instruction is truncated to the instruction bypass length, and is usually all `1`s.
- *bypass_length* — Specifies the length in bits of the device's JTAG scanchain when in bypass, which is usually 1 bit.

For example, consider a JTAG chain that includes three devices, arranged as follows:



This scan chain includes two peripheral devices, A (with a bypass instruction length of 5 bits and a bypass length of 1 bit) and B (with a bypass instruction length of 4 bits and a bypass length of 1 bit) with an XScale processor in between them. To specify this target, you would use the following command:

```
set target other(5,0x1f,1) xscale other(4,0xf,1)
```

Debug operations are not available for bypassed cores. When the probe scans the JTAG chain, these cores are placed in bypass and are not reported to MULTI. The only operations supported on these cores are the generic JTAG commands **ji**, **jd**, **jr**, and **vb**.

Specifying ARM Targets with an Embedded Trace Buffer

If your target has an embedded trace buffer (ETB) and is not a member of the Cortex family, use the following command to configure the target setting manually:

```
set target device_name armetb
```

where *device_name* is one of the ARM devices listed in “Supported Devices” on page 368.

Specifying CoreSight Targets

A CoreSight DAP appears as a single JTAG TAP or SWD device, but can provide access to multiple cores and other debug components. A CoreSight DAP and its member components are specified to the probe in the following manner:

```
csdap(component[*quantity] [component[*quantity]] ...)
```

For example, to specify a CoreSight DAP containing two Cortex-A15 cores, two Cortex-A7 cores, a CoreSight Trace Funnel, an ETB and a TPIU, you would use the target type:

```
csdap(a15*2 a7*2 cstf etb tpiu)
```

The following non-CPU CoreSight components are supported:

- `etb` — Embedded Trace Buffer
- `cti` — Cross-Trigger Interface
- `tpiu` — Trace Port Interface Unit
- `etf` — Embedded Trace Funnel
- `etr` — Embedded Trace Router*
- `cstf` — CoreSight Trace Funnel
- `stm` — System Trace Macrocel*
- `itm` — Instrumentation Trace Macrocel*

* Access to STM, ITM and ETR registers is provided. Collecting trace from an STM, ITM or ETR is not supported.

See "Supported Devices" on page 368 for a list of available CoreSight targets.

Note

The per-core options for specifying the AP Index and register base address must be correctly set. These include:

- Cortex-M Cores:
  - `ahb_index`
- ARMv7-A and ARMv7-R Cores:
  - `apb_index`
  - `memap_type`
  - `memap_index`
  - `cortex_addr`
  - `cti_addr`
  - `etm_addr`

  - `ptm_addr`
- Non-CPU cores:
  - `ap_index`
  - `address`

In most cases, the **detect** command can set them for you.

Specifying MIPS Targets with Multiple TCs and VPEs

If your target is a MIPS processor with the MT (multithreading) ASE (such as the 34K or 1004K) and has multiple thread contexts (TCs) and virtual processor environments (VPEs), you may need to specify your target manually. To do so, list all devices on the scan chain, using the following additional syntax:

- Specify additional VPEs as `mips_vpe`
- For each 34K processor, append `.n` to *device_name*, where *n* is the number of TCs.

For example, if you have a 34K with 2 TCs and 1 VPE (no additional VPEs) followed by another 34K with 3 TCs and 2 VPEs, use the following command to specify your target:

```
set target mips32_34kc.2 mips32_34kc.3 mips_vpe
```

Specifying PowerPC Targets with e6500 Cores

PowerPC e6500 cores are dual-threaded. Normally the probe lists all threads of all cores of a target in sequence: the probe's core `ID 0` is e6500 core 0 thread 0; `ID 1` is e6500 core 0 thread 1; `ID 2` is e6500 core 1 thread 0; and so on.

When debugging an operating system that does not use the secondary threads of each core, it may be helpful to have the probe hide the secondary threads. You can do this by appending `.singlethreaded` to the target setting. For example:

```
set target ppcT4240.singlethreaded
```

configures the probe to debug only thread 0 of each of the T4240's twelve cores. Only use the `.singlethreaded` suffix when the secondary threads are disabled.

Specifying Targets That Have An ICEPick

If you are debugging a Texas Instruments target that has an ICEPick, such as an OMAP3 or DaVinci board, specify it with the following argument:

```
icepick(id:device_name...)
```

where *id* is the position of the device in the scan chain behind the ICEPick (starting with `0)` and *device_name* is one of the devices listed in "Supported Devices" on page 368. For example, to specify a typical DaVinci board, use the following command:

```
set target icepick(0:arm926 1:armetb)
```

Any devices behind an ICEPick that you do not specify are bypassed by the probe; you do not need to use the `other` argument to bypass them. For example, to specify an OMAP3, use the following command:

```
set target icepick(3:omap3)
```

If you have a revision D ICEPick, use `icepick.d` instead of `icepick`. For example, the correct target setting for an OMAP4 is:

```
set target icepick.d(9:omap4)
```

Selecting the Current Core On Multiple-Core Targets

After you use the **detect** or **set target** command, the probe sets the currently selected core to the first core in the JTAG scan chain. The name of the selected core appears in the probe's prompt. Most probe commands are sent to this selected core. To change the selected core, use the following command:

```
t core
```

where *core* is the core's ID. To obtain a list of cores and their IDs, use:

```
tl
```

You can also use the **t** command to send commands to groups of cores, or send commands to a specific core without changing the selected core. For more information, see "Run Control Commands" on page 202.

Setting the Serial Communication Speed

To set the serial communication speed for a serial connection to a Green Hills Probe or SuperTrace Probe, use the command:

```
set baudrate baudrate
```

where *baudrate* is 2400, 4800, 9600, 19200, 38400, 57600, or 115200. This setting takes effect immediately.

If this command is entered from a serial console, you must immediately reconfigure your communications software to the new baud rate.

Enabling or Disabling the Status Checker

Green Hills Debug Probes feature a status checker that can be used to continually poll the target's status and report changes. To enable or disable the status checker, use the command:

```
set checker on|off
```

Note

If status checking is not enabled, you can use the **ti** command to request the current target status. MULTI is not dependent on the **checker** configuration option.

Setting the JTAG or SWD Clock Speed

You may want or need to adjust the JTAG or SWD clock speed to increase performance, or to resolve problems with your connection. Faster clock speeds result in higher performance. However, if the clock speed is too high, reliability can be compromised. In some cases, clock speeds might be rounded due to hardware requirements.

To set the clock speed, use the command:

```
set clock speed
```

where *speed* is the clock speed in Hz.

The supported values for *speed* vary depending on your probe model. As a result, some of the examples below may not be supported on your probe. To view a range of supported values, use the command:

```
set clock ?
```

Example 4.2. Using set clock

To set the JTAG or SWD clock speed to 10MHz, use the following command:

```
set clock 10000000
```

You can also specify clock speed with units of kilohertz (kHz) or megahertz (MHz). The unit specifier is case-sensitive. For example, the following commands both set the clock speed to 10MHz:

```
set clock 10 MHz
set clock 10000 kHz
```

Enabling or Disabling Target Power Detection

The status checker can be used to determine whether or not the target has power. To enable or disable this feature, use the command:

```
set power_detect on|off
```

This option is on by default. If you disable it by setting the option or loading a configuration file, you must always tri-state the probe's outputs before turning on your target (see “The User Button” on page 5). Re-enable the probe's outputs only after your target is in a state where it can be debugged.

Configuring Probe Run Mode Behavior on Host Disconnect

When using Probe Run Mode, you may need to specify whether or not the Run Mode Proxy continues running when the connection that started the Proxy exits. If you disconnect the Stop-Mode connection, it usually means you are done using the Run-Mode connection as well. However, if you do not want to continue using the Stop-Mode connection, set the option to on to make the Run-Mode connection persist when the Stop-Mode connection exits. You may also want to enable this option if you start the Probe Run Mode Proxy from a telnet session, so that the connection persists after you close the telnet session. To change this behavior, use the command:

```
set prm_ignore_disconnect on|off
```

where:

- **on** — The Probe Run Mode Proxy continues running even when the connection that started the Proxy disconnects.
- **off** — [default] When the Probe Run Mode Proxy is running and the connection that started the Proxy disconnects, the Proxy exits, interrupting anything that uses it.

Configuring Probe Run Mode Behavior on Target Reset

When using Probe Run Mode, you may want to control whether or not the Run Mode Proxy continues running when a target reset is initiated. To change this behavior, use the command:

```
set prm_ignore_reset on|off
```

where:

- **on** — The Probe Run Mode Proxy continues running even if you reset your target using the probe. Use this option if you are debugging out of ROM and want the Proxy to reconnect to the kernel when it boots.
- **off** — [default] When the Probe Run Mode Proxy is running and the probe resets the target, the Proxy exits, interrupting anything using the proxy. Usually, during development and when downloading an INTEGRITY kernel, the target reset at the beginning of your setup script ensures that the Proxy stops before the new kernel downloads and runs. If the target is reset by a mechanism other than the probe, the Proxy continues running.

Enabling or Disabling the Serial Terminal

Turns the probe's serial terminal console interface on or off. When turned off, the probe serial port is made available for use as a remote serial port with the **pterminal** command or other utility. When turned on (the default), the probe serial port presents a command line console interface which will accept debug commands in the same fashion as mpserv, gpadmin, or telnet to TCP port 23.

To turn the probe's serial terminal console interface on or off use the command:

```
set serial_terminal on|off
```

Holding down the USER button for 5 seconds when this option is off will temporarily re-enable the console terminal interface. The console terminal behavior will revert to the saved value the next time the probe boots.

serial_terminal must be off when using the probe as a remote serial port. See the **pterminal** command in “System Commands” on page 187 for more information.

Setting the User String

The *user string* is used to name and identify a Green Hills Probe or SuperTrace Probe, and appears on the startup screen. To set the user string, enter the command:

```
set user_string string
```

Disabling Multiple Binary Connections

The option **single_mpserv_only** prevents multiple binary **mpserv** connections to the probe. After a USB connection has been started, it does not terminate. To subsequently connect using Ethernet, reboot the probe or disable this option.

If there is already more than one binary connection, enabling this option does not close any current connections, but prevents further connections from being made.

When set to **off**, the probe allows multiple **mpserv** connections.

When set to **on**, the probe allows only one **mpserv** connection.

This option defaults to **off**. To change this setting, use the following command:

set single_mpserv_only on | off

Setting the CPU Reset Pulse Length

The reset pulse length might need to be adjusted from the default if your target requires a longer result pulse to reset the target. To specify the reset pulse length, enter the command:

```
set rst_pulse milliseconds
```

where *milliseconds* specifies the minimum time in milliseconds that the nRST line will be held low during a reset.

Note

For MIPS targets, the **rst_handshake_timeout** option may also need to be adjusted. See “MIPS” on page 119 for more information.

Setting the JTAG TAP Reset Pulse Length

The JTAG TAP reset pulse length may need to be adjusted from the default if your target needs a longer result pulse to reset the JTAG TAP controller. To specify the JTAG TAP reset pulse length, use the command:

```
set jrst_pulse milliseconds
```

where *milliseconds* specifies how long the nTRST line will be held low during a JTAG TAP reset.

Setting the Reset Settle Length

The reset settle length may need to be adjusted if your target needs more time to stabilize before issuing commands. To specify the reset settle length, use the command:

```
set rst_settle milliseconds
```

where *milliseconds* specifies how long the probe waits after an nRST pulse is issued before scanning any commands into the target.

You may need the probe to wait for an additional amount of time after waiting and detecting that the target has powered on. To specify the power on settle length, use the command:

```
set power_on_settle milliseconds
```

Setting the JTAG TAP Reset Settle Length

The JTAG TAP reset settle length may need to be adjusted if your target needs more time to stabilize before issuing commands. To specify the JTAG TAP reset settle length, use the command:

```
set jrst_settle milliseconds
```

where *milliseconds* specifies how long the probe waits after an nTRST pulse is issued before scanning any commands into the target.

Setting Reset Pin and JTAG TAP Interaction

Ideally, asserting the reset pin (nRST) on your target does not affect the JTAG TAP, so you can communicate with the TAP while asserting the reset pin. However, the reset pin and TAP do not operate independently on all targets. For more information, see “Diagnosing Reset Line Problems” on page 319.

If asserting the reset pin on your target resets or freezes the TAP, set the following option so that the probe is aware of your target's behavior:

set target_reset_pin independent | freezes_tap | resets_tap

- **independent** — [default] Asserting the reset pin does not affect the TAP at all.
- **freezes_tap** — Asserting the reset pin freezes the TAP. When the reset pin is deasserted, the TAP resumes operation. This occurs in targets where asserting the reset pin freezes the clock signal to the TAP, but the TAP reset line is not affected.
- **resets_tap** — Asserting the reset pin resets the TAP. This typically occurs when the reset line and the TAP reset line are wired together on the board.

Note

This option tells the probe what to expect from your hardware, but does not change the way your hardware operates.

Setting the Drive Strength of TraceEverywhere Pods (SuperTrace Probe)

To control the JTAG signal drive strength on TraceEverywhere (TE) pods, use the command:

```
set jtag_drive [auto|slow|medium|fast]
```

where:

- **auto** — selects the drive state based on the **logic_high** setting (`fast` for lower voltages, `medium` for higher voltages).
- **slow** — the output buffer is driven through an effective resistance of 5 ohms.
- **medium** — the output buffer is driven through an effective resistance of 28 ohms.
- **fast** — the output buffer is driven through an effective resistance of 110 ohms.

Setting the Byte Order (Endianness)

To specify the byte order of the target, use the command:

```
set endianness mode
```

where *mode* is either `big` for big endian or `little` for little endian.

Note

You can use the **detect** command or the **de** command to automatically detect the target byte order. See the table in "Green Hills Debug Probe Commands" on page 174 for descriptions of these commands.

Setting the Trace Clock Source (SuperTrace Probe)

To specify the clock-to-data timing used for capturing trace data, use the command:

set trace_clock_source [normal | bulk_delay | pll]

- **normal** — [default] Routes the trace clock from the target directly to the buffers that capture the trace data. In this configuration, the SuperTrace Probe's timing parameters are identical to those listed on its datasheet.

- **bulk_delay** — Delays the trace clock by about 20 ns. Usually, this setting is required only in cases where phase adjustment is necessary, but the trace clock is not compatible with phase-locked loop (PLL) clock management (for example, trace clocks that change at run time).
- **pll** — Uses a PLL for fine grained control over the clock-to-data timing used for capturing trace data. The PLL has two major restrictions:
    1.
        - On SuperTrace Probe v1 — Only ARM pods have a **pll** option, and it can only be used with trace clock frequencies above 50 MHz.
        - On SuperTrace Probe v3 with the Legacy Pod — All architectures have a **pll** option. On PPC405 and PPC440, it can only be used with trace clock frequencies above 50 MHz. On all other architectures, it can only be used with trace clock frequencies above 25 MHz.
        - On SuperTrace Probe v3 with the TraceEverywhere Pod — All architectures have a **pll** option, and it can only be used with trace clock frequencies above 5 MHz.

       (Remember to take into account any double-data-rate or clock division features of your trace port.)
    2. After the PLL is locked, its tolerance for clock period jitter is +/-1 ns. For these reasons, the PLL setting is unsuitable for trace ports that:
        - have relatively slow clocks
        - gate the trace clock
        - vary the trace clock frequency at run time

  If you use this setting, use the **trace_clock_phase** setting to adjust the clock-to-data timing that the SuperTrace Probe uses for capturing data from the trace port.

Setting the Trace Clock Phase (SuperTrace Probe)

This setting has no effect unless **trace_clock_source** is set to **pll**. To adjust the clock-to-data timing requirements of the SuperTrace Probe, use the command:

set trace_clock_phase *phase*

- ***phase*** — The phase offset, in 256ths of a trace clock period. The valid range for the phase is `-127` to `+127`. If the phase is set to 0, the timing requirements of the SuperTrace Probe match those listed in the probe's datasheet. If the phase is not `0`, the setup and hold times listed in the datasheet still apply, but the point around which they center is no longer the edge of the trace clock as seen at the clock input.

For example, if you set *phase* to `64`, the adjusted trace clock used for data capture is 1/4 of one trace clock period ahead of the trace clock at the clock input. If the trace clock were running at 100 MHz, the period would be 10 ns, meaning that the SuperTrace Probe would capture data from the trace port 2.5 ns after the corresponding edge (rising or falling) is seen on the clock signal at the probe's trace clock input.

Setting the Trace Clock Delay (SuperTrace Probe)

This setting has no effect unless **trace_clock_source** is set to **pll**, and the TraceEverywhere pod is connected. For other restrictions, see “Setting the Trace Clock Source (SuperTrace Probe)” on page 88.

To adjust the clock-to-data timing requirements of the SuperTrace Probe, use the command:

set trace_clock_delay $*delay_steps*

The allowed adjustment range is `-255` to `+255`, with the default being `0`; however, the usable range may also be limited by the frequency of the incoming trace clock. When this option is `0`, the timing requirements of the SuperTrace Probe match those listed in the SuperTrace Probe's datasheet.

When this option is not `0`, the setup and hold times listed in the SuperTrace Probe's datasheet still apply, but the point around which they center is no longer the edge of the trace clock as seen at the SuperTrace Probe's clock input. Instead it is the edge plus or minus some number of delay steps. The units of this parameter vary based on the frequency of the incoming trace clock, but are constrained to a minimum of 10 ps, a maximum of 40 ps, and a typical value of 26 ps per delay step.

Setting Trace Termination Control on TraceEverywhere Pods (SuperTrace Probe)

To control the trace termination circuitry on TraceEverywhere (TE) pods, enter the following command:

```
set trace_term_enable [on|off]
```

where:

- **on** — a 270 ohm pull-down is enabled on each of the trace signals into the pod.
- **off** — the pull-down is not enabled.

Setting the User Button Behavior

To configure what the probe does when you press the **User** button on its front panel, enter the following command:

set user_button tri-state | halt | publish

- **tri-state** — [default] Pressing the **User** button tri-states the probe, meaning that all target connector pins are put in a high impedance state. In this mode, the LEDs on the right side of the probe blink. When the probe is tri-stated, you can safely unplug your target board from your probe, even though the probe is powered on. To leave this mode, press the **User** button a second time, or issue the `jp on` command.
- **halt** — Pressing the **User** button halts all processors on the target.
- **publish** — Reserved for future use.

Specifying Download Verification

This setting specifies whether a downloaded image is read back from the target to compare to the downloaded image, and if so, whether the full image is read back and compared, or a small subset of the image is read back and compared.

If the data read back does not match the data downloaded, an error indicating the location and value of the first mismatch is output.

To configure what is read back from the target and compared to the download for verification, use the following command:

set verify_download off | sparse | full

- `off` — No image will be downloaded and compared. Set this option to `off` for maximum download speed.
- `sparse` — A small subset of the downloaded image is read back from the target and compared to the downloaded image.
- `full` — The full image is read back and compared.

Target-Specific Configuration Options

The options listed in this section work only for certain targets. These commands are listed below in tables according to the architecture family.

ARC

Unless otherwise specified, the following configuration options are available for the targets specified by the ARC device names (see "Specifying Your Target" on page 76 for information about specifying the target device name with the **set target** command).

agent For ARC processors, this setting specifies a location to write a `NOP` for use in pipe cleaning. The probe restores the data after it has finished using it. To change the **agent** setting, enter the command: **set agent** *address* Where *address* is an address that is not the target of a branch in your program. We recommend that you use an address outside the program altogether, in the stack, or in the heap of the program.

step_ints

Sets single-stepping behavior when interrupts are enabled.

When this option is turned off, interrupts are disabled on the target during a single-step. When this option is turned on, interrupts are left alone so a single-step can go to an interrupt handler instead of the next instruction.

This option defaults to `on`.

To turn **step_ints** on or off, enter the command:

set step_ints on | off

check_cache

Applies to ARCtangent-A5 targets only.

Controls whether or not the probe checks the validity of each cache entry before flushing it. Some revisions of the ARC A5 core have a cache bug that causes zeros to be written to memory when flushing an invalid cache entry with the dirty bit set. Set this option to `on` if your core has this bug.

To turn this setting on or off, enter the command:

set check_cache on | off

- `on` — [default] the probe checks the validity of each cache entry before flushing it.
- `off` — the probe does not check the validity of each cache entry before flushing it. If your target has this cache bug, the problem occurs most noticeably when setting breakpoints. For newer ARC A5 revisions that do not have this bug, you can safely set this option to `off`. You may see a minor performance increase in some memory operations.

ARM

Unless otherwise specified, the following configuration options are available for the targets specified by the ARM device names, except for XScale devices (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command). For XScale, see “XScale” on page 159.

abort_check Applies to ARM7 and ARM9 cores only. Enables or disables abort checking. Unless your probe is connected to a target that incorrectly reports data aborts in debug mode, do not explicitly set this option. To turn abort checking on or off, enter the command: **set abort_check** on | off • `on` — [default] Any data abort generated during a probe memory access causes that memory access to fail. • `off` — The probe ignores data aborts generated during probe memory accesses. This option may improve compatibility with targets whose external peripherals incorrectly report data aborts.
address Applies to CoreSight components other than CPUs. The base address of the registers for this debug component. To set this option, use the following syntax: **set address** [*address*] where *address* is within the supported range `0x0` – `0xfffff000`, and must be aligned to `0x1000`.
ahb_index Applies to ARMv7-M and ARMv6-M targets. The index of the AHB-AP that provides access to the core's debug registers. To set this option, use the following syntax: **set ahb_index** *value* where *value* is within the supported range of `0x00` – `0xff`.

ap_index

Applies to CoreSight components other than CPUs.

The index of the MEM-AP that provides access to the component's registers.

To set this option, use the following syntax:

set ap_index *value*

where *value* is within the supported range of `0x00` – `0xff`.

apb_index

Applies to ARMv7-A and ARMv7-R targets.

The index of the APB-AP that provides access to the core's debug registers.

To set this option, use the following syntax:

set apb_index *value*

where *value* is within the supported range of `0x00` – `0xff`.

arm720_r4_cp_access

Revision 4 of the ARM720 core changed the way that the probe accesses coprocessor 15 registers. When this option is **on**, the probe accesses coprocessor 15 registers in a manner that is acceptable for revision 4.

When this option is set to **off**, the probe accesses coprocessor 15 registers in a manner that is acceptable for revision 3 and earlier. You must turn this option **on** if you are using the revision 4 ARM720 core.

set arm_720_r4_cp_access on | off

associated_core

Associates the ARM embedded trace buffer (ETB) with a specific core. If your target has multiple cores, this option is necessary because the probe cannot detect which core an ETB is associated with.

To set the associated core, enter the command:

set associated_core *core*

be_mode

Controls the big endian mode used for cores with two different big endian modes. This option is only visible when big endian mode is enabled. To set the mode, enter the command `set endianness big`, and then enter:

set be_mode be32 | be8

- `be32` — The probe uses BE-32 mode. This is the correct setting for ARMv7-R processors that have the `SCR[IE]` set, even though such processors technically run in `BE-8` mode.
- `be8` — The probe uses BE-8 mode.

catch_*exception*

Determines whether or not to halt the core before executing the first instruction of the exception handler for *exception*. The following values for *exception* apply to ARM9, ARM11, and Cortex cores, except the Cortex-M family:

- `abort` — Data Abort
- `fiq` — FIQ
- `irq` — IRQ
- `prefetch` — Prefetch Abort
- `reset` — Reset
- `undef` — Undefined Instruction

The following value for *exception* applies to ARM9, ARM11, and XScale:

- `swi` — Software Interrupt (SWI)

The following value for *exception* applies to Cortex-A and Cortex-R:

- `svc` — Secure Monitor Call (SMC) and Supervisor Call (SVC)

The following values for *exception* apply to Cortex-M3 and Cortex-M4 processors only:

- `buserr` — Normal Bus Error
- `chkerr` — Usage Fault Enabled Checking Error
- `harderr` — Hard Fault Error (also applies to Cortex-M0)
- `interr` — Interrupt Error
- `mmerr` — Memory Management Fault
- `nocperr` — Usage Fault Access to Coprocessor
- `reset` — Reset (also applies to Cortex-M0)
- `staterr` — Usage Fault State

The default setting for all **catch_*exception*** options is `off`.

To turn abort checking on or off, enter the command:

set catch_*exception* on | off

- `on` — Sets up the debug logic so the core halts before executing the first instruction of the exception handler.
- `off` — The probe does not enable the trap logic, and exceptions execute normally.

For example:

set catch_abort on

catch_por

Applies to ARM9 and ARM11 cores only. Requires that you set the **power_detect** option to `on`.

Determines whether or not to stop the target as it powers on by asserting the reset signal to the target when the power is off. When the probe senses that power to the target is on, it programs the target debug port to catch the reset exception and then releases the reset signal. The default is `off`.

To enable or disable this option, use the following syntax:

set catch_por on | off

- `on` — Sets up debug logic to halt the core when the target first powers on, before executing the first instruction of the reset exception vector. It is similar to the **catch_reset** option in behavior except that it only affects the reset exception that occurs as the target is powered on. The probe extends the time reset is asserted to the target at power-on by some amount of time, generally a few milliseconds.
- `off` — Do not attempt to stop the target as it powers on.

cortex_addr

Applies to ARMv7-A and ARMv7-R only.

Specifies the address of the debug registers on the APB.

To specify the address, use the following syntax:

set cortex_addr *address*

cti_addr

Specifies the address of the Cross Trigger Interface (CTI) registers on the APB for the CTI associated with the PTM or ETM of this core.

To specify the address, use the following syntax:

set cti_addr *address*

dbcom_channel The probe can export the ARM debug communications registers for special debug uses. To change this option, enter the following command: **set dbcom_channel** none | serial | syscall When the `none` argument is used, the special debug communications registers export is not made. When the `syscall` argument is used, the debug communications registers are exported as a special host I/O port for MULTI. This allows all of MULTI's host I/O port to be used without ever halting the core, and interrupts will continue to be serviced. **dbcom_handler** requires a special handler that is configured by the options **dbcom_handler_addr** and **dbcom_handler_size**. When the `serial` argument is used, the debug communications registers are exported as a virtual serial port. The least significant byte of the 32-bit data register is used.
dbcom_handler_addr The address of the handler used to enable host I/O over the ARM debug communications registers. This value must be in writable memory within 32 MB of the `__dotsyscall` section. The configuration option **dbcom_channel** must be set to **syscall** to use this configuration option. To set this option, use the following syntax: **set dbcom_handler_addr *address***
dbcom_handler_size The size of the handler used to enable host I/O over the ARM debug communications registers. The configuration option **dbcom_channel** must be set to **syscall** for this configuration option to be used, and should be a size of at least `0x400`. To set this option, use the following syntax: **set dbcom_handler_size *size***

dbgrq_halt

Specifies whether or not the probe uses the ARM `DBGRQ` signal to halt the core.

To set this option, use the following syntax:

set dbgrq_halt on | off

- `on` — The probe halts ARM7 and ARM9 cores by scanning the ICEBreaker/EmbeddedICE control register with the bit for `DBGRQ` set high. This is the ARM-recommended way to halt ARM7 and ARM9 cores, but does not work reliably for some ARM7 implementations.
- `off` — [default] The probe halts ARM7 and ARM9 cores by setting a hardware execute breakpoint with a mask such that the next instruction fetch halts the core. This method works for all ARM7 and ARM9 cores that have been tested with the probe.

If you do not know which setting to use, set this option to `off` for ARM7TDMI and ARM7TDMI-S cores, and `on` for all other ARM7 and ARM9 cores.

disable_swbp

Disables software breakpoints.

Some chips use the same resources for software and hardware breakpoints, which limits the available number of each type of breakpoint. By default, Green Hills Debug Probes allocate these resources to be biased toward software breakpoints. In some cases the hardware breakpoints might not be available at all. Setting this option to `on` allows you to use the maximum number of hardware breakpoints available on your hardware.

This option defaults to `off`, which means that software breakpoints are enabled. To change this setting, enter the following command:

set disable_swbp on | off

disable_watchdog

This option applies to Spansion FCR4 and FR5 targets

Specifies whether or not the probe disables the watchdog timer on the target as part of the reset sequence.

To set this option, use the following syntax:

set disable_watchdog on | off

- `on` — [default] The probe disables the watchdog timer on the target as part of the target reset sequence.
- `off` — The probe does not disable the watchdog timer.

es2_boot_workaround

Applies to OMAP3 only.

Configures the probe to work around an errata in OMAP3 revision ES2.0 and earlier that can cause a processor to become stuck in secure monitor mode at boot.

To set this option, use the following syntax:

set es2_boot_workaround on | off

- `on` — [default] The probe works around the errata by detecting when the target has been reset and clearing the secure scratchpad RAM before halting.
- `off` — The probe does not use the workaround.

etm_addr

Applies to ARMv7-A and ARMv7-R only.

Specifies the address of the ETM registers on the APB.

To specify the address, use the following syntax:

set etm_addr *address*

etm_arch_version

This option is only available in firmware 3.2.4 or newer. If this option is not available, use **etm_version** instead.

Some ARM cores (such as the ARM9E-S, ARM9EJ-S, and their variants) can use either the ETM version 1.x or ETM version 3.x trace protocol. This option selects which protocol to use with your cores. Use `none` if your target has no trace hardware.

To set this option, use the following syntax:

set etm_arch_version *version_num*

where *version_num* is one of the following `ETM` version numbers: `none`, `1.0`, `1.1`, `1.2`, `1.3`, `2.0`, `3.0`, `3.1`, `3.2`, `3.3`, or `3.4`. The default value is `1.0`.

etm_version

This option is only available in firmware 3.2.4 or older. If this option is not available, use **etm_arch_version** instead.

When set greater than 0, `ETM` register access will be allowed, and `ETM` trace support for the given version of `ETM` will be enabled. Note that either a SuperTrace probe or an embedded trace buffer is required to collect `ETM` trace data.

To set this option, use the following syntax:

set etm_version *version_num*

where *version_num* is the `ETM` version number. The default value is 1.

fast_dl

Enables or disables fast downloading.

When **fast_dl** is turned on, the Probe assumes that the target can complete memory writes as fast as the Probe can issue them. Turning this setting on increases memory write speed by up to 300% in some cases. However, it can cause incorrect operation of targets that have slow CPUs or slow memory.

This option defaults to `off`.

To turn fast downloading on or off, enter the command:

set fast_dl on | off

force_dbgack

Set this option to **on** to force `DBGACK` high while halted in debug mode. `DBGACK` is normally asserted while in debug, but some operations can cause `DBGACK` to go low if this option is not on.

To enable or disable this option, use the following syntax:

set force_dbgack on | off

halt_settle

Sets the time interval, in milliseconds, that the probe waits after the target has halted and before it reads registers.

set halt_settle *time*

where *time* is the time interval, in milliseconds.

For example, to set the time to one second, enter the following command:

```
set halt_settle 1000
```

handler_base

For ARM9 cores only (not ARM9E).

Specifies the location of memory that the probe can use as scratch space for accessing the data cache. The specified area must have 64 bytes of valid writable memory.

This option only applies when the data cache is enabled.

To set this address, enter the following command:

set handler_base *address*

The default setting is `0x20`.

handler_base_physical For ARM9 cores that use the cache debug handler. This option specifies whether the address given in `handler_base` is physical or virtual. When this option is set to `on`, the address is treated as physical. When **handler_base_physical** is set to `off`, the address specified in `handler_base` is treated as a virtual address. To enable or disable this option, use the following syntax: **set handler_base_physical** on | off **set handler_base_physical** should be `off` unless you are using an operating system that does not directly map the memory region that contains the `handler_base` address. You do not have to enable this option when debugging INTEGRITY applications.
has_etm This option specifies whether the core has an Embedded Trace Macrocell (ETM) to enable access to the ETM registers. Set this option to `on` if the core has an ETM. If no ETM is present, set this option to `off`. To set this option, use the following syntax: **set has_etm** on | off
has_tpiu This option specifies whether the core has a Trace Port Interface Unit (TPIU) to enable access to the TPIU registers. Set this option to `on` if the core has a TPIU. If no TPIU is present, set this option to `off`. To set this option, use the following syntax: **set has_tpiu** on | off
ignore_csyspwrupack Applies to CoreSight DAP. When first connecting to a CoreSight system, the probe sets `CSYSPWRUPREQ` and `CDBGPWRUPREQ` in the Debug Port Control and Status Register, and waits until the target sets the `CSYSPWRUPACK` and `CDBGPWRUPACK` bits. Some systems do not properly respond to `CSYSPWRUPREQ` and in turn never set `CSYSPWRUPACK`. Turn this option `on` to debug such systems. To enable or disable this option, use the following syntax: **set ignore_csyspwrupack** on | off

inst_endianness Applies to Cortex-M only. Specifies whether Big Endian Cortex-M cores perform instruction fetches in Big Endian or Little Endian. Set this option to match the behavior of your core. To set this option, use the following syntax: **set inst_endianness** big | little • `big` — Use this setting if instruction fetches are Big Endian. • `little` — Use this setting if instruction fetches are Little Endian.
l2_cache_address Applies to ARM11 and Cortex-A9 only. Specifies the base address for the L2 cache control registers. If your target has an L2 cache, set this option and the **l2_cache_present** option. To specify the address, use the following syntax: **set l2_cache_address *address***
l2_cache_present Applies to ARM11 and Cortex-A9 only. Specifies whether or not your target has an L2 cache. If your target has an L2 cache, set this option to `on` and then set the **l2_cache_address** option. To set this option, use the following syntax: **set l2_cache_address** on | off
memap_index The index of the AHB-AP or AXI-AP that provides access to the core's memory bus. To set this option, use the following syntax: **set memap_index** [*value*] where *value* is within the supported range of `0x00` – `0xff`.

memap_type Applicable to ARMv7-A and ARMv7-R targets. Informs the probe whether a MEM-AP is present and provides access to the core's memory bus, and what sort of MEM-AP it is. To set this option, use the following syntax: **set memap_type** none | axi | ahb • `none` — There is no MEM-AP. • `axi` — There is an AXI-AP. • `ahb` — There is an AHB-AP.
monitor_mode This option controls the probe's support for ARM Real Monitor. Real Monitor is a monitor program that when integrated into the target application, allows the target to be debugged without halting the CPU. This allows critical interrupts to be handled even when the foreground application is stopped in the debugger. To enable or disable this option, use the following syntax: **set monitor_mode** on | probe | off • `on` — Sets several debug registers to the appropriate value for RMserv, and disables write access to the registers. This setting is only intended for internal use by RMserv. • `probe` — Causes the probe to enter Real Monitor mode. When the probe is in Real Monitor mode, the target is never halted. Debug commands are issued to Real Monitor running on the target. The probe's amber halt light will be illuminated when Real Monitor is in the `stopped` state. To enter Real Monitor mode, a Real Monitor enabled image must be loaded and running on the target. If the probe is initially unable to communicate with Real Monitor, **monitor_mode** will remain `off` despite attempts to set it to `probe`. The probe automatically sets **monitor_mode** to `off` (the default value) if the target is reset or loses power.

new_946_cache_mgmt

Applies to ARM 946 processors only

Specifies whether or not the probe uses an improved technique for managing instruction and data cache coherency. This technique results in less probe intrusion and causes cache visualization commands (such as `clf` and `clr`) to show the actual values in the data cache. To use this option, enter the following command:

set new_946_cache_management on | off

- `on` — Use the improved cache management technique. If using this setting causes incorrect operation, contact Green Hills support.
- `off` — Use an older, more intrusive cache management technique. Use this setting only if the `on` setting increases the frequency with which you must explicitly invalidate the instruction or data cache to maintain coherency.

num_dwt

Applies to the Cortex-M family of processors only.

Specifies the maximum number of *DWT hardware breakpoints* the probe can set, where a DWT hardware breakpoint is used for a hardware data breakpoint, hardware execute breakpoint with a mask, or hardware execute breakpoint set at `0x20000000` or higher. Lower settings for this option provide more resources for trace tools; it must be set to a value less than the maximum, or trace triggers will not work.

To set this option, use the following syntax:

set num_dwt *num*

Changing this option while mpserv is connected or while breakpoints are set will result in unpredictable behavior.

omap5_variant

Applies to Cortex OMAP5 only.

Use this option to tell the probe what sort of OMAP5 the target is.

To set this option, use the following syntax:

set omap5_variant auto | OMAP543x_ES1 | OMAP543x_ES2 | OMAP57xx

- `auto` — The probe will try to automatically determine the OMAP5 variant each time this information is needed.
- `OMAP543x_ES1` — Use this option if your target is an OMAP543x with silicon revision ES1.
- `OMAP543x_ES2` — Use this option if your target is an OMAP543x with silicon revision ES2.
- `OMAP57xx` — Use this option if your target is an OMAP57xx or DRA7x.

override_mmu Set this option to bypass the permissions set in the MMU, and access all memory with administrator privileges while debugging. **override_mmu** enables reading and writing protected memory, and setting software breakpoints in protected memory. To enable or disable this option, use the following syntax: **set override_mmu** on | off
override_mpu Set this option to bypass the permissions set in the MPU, and access all memory with administrator privileges while debugging. **override_mpu** enables reading and writing protected memory, and setting software breakpoints in protected memory. To enable or disable this option, use the following syntax: **set override_mpu** on | off
ptm_addr Applies to ARMv7-A processors with a PTM. Specifies the address of the PTM registers on the APB. To specify the address, use the following syntax: **set ptm_addr *address***
reset_detection Applies to Cortex-A9, A5 and A15. The debug registers of most SoCs are always available, and the reset and power state of a core can be determined by reading DBGPRSR. Some SoCs make their debug registers inaccessible when a core is held in reset or in a low power state. This option allows you to select an SoC-specific method to determine a core's power and reset state. To change this setting, enter the command: **set reset_detection** standard | *soc-type* • `imx6` — Determine if the core is held in reset by reading `SRC_SCR` on a Freescale iMX.6. • `vybrid` — Determine if the core is held in reset by reading `CCOWR` of a Freescale Vybrid. • `rcar` — Determine if the core is held in reset by reading `CA15RESCNT` or `CA7RESCNT` on an Renesas R-Car H2 or V2H. • `standard` — Determine if the core is held in reset by reading `DBGPRSR`.

reset_fixup

For ARM7 cores only.

Set this option to use an alternate reset sequence, allowing you to debug code immediately following a reset on targets that do not have an external reset signal or cannot be stopped at the reset vector before executing initial boot instructions. For more information, see “Setting Reset Pin and JTAG TAP Interaction” on page 87.

To change this setting, enter the command:

set reset_fixup on | off

- `on` — When the probe receives a **tr** or **tr d** command, it does the following:
    1. Halts the processor.
    2. Sets the PC to `0x0`.
    3. Sets the CPSR to `0xd3`.
- `off` — [default] The probe uses the standard ARM7 reset sequence.

rm_off_on_panic

Real Monitor enters a panic state if an error occurs, and sends a panic message to the probe. A common cause for entering the panic state is single-stepping or setting a breakpoint in code that is reached from an interrupt handler.

To enable or disable this option, use the following syntax:

set rm_off_on_panic on | off

- `on` — The probe exits Real Monitor mode in response to a panic message.
- `off` — The probe remains in Real Monitor mode after receiving a panic message.

rm_use_mem_descriptor

Real Monitor provides a memory access descriptor table in its configuration block. The table contains a list of memory regions, and whether read or write access is permitted, breakpoints are allowed, or cache synchronization is needed for each region.

To enable or disable this option, use the following syntax:

set_rm_use_mem_descriptor on | off

- `on` — The probe checks the memory descriptor table before memory access is allowed or breakpoints are set.
- `off` — [default] The probe ignores the memory access descriptor table.

rst_dpll3

Applies to Cortex-A8 OMAP3 processors only.

The probe performs a soft reset on an OMAP3 by writing to `PRT_RSTCTRL`. This setting controls whether the `RST_DPLL3` bit is set in that write.

To set this option, use the following syntax:

set rst_dpll3 on | off

- `on` — During soft reset, the probe sets `PRT_RSTCTRL.RST_GS` and `PRT_RSTCTRL.RST_DPLL3`.
- `off` — During soft reset, the probe sets only `PRT_RSTCTRL.RST_GS`.

rst_type

The reset method for ARM7TDMI, ARM7TDMI-S and ARM720 targets. Certain ARM targets cannot be stopped at the reset vector. Different methods can be used to debug startup code.

To set this option, use the following syntax:

set rst_type hard | simulate | precise

- `hard` performs a normal hardware reset, but might not halt the target at the reset vector.
- `simulate` emulates a reset, but will not actually pulse the reset line.
- `precise` performs a hard reset with a precise halt at the reset vector for those targets that can be precisely halted at the reset vector.

rsttime1

Applies to Cortex OMAP5 only.

The OMAP5 family of TI processors may require a specific value to be written to `PRM_RSTTIME.RSTTIME1` prior to being reset. This option controls whether the probe writes `PRM_RSTTIME.RSTTIME1` prior to resetting an OMAP5.

To set this option, use the following syntax:

set rsttime1 *value*

- If *value* is non-zero, this value is written to `PRM_RSTTIME.RSTTIME1` before resetting an OMAP5.
- If *value* is zero, `PRM_RSTTIME` is not written all.

rtck_use_timeout

This option is used only when **use_rtck** is not set to `off`.

The clock to the JTAG logic (not the TCK clock going to the target, which is only active when JTAG commands are issued to the target) must be active to prevent the whole device from becoming slow. If this clock is stopped, it might appear that the probe had completely frozen. However, the implementation of RTCK-based adaptive clocking gates this clock to the RTCK signal coming back from the target. If the RTCK signal from the target stops for a long period of time or is not wired correctly, the probe can freeze.

By default, this option is enabled, and adds a timeout so that the JTAG logic gets a clock even if the RTCK line is completely stopped. This prevents the probe from freezing if the target's RTCK signal does not work as expected, but it also enforces a minimum TCK speed when using RTCK-based adaptive clocking of about 5 to 10 kHz. If the target is running at a speed below 20 kHz or goes to sleep, the probe's RTCK timeout will cause it to not strictly follow the RTCK clocking discipline, and it might lose debug control of the target.

To enable or disable this option, use the following syntax:

set rtck_use_timeout on | off

Using the `off` arguments to disable the RTCK timeout causes the probe to strictly follow the RTCK clocking discipline, but at the expense of making the probe vulnerable to freezing if the target's RTCK signal freezes. It is recommended that you only enable this option if you are certain that the RTCK signal generation circuitry works correctly, and you experience problems when the target either runs very slowly or sleeps, and the RTCK timeout is enabled.

Note: Disabling this option is not safe for targets with improperly wired RTCK circuitry. By default this option is `on`.

step_ints

Applies to ARM Cortex-M targets only.

Sets single-stepping behavior when interrupts are enabled.

When this option is turned off, interrupts are disabled on the target during a single-step. When this option is turned on, interrupts are left alone so a single-step can go to an interrupt handler instead of the next instruction.

This option defaults to `on`.

To turn **step_ints** on or off, enter the command:

set step_ints on | off

vfp_type

Applicable to ARMv7-A cores.

Indicates whether the core has a VFP-D16 implementation, VFP-D32 implementation, or no VFP at all.

To set this option, use the following syntax:

set vfp_type none | vfp_d16 | vfp_d32

simulate_singlestep

When set to `on`, if you are single-stepping over certain branch instructions, that instruction is simulated rather than executed.

tcm_cpu_base

The base address of *tightly-coupled memory* (TCM RAM and TCM flash) differs depending on whether you access it via the CPU or AHB-AP. This option sets the base address of TCM when accessed via the CPU. To set the base address, enter the following command:

set tcm_cpu_base *base*

where *base* is the base address of TCM when accessed via the CPU.

tcm_ahb_base

The base address of TCM RAM and TCM flash differs depending on whether you access it via the CPU or AHB-AP. This option sets the base address of TCM when accessed via the AHB-AP. To set the base address, enter the following command:

set tcm_ahb_base *base*

where *base* is the base address of TCM when accessed via the AHB-AP.

tcm_size

To set the size of TCM enter the following command:

set tcm_size *size*

where *size* is the size of TCM in bytes.

toggle_hwbp

Specifies whether or not to use a workaround to fix hardware breakpoints on certain ARM cores. Do not set this option unless you know that your core requires both hardware breakpoints to be set before either of them will function. To use this option, enter the following command:

set toggle_hwbp on | off

- `on` — The probe ensures that both hardware breakpoints are set when breakpoints are in use.
- `off` — [default] do not use the workaround.

tpiu_addr Applies to ARMv7-A and ARMv7-R only. Specifies the address of the Trace Port Interface Unit (TPIU) on the APB. To specify the address, use the following syntax: **set tpiu_addr *address***
tpiu_type Applies to ARMv7-A and ARMv7-R only. Specifies whether or not there is a TPIU on your target, and if so, what kind. To specify the address, use the following syntax: **set tpiu_type** none | full | lite • `none` — [Default, if there is no APB ROM table] There is no TPIU on the target. • `full` — The target has a full TPIU. • `lite` — The target has a TPIU-Lite.
use_bkpt_inst The ARM instruction set has included a `BKPT` instruction since version 5. The `BKPT` instruction allows software breakpoints to be set without using a hardware breakpoint. When this option is set to **on** software breakpoints are set with the `BKPT` instruction, and both hardware breakpoints are available for use. Unfortunately, revisions to some of the ARM cores do not properly handle the `BKPT` instruction by dropping into debug mode, and generate an `SWI` exception instead. This option should be on if you need both hardware breakpoints, but on some ARM implementations it can cause problems when enabled with software breakpoints not being hit. It defaults to off. To change this option, enter the following command: **set use_bkpt_inst** on | off
use_hw_singlestep ARM9 cores have a hardware single-step unit. The unit was removed from the ARM9E specification, but is present on many ARM9E cores. When this option is set to `on`, the hardware single-step unit is used for single-stepping. When the option is set to `off`, single-stepping is done using hardware breakpoints. The default setting is `on` for all ARM9E cores except the ARM968E-S, which is known to lack the hardware single-step unit.

use_max_reset

Applies to ICEPick targets only.

Specifies whether the probe tells the ICEPick to attempt a soft reset when instructed to reset the target.

set use_max_reset on | off

- `on` — Instructs the ICEPick to perform soft resets. This may make resets more effective.
- `off` — [default] Does not instruct the ICEPick to perform soft resets. Use this setting if the `on` setting leaves your target in a bad state after reset.

use_rtck

Available for any adapter with the RTCK signal. When it is enabled, the **clock** option is hidden and has no effect.

This option controls whether or not the probe uses the RTCK signal to adjust the TCK clock frequency in real time to a value that is both fast and safe for the target. This signal is useful if you need to connect your probe to many targets with different core clock rates, or if the core clock rate of your target varies while you debug your program.

When this option is enabled, the probe uses an adaptive clocking scheme based on RTCK to generate the TCK debug clock signal. The maximum clock rate generated in this way is 15 MHz (if TCK and RTCK are connected together) and the minimum is slightly less than 10 kHz (if RTCK is not connected at all), unless the **rtck_use_timeout** option is disabled (it is enabled by default).

Some targets use a three-stage synchronizer circuit (documented in the ARM9EJ-S technical reference) to generate the RTCK signal. `fast_download` and `fast_always` take advantage of the specific behavior of this circuit to increase speed, especially on targets that run at clock speeds below 60 MHz.

To set this option, enter the following command:

set use_rtck off | normal | fast_download | fast_always

- `off` — [default] Do not use RTCK. Use this setting if your probe's performance becomes dramatically slower or the probe freezes with the other settings. This might mean that the RTCK signal is not properly wired on your board. If this is the case, set your JTAG clock speed manually.
- `normal` — Uses the ARM-documented *simple ping-pong* algorithm for synchronization. If the **rtck_use_timeout** option is enabled, it is usually safe to use this setting.
- `fast_download` — Supported on the Green Hills Probe with targets that have a three-stage synchronizer circuit. Increases speed while downloading or using the probe's **vm** command. The core clock speed must not change during these times. We recommend that you test download data integrity using the **vm** memory test when using this setting.
- `fast_always` — Supported on the Green Hills Probe with targets that have a three-stage synchronizer circuit. Increases speed while the target is in debug mode. The core clock speed must not change during this time.

Blackfin

Unless otherwise specified, the following configuration options are available for the targets specified by the Blackfin device names (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command).

fast_dl Enables or disables fast downloading. When **fast_dl** is turned on, the Probe assumes that the target can complete memory writes as fast as the Probe can issue them. Turning this setting on can increase memory write speed significantly in some cases. However, it can also cause incorrect operation of targets that have slow CPUs or slow memory. This option defaults to `off`. To turn fast downloading on or off, enter the command: **set fast_dl** on | off
step_ints Sets single-stepping behavior when interrupts are enabled. When this option is turned off, interrupts are disabled on the target during a single-step. When this option is turned on, interrupts are left alone so a single-step can go to an interrupt handler instead of the next instruction. This option defaults to `on`. To turn **step_ints** on or off, enter the command: **set step_ints** on | off

ColdFire

Unless otherwise specified, the following configuration options are available for the targets specified by the ColdFire device names (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command).

auto_clock Allows automatic adjustment of the BDM clock. When this option is on, the probe periodically measures the frequency of the target's core clock and automatically adjusts the BDM clock (that is, the **clock** configuration option) to the safest and fastest speed. When the **auto_clock** option is off, you can use the **set clock** command to set the clock to a higher speed than the **auto_clock** would have selected (see “Setting the JTAG or SWD Clock Speed” on page 82). This option defaults to `on`. To turn the automatic clock feature on or off, enter the command **set auto_clock** on | off
auto_clock_min Sets the minimum value for automatic adjustment of the BDM clock. **set auto_clock_min** *speed* When **auto_clock** is on, *speed* is the lowest value to which the clock can be adjusted automatically. *speed* can be specified in MHz or kHz and defaults to 5 kHz.
auto_clock_max Sets the maximum value for automatic adjustment of the BDM clock. **set auto_clock_max** *speed* When **auto_clock** is on, *speed* is the highest value to which the clock can be adjusted automatically. *speed* can be specified in MHz or kHz and defaults to 15 MHz.
auto_clock_ratio **set auto_clock_ratio** *n* When **auto_clock** is on, the probe attempts to set the BDM clock so that it is a fraction of the PST clock. The BDM clock is set to the PST clock speed divided by *n*, within the bounds of **auto_clock_min** and **auto_clock_max**, rounded to the nearest 50 kHz. The default is `12`. For example, if **auto_clock_ratio** is set to `12` and the PST clock is 100 MHz, the BDM clock will be set to 8.35 MHz.

dcache_sets Applies to ColdFire v5 and v5x only. This setting controls the number of sets in an n-way set associative cache. An n-way set associative cache is made up of a number of sets, each with a number of cache lines. The size of the cache is: `sets * ways * line_size` where *sets* is the number of sets, *ways* is the number of cache lines, and *line_size* is the size of each cache line. To change this setting, enter the command: **set dcache_sets** *sets*
dcache_ways Applies to ColdFire v5 and v5x only. This setting controls the number of ways (n) in an n-way set associative cache. An n-way set associative cache is made up of a number of sets, each with a number of cache lines. The size of the cache is: `sets * ways * line_size` where *sets* is the number of sets, *ways* is the number of cache lines, and *line_size* is the size of each cache line. To change this setting, enter the command: **set dcache_ways** *ways*
fast_dl Turns fast downloading on or off. When **fast_dl** is turned on, the Probe assumes that the target can complete memory writes as fast as the Probe can issue them. Turning this setting on can increase memory write speed significantly in some cases. However, it can also cause incorrect operation of targets that have slow CPUs or slow memory. This option defaults to **off**. To turn fast downloading on or off, enter the command: **set fast_dl** on | off

handler_base

For 5407 and v4e cores only.

Specifies a location of memory that the probe can use as scratch space for cleaning the data cache. The specified area must be 2-byte aligned and have 64 contiguous bytes of valid writable memory.

The ColdFire debug interface has no way to clean its data cache from its debug port. It has to download a small target agent to clear the data cache. The target agent is downloaded and run from the memory location specified by **handler_base**. The scratch memory used is restored after the agent runs.

The agent uses the `CPUSHL` instruction on every cache line to force a write of a dirty cache line back to memory. All pending data cache writes and dirty lines are written to memory.

To set this address, enter the command:

set handler_base *address*

where *address* is an address in available memory that is not used by the application running on the target.

The default setting is `0x2000`.

reset_clears_pst

Controls whether the probe stops monitoring the `PST` signal for target state when the target is reset. This option is useful for targets that use the `PST` pins for a purpose other than reporting target state by default. When enabled, this option works by disabling the **use_pst** option on reset. To change this setting, enter the command:

set reset_clears_pst on | off

where:

- `on` — Disables the **use_pst** option on reset. Use this setting for targets that use the `PST` pins for a purpose other than reporting target state by default.
- `off` — Leaves **use_pst** unchanged during reset. Use this option for targets that use the `PST` pins to report target state by default.

This option defaults to `off`.

use_pst

Controls whether the probe uses the `PST` signal on the BDM port to determine target state. To change this setting, enter the command:

set use_pst on | off

where:

- `on` — Determines target state using PST signals.
- `off` — Determines target state using heuristics and Configuration/Status Register (CSR) reads. This method may be unreliable on some V2 cores, such as the 5206e.

This option is turned off for the Motorola SBC5206eC3 evaluation board because its `PST` lines do not work.

MIPS

Unless otherwise specified, the following configuration options are available for the targets specified by the MIPS device names (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command).

access_mode

For Broadcom 1250 targets only.

Alters the way EJTAG instructions are fed to the processor.

This setting defaults to `0x3a`.

You should not change this setting unless directed to by Green Hills Technical Support. If you are instructed to change this setting, enter the command:

set access_mode *value*

agent

To perform some operations, the probe downloads a small program, called an agent, onto your target. This setting controls the address to which the probe downloads the agent.

The agent's default address is `0x0`. To change the **agent** setting, enter the command:

set agent *address*

Where *address* is an address in `kseg1` with 64 bytes of contiguous memory.

bp_in_delay_slots

Sets the behavior for breakpoints in delay slots.

When this option is turned on and breakpoints in delay slots are hit, the program counter appears on the branch associated with the delay slot and not on the delay slot.

When this option is turned off, breakpoints in delay slots are ignored. When the Probe hits a delay slot breakpoint, the breakpoint is removed and the Probe silently resumes program execution.

This setting can reduce confusion when the Probe hits a delay slot breakpoint and reports back to the program counter at the associated branch instead of the delay slot.

This option defaults to `off`. To turn it on or off, enter the command:

set bp_in_delay_slots on | off

dbscratch_addr

dbscratch_size

For IDT323XX targets only.

Sets the size and location of the scratch area required by IDT323xx to successfully debug code in cached memory. The scratch area needs to be at least 16 bytes and must be 4-byte aligned.

The **dbscratch_addr** and **dbscratch_size** settings default to `0`, which is incorrect for most boards. Set the address to something reasonable and the size to `0x10`. For example:

```
idt323xx[h] % set dbscratch_addr 0x80800000
idt323xx[h] % set dbscratch_size 0x10
```

Be careful that your application does not use the debug scratch memory.

fast_dl

Turns fast downloading on or off.

When **fast_dl** is turned on, the Probe assumes that the target can complete memory writes as fast as the Probe can issue them. Turning this setting on increases memory write speed by up to 300% in some cases. However, it can cause incorrect operation of targets that have slow CPUs or slow memory.

A common scenario in which fast downloading may need to be disabled is if you are using a target with a core that is synthesized into an FPGA or other programmable logic device. If your download or **vb** test fails with the error message `Unexpected write pending`, turn **fast_dl** off.

This option defaults to `on`.

To turn fast downloading on or off, enter the command

set fast_dl on | off

flush_data

Specifies whether the probe flushes the data cache and invalidates the instruction cache after memory writes. To change this setting, enter the command:

set flush_data on | off

where:

- `on` — [default] The probe flushes the data cache and invalidates the instruction cache after memory writes. It does not affect memory writes due to software breakpoints or code download.
- `off` — Do not flush the data cache and invalidate the instruction cache after memory writes.

idt_step

For IDT323xx targets only.

Specifies whether single-stepping is done with hardware or software breakpoints.

When debugging code in read-only memory, use hardware breakpoints to single-step.

To change the **idt_step** setting, enter the command:

set idt_step swbp | hwbp

where:

- `swbp` — [default] Specifies single-stepping with software breakpoints.
- `hwbp` — Specifies single-stepping with hardware breakpoints.

rst_handshake_timeout

Controls timing of reset sequence to allow handshaking protocol.

Newer MIPS targets include a handshaking protocol for performing reset. The **rst_handshake_timeout** option sets the amount of time by which the standard reset sequence can be extended to wait for this handshake to complete.

The **rst_handshake_timeout** default is 1 second. If this option is set to `0`, no handshake is attempted.

To set the **rst_handshake_timeout**, enter the command:

set rst_handshake_timeout *time*

where *time* is the length of time in seconds to extend the reset sequence.

MIPS32 4Kc and MIPS64 20Kc and 24Kc targets do not implement this handshake. If you are using one of these targets, set this option to `0`.

Note

The **rst_pulse** and **rst_settle** times are always kept as minimum times for all targets. To disable the reset sequence, set **rst_settle**, **rst_pulse**, and **rst_handshake_timeout** to `0`. (For more information about **rst_pulse** and **rst_settle**, see "Setting the CPU Reset Pulse Length" on page 86 and "Setting the Reset Settle Length" on page 86.) Never pulse the nRST line when the **rst** command is issued.

sram_config_base

Applies to MIPS4K processors only.

Specifies the physical base address of the SRAM configuration registers. If **sram_config_enable** is set to **on**, the probe tries to read these registers using the following offsets from the base address:

- `0x00` — ISRAM control register
- `0x10` — ISRAM mask register
- `0x20` — DSRAM control register
- `0x30` — DSRAM mask register

To set the physical base address, enter the command:

set sram_config_base *physical_address*

- *`physical_address`* — The physical base address.

sram_config_enable

Applies to MIPS4K processors only

Specifies whether the probe can access internal SRAM even when the user program configures the SRAM configuration registers to disallow access. To specify this option, enter the command:

set sram_config_enable on | off

- `on` — The probe watches the SRAM configuration registers at the location specified by **sram_config_base**, allowing it to access internal SRAM even when these registers disallow access.
- `Off` — The probe can only access internal SRAM as specified by the SRAM configuration registers.

step_ints

Sets single-stepping behavior when interrupts are enabled.

When this option is turned off, asynchronous interrupts are disabled on the target during a single-step. When this option is turned on, interrupts are left alone so a single-step can go to an interrupt handler instead of the next instruction.

This option defaults to `off`.

To turn **step_ints** on or off, enter the command:

set step_ints on | off

timeout Specifies the maximum timeout between JTAG accesses. This option is necessary on processors that work at a very slow core clock frequency. **set timeout** *timeout* • *timeout* — The maximum timeout between JTAG accesses, in milliseconds, from 500 to 60000. The default value is `500`, which is suitable for most processors.
use_agent Enables or disables accelerated downloads and block reads. To turn this feature on or off, use the following command: **set use_agent** on | off • `on` — [default] The probe uses a small, target-resident agent to speed up block memory accesses. This setting should not cause problems on the target as long as the **agent** option points to the start of a block of memory that is writable and executable. • `off` — The probe does not use the agent to accelerate downloads and block reads. Use this setting if you experience problems with the agent.

PowerPC

Unless otherwise specified, the following configuration options are available for the targets specified by the PowerPC device names (see "Specifying Your Target" on page 76 for information about specifying the target device name with the **set target** command). For additional usage notes regarding unchanged PowerPC configuration options, see "Notes Regarding Other PowerPC Configuration Options" on page 154.

32_bit_bus

For PowerPC 603 cores only.

PowerPC 603 targets can be configured in either 64-bit bus mode (the default for all PowerPC targets) or in 32-bit bus mode. The probe needs to know which mode the target is using to correctly access memory.

If the target is configured in 32-bit bus mode, set this option to `on`, and the probe uses a 32-bit wide data bus to access memory.

This option defaults to `off`. To change this setting, enter the command:

set 32_bit_bus on | off

agent

Applies to PowerPC 75x, 7400, 7410, 7448, 86xx, and e500v2–based cores only.

To perform some operations, the probe downloads a small program, called an agent, onto your target. This setting controls the address to which the probe downloads the agent.

The agent occupies up to 128 bytes of memory. Your program must not use any of the memory that the agent occupies.

On PowerPC 75x, 7400, and 7410 targets, the probe uses the agent to flush data from the L2 cache. If the probe can access the L2 cache on your target directly, or if the L2 cache on your target is disabled, set the agent's address to `0x0` to turn it off. If the probe cannot access the L2 cache directly (such as with the IBM 750L), set the agent's address to any nonzero, 4-byte aligned address in RAM to turn it on.

On PowerPC e500v2-based targets, the probe uses the agent only if the **freeze_timers** option is enabled. The agent helps control the timers when the target enters and exits debug mode, and helps synchronize the timers on multicore targets. Set the agent's address to an unused 32-byte aligned address in RAM that has a valid TLB mapping with supervisor execute permission.

On PowerPC 86xx targets, you must set **agent** whenever the L2 cache is enabled. On the 8641D, you must set it whenever the L2 cache or L1 data cache is enabled. Data cache way locking via `ldstcr` is not supported when the agent is enabled.

The agent's default address is `0x0`. To change the **agent** setting, enter the command:

set agent *address*
where *address* is a 64-byte aligned address (if your target is a 86xx, *address* must be 128-byte aligned).

always_inval_icache

For PowerPC 603, 82xx, 5200, 7xx, 7400, and 7410 cores only.

When set to `on`, the probe invalidates the L1 instruction cache every time a memory write is performed. Invalidating the L1 instruction cache is only necessary if instructions that the processor can execute are written to memory. Doing this for all memory writes means that writes consisting of data that the processor never executes as instructions can cause the probe to spend time unnecessarily invalidating the instruction cache.

When set to `off`, the probe assumes all memory writes consist of data that the processor will not execute, and does not invalidate the instruction cache, unless specially told to. For example, when MULTI downloads a program, it informs the probe that the data being written to memory are instructions, and the probe should invalidate the instruction cache.

Note that when performing memory writes using the **mw** and **mf** commands, placing the `i` suffix on the address being written forces the probe to invalidate the instruction cache.

This option defaults to `off`. To change this setting, enter the command:

set always_inval_icache on | off

auto_modify_tlb

For PowerPC 4xx cores only.

When debugging an application in mapped memory, if a memory operation is requested but the corresponding entry in the TLB does not allow that operation, it fails. When set to `on`, the probe modifies the entry in the TLB so that the requested operation can succeed. It then changes the TLB back to the original state so that the target state is unchanged.

When set to `off`, requested memory operations that do not have the appropriate permissions will return an error.

This option defaults to `on`. To change this setting, enter the command:

set auto_modify_tlb on | off

cache_model

Applies to PowerPC 744x and 745x targets only.

Whenever the probe initiates a memory write at a certain address, it must first check if the region of memory corresponding to the address is cached by the target. A sequential check for a valid line is done through the L1, L2, and (where applicable) L3 caches. If the address is not found to be cached, the value is written to memory. If the address is found to be cached, the probe uses one of two methods, `dirty` or `writethrough`, to maintain coherency. To set the method, enter the command:

set cache_model dirty | writethrough

- `dirty` — the probe updates the data of the first valid cache line found, and sets the status bits in that line to flag it as dirty. It does not proceed through further cache levels, or necessarily write the value to memory. This model is usually faster than the `writethrough` model, but may not work correctly in all situations.
- `writethrough` — [default] the probe updates the data of the first valid cache line found and continues searching through the cache model for a line that is both valid and dirty. If it finds a dirty line, it updates the data in that line and the memory write returns. If it does not find a dirty line after traversing all caches, it performs a write to main memory. This model is usually slower than `dirty`, because the probe checks more caches, but maintains more thorough coherency.

catch_branch

For PowerPC 603, 82xx, 5200, 7xx, and 74xx cores only.

Specifies whether or not a PowerPC core halts when a branch exception occurs. When set to `on`, the probe sets up the debug logic so that these cores halt when a branch exception occurs. When set to `off`, the probe does not halt when a branch exception occurs, but passes the exception to the branch trace exception vector (`0xd00`).

If user code does not have a branch trace exception handler, this setting should be on.

This option defaults to `on`. To change this setting, enter the command:

set catch_branch on | off

catch_illegal

Applies to PowerPC 603, 82xx, 83xx, 86xx, 5200, 7xx, and 74xx cores only.

Specifies whether or not the probe resumes the core at the program interrupt vector immediately after halting it when it encounters an illegal instruction.

If user code does not have an illegal instruction exception handler, set **catch_illegal** to `on`. In some situations (for example, if the user code is an RTOS that uses illegal instructions to set breakpoints) you may need to set this option to `off` for proper operation.

To change this setting, enter the command:

set catch_illegal on | off

- `on` — [default] When the core halts due to an illegal instruction, the probe does not resume the core at the program interrupt vector. The `PC` and `MSR` registers are moved from the corresponding `SRR` registers to point to the illegal instruction at the time of the exception.
- `off` — When the core halts due to an illegal instruction, the probe immediately resumes the core at the program interrupt vector (`0x700`).

This option has no effect if **soft_stop** is set to `off` or **disable_swbp** is set to `on`, disabling debug exceptions. In this case, the probe allows the processor to run without interruption.

catch_por

Applies to PPC4xx cores only. Requires that you set the **power_detect** option to `on`.

Determines whether or not to stop the target as it powers on by asserting the halt signal to the target when the power is off. When the probe senses that power to the target is on, it requests that the target stay halted and then releases the halt pin. This behavior requires a functioning halt signal to the target (on COP, this is usually labeled `SRST`). The default is `off`.

To enable or disable this option, use the following syntax:

set catch_por [on | off]

- `on` — Sets up debug logic to halt the core when the target first powers on, before executing the first instruction of the reset exception vector.
- `off` — Do not attempt to stop the target as it powers on.

catch_wakeup

Applies to some PPC460 cores only.

Determines whether or not to stop the target as it resumes from a low power state. The default is `off`.

The probe does this by asserting the halt signal to the target when it is in a low power state. When the probe notices that the target has left the low power state, it requests the target halt and then releases the halt pin. This behavior requires a functioning halt signal to the target (on COP, this is usually labeled `SRST`).

To enable or disable this option, use the following syntax:

set catch_wakeup [on | off]

- `on` — Stops the target from running when exiting a low power state.
- `off` — Do not attempt to stop the target as it resumes from a low power state.

censor_hi

Applies to PowerPC 56xx and some 57xx cores only.

Sets the high 32 bits of the devices censorship unlock word. For more information, see **censor_unlock**. To change this setting, enter the command:

set censor_hi *value*

censor_lo

Applies to PowerPC 56xx and some 57xx cores only.

Sets the low 32 bits of the devices censorship unlock word. For more information, see **censor_unlock**. To change this setting, enter the command:

set censor_lo *value*

censor_unlock

Applies to PowerPC 56xx and some 57xx cores only.

Some PowerPC e200–based devices have a device censorship feature that can control access to the flash subsystem and the debug interface (Nexus). When configured, the device requires a 64-bit password transmitted over JTAG before any debug functions are allowed.

The **censor_hi** and **censor_lo** settings contain the high and low 32-bits of the password, respectively. **censor_unlock** controls how the probe attempts to unlock the device during reset. To change this setting, enter the command:

set censor_unlock on | off

Where:

- `on` — The probe sends the 64-bit password to the device's `CENSOR_CTRL` register.
- `off` — [default] The probe does not attempt to unlock the device.

check_mem_access

Applies to PowerPC 405, 440, and 460 only.

Controls asynchronous machine check detection using the `MCSR` register

Newer PowerPC 405, 440, and 460 CPUs have an `MCSR` register that allows the probe to asynchronously detect machine checks caused by memory accesses. When the probe detects a machine check, it can report an error with the memory access and prevent the application on the target from seeing it when the CPU is resumed.

To change this setting, enter the command:

set check_mem_access on | off

- `on` — [default] The probe performs this check when the target has an `MCSR` register.
- `off` — The probe does not perform this check or report a memory access error. The CPU handles the exception.

databp_translate

Applies to PowerPC 86xx cores only.

On some PowerPC targets, an additional condition must be met to trigger a data (r/w) hardware breakpoint — the BT bit in the data breakpoint register (DABR[BT]) must match the data translation bit in the machine status register (MSR[DR]). The probe cannot know whether or not address translation will be enabled when the breakpoint occurs, so it makes a best guess about how to set BT based on the setting of this option.

This option applies equally to DABR and DABR2.

To change this setting, enter the command:

set databp_translate [msr | manual | off | on]

- `msr` — [default] The probe sets BT to match the value of MSR[DR] every time the processor resumes. This behavior is sufficient for most needs. Use this setting when debugging code that does not change the MSR.
- `manual` — The probe sets BT based on the `0x00000004` bit (bit 29) of the breakpoint address. If bit 29 of the address is set when the breakpoint is first created, then the probe sets BT. The value of BT is not dependent on the current MSR, and the probe does not change the value of BT when it resumes. BT maintains its value until the breakpoint is cleared. This setting may be useful for setting a data breakpoint that triggers regardless of the value of MSR[DR]. For example, to capture all r/w access to `0x1000`, you could set up two hardware breakpoints with different BT settings by issuing `bs rw 0x1000` and `bs rw 0x1004`.
- `off` — BT is always set low (0). As a result, data breakpoints are never hit if data translation is turned on in the MSR. This may be helpful for specifically debugging exception-vector code.
- `on` — BT is always set high (1). As a result, data breakpoints are never hit if data translation is turned off in the MSR. This may be helpful for specifically debugging user-space code.

disable_bp

Applies to PowerPC 4xx cores only.

This option allows you to disable all breakpoints from the probe's perspective. This allows you to do run-mode debug on your PowerPC 4xx target when the run-mode debug agent uses the TRAP instruction, which is the same as the breakpoint instruction used by the probe.

The primary use for this option is debugging a running Linux system while using the probe to set up your board and download your image to the target.

This option defaults to `off`, and should not be changed for most applications. To change this setting, enter the command:

set disable_bp on | off

disable_swt

This option does not apply to all PowerPC cores.

Specifies whether or not the probe alters the software watchdog timer (SWT) on the target as part of the reset sequence. To change this setting, enter the command:

set disable_swt on | off

- `on` — [default] The probe disables the soft lock and timer-enable bits in all SWT modules on the target as part of the target reset sequence.
- `off` — The probe does not alter the SWT configuration during reset.

disable_swbp

Applies to PowerPC 7xx, 74xx, and 86xx cores only.

Disables software breakpoints.

Some chips use the same resources for software and hardware breakpoints, which limits the available number of each type of breakpoint. By default, Green Hills Debug Probes allocate these resources to be biased toward software breakpoints, and in some cases the hardware breakpoints may not be available. Setting this option to `on` allows you to use the maximum number of hardware breakpoints available on your hardware.

On a PowerPC 8641D, if **sync_cores** is enabled, changing this option affects both cores. If **sync_cores** is disabled, changing this option affects just the current core; you can set this option independently on each core.

If this option is enabled, the **step** option should not be set to `swbp`. Use `trace` or `hwbp` instead.

This option defaults to `off`, which means that software breakpoints are enabled. To change this setting, enter the command:

set disable_swbp on | off

edm

Applies to PowerPC e500v2, e500mc, e5500, and e6500–based cores only.

Specifies whether or not to enable external debug mode (EDM).

To change this setting, enter the command:

set edm on | off

- `on` — [default] The probe takes control of all of the target's debug resources. Hardware and software breakpoints are available and the EDM bit in `dbcr0` is set to `1`.
- `off` — The probe does not set the EDM bit in `dbcr0`, and the code running on the target is responsible for managing debug resources. Hardware breakpoints are not available. Software breakpoints are available only when debugging QorIQ targets with e500mc or newer cores. Use this setting when your operating system requires use of hardware debug resources, such as when debugging Linux. The code running on the target is responsible for managing most debug resources.

Note: Changing **edm** after booting an operating system is unsupported.

execbp_translate

Applies to PowerPC 86xx cores only.

On some PowerPC targets, an additional condition must be met to trigger an execute (`x`) hardware breakpoint — the TE bit in the instruction breakpoint register (IABR[TE]) must match the instruction translation bit in the machine status register (MSR[IR]). The probe cannot know whether or not address translation will be enabled when the breakpoint occurs, so it makes a best guess about how to set TE based on the setting of this option.

To change this setting, enter the command:

set execbp_translate [auto | msr | manual | off | on]

- `auto` — [default] Identical to `msr`, except that the probe never sets TE when the breakpoint address points to the 8 k of exception vector space (either `0x0` or `0xfff00000`, as determined by the current value of MSR[IP] at the time the breakpoint is set). Usually, this setting provides the best default behavior.
- `msr` — The probe sets TE to match the value of MSR[IR] every time the processor resumes. This behavior is sufficient for most needs. Use this setting when debugging code that does not change the MSR.
- `manual` — The probe sets TE based on the least significant bit (`0x1`) of the breakpoint address: if that bit of the address is set when the breakpoint is first created, the probe sets TE. The value of TE is not dependent on the current MSR, and the probe does not change the value of TE when it resumes. TE maintains its value until the breakpoint is cleared.
- `off` — TE is always set low (0). As a result, execute breakpoints are never hit if instruction translation is turned on in the MSR. This may be helpful for specifically debugging exception-vector code or interrupt handlers.
- `on` — TE is always set high (1). As a result, execute breakpoints are never hit if instruction translation is turned off in the MSR. This may be helpful for specifically debugging user-space code.

fast_dl

For PowerPC 603, 5200, 82xx, 7xx, and 74xx cores only.

Enables or disables fast downloading.

When **fast_dl** is turned on, the Probe assumes that the target can complete memory writes as fast as the Probe can issue them. Turning this setting on increases memory write speed by up to 300% in some cases. However, it can cause incorrect operation of targets that have slow CPUs or slow memory.

This option defaults to `on`.

To turn fast downloading on or off, enter the command

set fast_dl on | off

fill_tlb For PowerPC 405 cores only. Specifies whether the probe runs the TLB handler to resolve TLB misses during a debugging session. This option defaults to `off`. To change this setting, enter the command: **set fill_tlb** on | off
fpr_buf For PowerPC 5xx cores only. Sets the scratch area required for floating-point register access on PowerPC 5xx targets. The scratch area needs to be at least 8 bytes and must be 4-byte aligned. The `fpr_buf` default is `0x7eff0`. To set the scratch area, enter the command **set fpr_buf** *address* where *address* is an address in available memory that is not used by the application running on the target.
freeze_timers For PowerPC 4xx, 55xx, 560x, 563x, 5668, and e500v2-based cores only. On these targets, the probe can freeze the timers when the processor enters debug mode. This enables any functions that rely on the timers to work correctly as though the target never halted. Additionally, when the timers are frozen, any watchdog timers do not reset the board when the target enters debug mode. On PPC e500v2, the **agent** configuration option must be configured for proper operation of **freeze_timers**. In some cases, the timers continue to run for a brief period while the probe is halting the core. When set to `on`, timers are frozen when the target enters debug mode. When set to `off`, timers continue to run while the target is in debug mode. This option defaults to `off`. To change this setting, enter the command: **set freeze_timers** on | off

icache_step

Applies to PowerPC 744x, 745x, and 86xx cores only.

When single stepping over instructions in certain contexts, these targets might unexpectedly jump to the program interrupt vector (`+0x700`). Set this option to `off` to disable the instruction cache while single stepping, which fixes this behavior in most cases.

set icache_step on | off

- `on` — Do not disable the instruction cache while single stepping. This is the default for PowerPC 744x and 745x cores.
- `off` — Disable the instruction cache while single stepping. This is the default for PowerPC 86xx cores.

immr_base

Applies to PowerPC 82xx (except 8240 and 8245) cores only.

Specifies the base address of the internal memory-mapped register set upon reset of the target.

This setting can be essential for proper operation of PowerPC 82xx targets. It is used in conjunction with the **sypcr_write_enable** option. To find the correct address, consult your board or processor manual.

The default setting is `0`.

To specify a different address, enter the command

set immr_base *address*

inval_entire_icache

For PowerPC 440 and 460 cores only.

When set to `on`, the probe invalidates the entire instruction cache whenever the address in question is marked as a physical instruction address. This is to handle instruction cache synonyms that occur when using virtual addresses that do not directly map to the same physical address.

When set to `off`, the probe does not invalidate the entire instruction cache, possibly leaving valid instruction cache synonyms for other virtual addresses that could cause modified code (such as for software breakpoints) to be missed by the processor.

This option defaults to `off`. To change this setting, enter the command:

set inval_entire_icache on | off

l2cache_size For PowerPC 75x cores only. Specifies the size of the L2 cache. The probe needs to know the correct L2 cache size so that the probe can debug a target with its L2 cache on and provide access to the L2 cache with the following commands: • **clr** • **clst** • **clsa** Note that this option does not enable the target's L2 cache. The L2 cache must be configured independently via the L2CR register. To set the L2 cache size, enter the command: **set l2cache_size** 1mb | 512kb | 256kb The default setting is `1mb`.
memory_parity For PowerPC 7400 and 7410 cores only. When connected to PowerPC 7400 and 7410 targets, the probe can optionally write the data parity bits when performing memory writes. This option also enables writing parity bits when performing L2 cache writes. If data parity checking is enabled on the target, this option must be enabled to prevent parity errors. This option defaults to `off`. To change this setting, enter the command: **set memory_parity** on | off

mmu_support

Applies to PowerPC 603, 7xx, 82xx, 83xx, 86xx, MGT5200, 7400, 7410, 7450, 4xx, e500v2, e500mc, e5500and e6500–based cores only.

Use this option to help the probe determine when and how to translate addresses. If you are debugging an operating system or other application that uses the MMU, this setting gives the probe a guideline for how to interpret addresses.

For the PowerPC 4xx, only the `auto` and `linux` settings are available. The `auto` setting has the same behavior as `on` (described below).

For the PowerPC 7450, e500v2, e500mc, e5500, and e6500, the `linux` setting is not available. If you are debugging Linux on one of these targets, use the `on` setting.

For the e500mc, e5500, and e6500, only the `on` and `off` options are available. The default is `on`.

When debugging INTEGRITY, `auto` is the recommended setting, as long as the MMU does not map any address in KernelSpace to a different physical address. If this is not true for your system, and if you encounter any problems when using the `auto` setting, try the `on` setting instead.

To change this setting, enter the command:

set mmu_support on | off | auto | linux

where the arguments specify the behavior described below.

- `on` — The probe assumes that every address is an effective address that might require address translation, and attempts to translate every address. (Memory accesses take the longest amount of time with this option selected.)
- `off` — The probe assumes that no addresses require translation. This is the recommended option if the MMU is turned off, or if all memory is directly mapped. (This option results in the fastest possible memory access by the probe.)
- `auto` — The probe assumes that only user-mode memory addresses might need to be translated, and that supervisor-mode memory addresses are directly mapped. If the MMU is disabled, or the processor never runs in user mode, this option has the same performance as the `off` setting.
- `linux` — The probe assumes that target is running a Linux operation system and attempts to translate addresses assuming a Linux memory map.

The default setting is `auto`.

no_boot_rom

For PowerPC e500v2–based cores only.

These cores require valid memory and opcodes at the reset vector for the probe's reset sequence to work. Boards that do not have valid flash or ROM located at the reset vector can be correctly reset when this option is enabled.

When set to `on`, and a target reset operation is requested, the probe maps half of the CPU's internal L2 cache to the reset vector, and writes a "branch to self" opcode to the reset vector. This setup work is done while the target is held in reset. When the reset is complete, the probe leaves the L2 cache mapped as `SRAM`, and also initializes all of the `IVOR` registers so that the code can immediately be run in the highest page of memory.

When set to `off`, the probe's reset behavior is determined by the **reset_fixup** configuration option.

The default setting is `off`. To change this setting, enter the command:

set no_boot_rom on | off

override_rcw This configuration option affects how the probe resets the target. When set to `off`, the probe relies on the target having a valid reset configuration word (RCW) available to the processor at reset. If the processor cannot read a valid RCW, or if the RCW is misconfigured, the probe may not be able to reset the target. Having an erased or missing flash part can lead to an invalid RCW. For 83xx, when set to `on`, the probe will override the RCW using the values specified by the **rcw_high** and **rcw_low** configuration options. The processor will use the RCW specified by the probe and not try to look for one in the usual places. This should allow the probe to successfully reset the target every time; however, if a target does not have a valid RCW, it may not boot when the probe is not connected. It is recommended that this option only be set to `on` in order to recover the actual RCW, or during board bring-up (to troubleshoot why an existing RCW does not work). Whenever you reset your 83xx target with **override_rcw** enabled, the **rcw_high** and **rcw_low** settings are stored on the target. Those values are used for all subsequent resets until the target is power-cycled, or you re-enable **override_rcw** with new values. For QorIQ targets with e500mc or newer cores, the target's RCW is 512 bits in length. The probe provides access to the RCW value with 16 registers `rcw1` through `rcw16`. These registers are populated with the current RCW value at each target reset. The top bit of `rcw1` corresponds to bit 0 of the RCW described in the processor reference manual. To override the target's RCW, first write to one or more of the `rcw5-rcw16` registers. Note that overriding the values of `rcw1` through `rcw4` is not supported. Then set **override_rcw** to `on`, and finally reset the target. On QorIQ, the **override_rcw** option is a volatile option and must be explicitly enabled before each reset. The default for this option is `off`. To change this setting, enter: **set override_rcw** on | off
preserve_dcache Applies to PowerPC 440 and 460 targets only. Controls whether or not memory accesses through the probe alter the data cache. Set this option to `on` if you are using the **cacheview** command in the MULTI Debugger or are debugging cache-related issues. Otherwise, set this option to `off`. **set preserve_dcache** on | off • `on` — [default] memory accesses through the probe do not alter the data cache. The probe checks each address accessed, and if the address is not already in the cache, it temporarily modifies the TLB to prevent that address from being cached. • `off` — memory accesses through the probe alter the data cache if the accessed address is not already in the cache. This setting improves performance.

quiesce

Applies to PowerPC e500v2–based processors supported by probe firmware 3.8 or newer.

Specifies whether or not a platform quiesce is issued when a core enters debug mode. To change this setting, enter the following command:

set quiesce on | off

- `on` — Any time a core halts, the probe sends a quiesce request to the SoC platform. This request suspends all activity in blocks such as the DMA controller, and prevents the possibility of a probe debug command from interfering with an active transfer. We recommend that you enable this option when debugging an operating system that makes use of DMA. However, in multi-core systems like the QorIQ P2020, whenever one core halts, the other core halts as well. This behavior may not be desirable for AMP-type applications.
- `off` — [default] No platform quiesce is issued on halt. Cores can be halted independently. This setting is appropriate for most stand-alone applications.

rcw_high

rcw_low

Applies to PowerPC 83xx cores only.

The values specified by **rcw_high** and **rcw_low** can be used to configure the reset configuration word during reset. To view the values of the reset configuration word after reset, view the memory-mapped registers RCWHR at offset `0x904` from the IMMR base and RCWLR at offset `0x900` from the IMMR base. For more information about how the fields in these registers are defined, refer to your processor reference manual.

The values of these configuration options have no effect when the **override_rcw** setting is disabled. For more information, see the **override_rcw** option in this table.

The syntax for these options is:

set rcw_high *value*

set rcw_low *value*

where *value* is the 32-bit value to write to the specified register during reset.

reset_erpn

For PowerPC 440 and 460 cores only.

When performing a debug-mode reset (**tr d**), the probe initializes the TLB on the 440 core to contain a single entry mirroring what would be added to the shadow TLB on reset. This option controls the Effective Real Page Number (ERPN) of that entry.

This option defaults to **1**, as per the hardware user manual. To change this setting, enter the command:

set reset_erpn *ERPN*

where *ERPN* is between `0` and `0xF` (inclusive).

reset_fixup

For PowerPC e500v2–based cores only.

These cores have a chip bug that prevents accessing the TLB or performing a single-step directly out of reset. Setup scripts typically reset the target, and then often need to enable entries in the TLBs. It might be useful to single-step to debug the instructions executed at CPU reset. This configuration option enables a workaround so that these features will work after a reset is performed. The workaround procedure is the same as the procedure used by the **no_boot_rom** configuration option.

If the **no_boot_rom** configuration option is on, **reset_fixup** is ignored. Also, the workaround is not used if a **tr r** command is issued because it is not required in that case.

When set to `on`, the probe performs the **no_boot_rom** procedure during reset so that the `TLB` and single-step features will work properly. The main difference between enabling **no_boot_rom** and **reset_fixup** is that this option returns all registers and the `L2` cache to their default state out of reset when the procedure completes.

When set to `off`, the probe performs its standard reset operation.

The default setting is `on`. To change this setting, enter the command:

set reset_fixup on | off

reset_sequence

For PowerPC 603, 82xx, and 5200 cores only.

Sets the software breakpoint method.

Some PowerPC targets based on the G2 core have a quirk when resuming from the reset vector. They stop a few instructions after the reset vector for no apparent reason. The probe typically reports this as an unknown exception. This option attempts to prevent this quirk from occurring by offering an alternate reset sequence.

For some targets, whether or not they require the alternate reset depends on the contents of the flash. Typically, if the flash is empty, the normal reset can work. If the flash has valid data, then the alternate reset is required. The `auto` setting is the best choice for these targets because it selects the reset sequence based on the contents of flash.

To change this setting, enter the command:

set reset_sequence normal | alternate | auto

where the arguments specify the behavior described in the following.

- `normal` — The probe uses the standard COP sequence to reset the target.
- `alternate` — The probe uses a slightly different mechanism to reset the target, which sets a hardware breakpoint at the reset vector and runs to it twice. This can prevent unknown exceptions from occurring the first time the processor is resumed after reset.
- `auto` — The probe looks at the contents of flash at the reset vector. If the flash is erased, the probe performs a normal reset sequence. If the flash has valid opcodes, the probe performs the alternate reset sequence.

The default setting is `normal`.

reset_type

For PowerPC 4xx cores only.

With PowerPC 4xx cores, the probe can reset the target using one of three reset types.

To change this setting, enter the command:

set reset_type core | chip | system

- `core` — Resets the processor core.
- `chip` — Resets the processor core, as well as all on-chip peripherals.
- `system` — Performs a `chip` reset and asserts the system reset pin. A precise reset to the reset vector can only be achieved if the `HALT` signal (`SRST` for the COP interface) is properly connected to the CPU; otherwise, the CPU runs free. For more information about processor reset types, see your processor's manual.

The default setting is `chip`.

For more information about processor reset types, see your processor manual.

reset_vector For PowerPC 603, 7xx, 7400, 7410, and 82xx cores only. Specifies the location of the reset exception vector. (The probe catches certain PowerPC exceptions to perform its debugging actions.) To change this setting, enter the command: **set reset_vector** *address* where *address* is either `0xfff00100` or `0x00000100`. The default setting is `0xfff00100`.
running_mem_access Specifies how the probe controls access to memory while the core is running. **set running_mem_access** allow | auto | disallow where: • `allow` — [default] Always allows access to memory while the core is running. The probe assumes all addresses are physical, and the accessed data may not be cache coherent. • `auto` — Uses a heuristic to determine whether or not to allow access to memory while the core is running. If the access to the address would return incoherent data or if the address is specified as virtual, the probe returns an error. • `disallow` — Never allows access to memory while the core is running.

safe_lsrl

For PowerPC 744x and 745x cores only.

Specifies whether the probe uses the LSRL scan chain when reading or writing certain PowerPC special purpose registers such as L2CR, L3CR, and DEC. On PowerPC 744x and 745x, these registers can be accessed quickly by using the shorter MSS_NRM scan chain. However, such accesses can cause spurious decrementer and thermal exceptions when external interrupts are enabled.

If external interrupts are disabled, this setting should not affect probe behavior except for the speed of sized memory writes.

To change this setting, enter the command:

set safe_lsrl on | off

where:

- `on` — Causes the probe to use the LSRL chain. This can slow down sized memory writes (1, 2, 4, 8 bytes), but prevents the probe from causing decrementer and thermal exceptions upon resuming the processor.
- `off` — Causes the probe to use the shorter MSS_NRM chain. If external interrupts are enabled, when the probe resumes the processor, the processors have a decrementer exception, and possibly a thermal exception pending. The thermal exceptions only occur on PowerPC 7445 and 7455.

This option defaults to `on`.

scratch_addr

For PowerPC 4xx cores only.

If `use_cache_scratch` is set to `off`, `scratch_addr` specifies where the probe can find 16 bytes of scratch memory used for tasks such as FPU register access. This memory is backed up before and restored after use, so it can be shared with the running program.

set scratch_addr *address*

Where *`address`* is a 32–bit, 16–byte aligned address that points to usable RAM.

The default value for *`address`* is `0x4000`

service_bus_read32

Applies to PowerPC 744x and 745x cores only.

Specifies whether memory is read using the service bus or by stuffing instructions. The service bus can read memory faster, but might have difficulty accessing certain memory ranges, such as off-chip memory-mapped registers. Instruction stuffing is slower, but more reliable for accessing any memory region that code can access. This option only applies to 1-, 2-, 4-, and 8-byte reads.

To change this setting, enter the command:

set service_bus_read32 on | off

where:

- `on` — Causes the probe to use the service bus for all memory read requests of 1, 2, 4, and 8 bytes. This is faster, but may not be able to read all memory regions. Some 7447 and 7457 cores might time out when reading memory. If this happens on your target, set this option to `off`.
- `off` — [default] Causes the probe to use instruction stuffing for all memory read requests of 1, 2, 4, and 8 bytes. This is slower, but might be able to read more memory regions.

simple_rst_run

Does not apply to all processors. If you attempt to set this command on an unsupported processor, the probe reports a `no match` error.

Specifies whether or not the probe performs a simple reset instead of a precise reset when a reset and run command (such as **tr r**) is issued. During a precise reset, the probe performs some maintenance that may be required in order to put the target in a good state.

If precise resets do not work, but all other run control does work, the TAP reset line and CPU reset line may be tied together. Try configuring the **target_reset_pin** setting before configuring **simple_rst_run**.

To change this setting, enter the command:

set simple_rst_run on | off

- `on` — The probe resets the target by toggling the reset lines on the JTAG port, but does not perform any additional debug initialization. This setting might help if your target has a valid ROM monitor, but precise resets do not behave as expected.
- `off` — [default] The probe performs a precise reset, and then resumes the target.

slow_memory_read For PowerPC 603, 82xx, 5200, 7xx, 7400, and 7410 cores only. On some faster PowerPC targets, such as 750FX and 7400, the CPU may need additional clock cycles to read the contents of slower memory, such as ROM or flash memory. When set to `on`, this option instructs the probe to run the target for additional clock cycles when performing memory reads. Note that this slows down all memory reads by a small amount. When set to `off`, the probe uses the default number of clock cycles when reading data from memory. This option defaults to `on`. To change this setting, enter the command: **set slow_memory_read** on | off
soft_stop For PowerPC 603, 5200, 82xx, and 83xx cores only. PowerPC processors based on the G2 core (603, 5200, 82xx, and 83xx) have a feature called “soft stop” that, when enabled, causes the processor to halt and enter debug mode any time an illegal instruction, hardware breakpoint, or trace exception is encountered. Even if there are no breakpoints set by the user, the target enters debug mode if the code running on the target causes such an exception. This can be passed back to the processor using the **catch_illegal** (above) configuration option, but with significant overhead. Disabling **soft_stop** prevents the processor from entering debug mode for any reason except a user-requested halt. Furthermore, the revision 1 PowerPC 5200 cores have a bug that results in a loss of memory access if the processor enters debug mode while another DMA bus master is enabled. For example, using the processor's Ethernet device can cause this problem. One way to avoid this is to prevent the processor from entering debug mode, which essentially makes the probe only a boot loader To change this setting, enter the command: **set soft_stop** on | off where: • `on` — Allows the processor to soft stop, or halt, when it encounters an illegal instruction, which is used for software breakpoints. This is the recommended setting for standard debugging operation. • `off` — Prevents the processor from soft stopping when it encounters an illegal instruction, which is used for software breakpoints, hardware breakpoints, or the SE and BE bits in the MSR. Instead it jumps to the appropriate exception vector, as if the probe were not attached at all. This means that breakpoints will not work. However, the probe can still halt the processor in this mode. This setting is particularly useful for downloading and running INTEGRITY with user space tasks. This option defaults to `on`.

sram_size Some PowerPC targets have internal SRAM protected by ECC bits. These bits must be initialized before use, or a machine check occurs. This option specifies how many bytes of the target's SRAM to initialize whenever a reset is performed. To change this setting, enter the command: **set sram_size** [*size*] where *size* specifies the number of SRAM bytes to initialize. The default value for this setting varies depending on your target setting. If there are multiple chip variants supported by a particular target, the default value is set based on the chip variant with the largest SRAM size. Setting *size* to `0` disables SRAM initialization.
step Specifies which method the probe uses to single-step through code. There are three available methods, but none of the three work in every case because of bugs in the PowerPC hardware. If you experience trouble single-stepping, changing the **step** setting might help. To change this setting, enter the command: **set step** hwbp | swbp | trace where: • `hwbp` — Causes a hardware breakpoint to be set at the next address. This step method fails in certain cases just after a branch has been taken, but does allow ROM debugging. • `swbp` — Causes a software breakpoint at the next address. This step method works as long as the code you are debugging is in memory that can be written. It fails if the code resides in ROM or flash, or if **disable_swbp** is enabled. • `trace` — Enables stepping using the chip's built-in single-step functionality. This step method does not properly single-step over all instructions. On PowerPC 82xx, it fails for some **mtspr** and **mfspr** instructions. On other targets, it might fail for certain instructions when the instruction cache is enabled. The default setting is `swbp`. Some special care may be needed if code that the probe is debugging on the target includes an **mtmsr** instruction. This instruction cannot be stepped over using trace step. Also, if an **mtmsr** changes the `MSR[IP]` bit, then breakpoints will not work after the **mtmsr** occurs, until the target is halted. One solution to this problem is to modify the setup script so that it sets the `MSR[IP]` bit to the same value that the **mtmsr** instruction sets.

step_fixup Applies to PowerPC e500v2–based cores only. Controls whether or not the probe uses a workaround to prevent problems when single-stepping onto a software breakpoint. This option is necessary because in some cases, these processors cannot single-step onto a software breakpoint without causing an exception or otherwise corrupting the state of the processor. To turn **step_fixup** on or off, use the command: **set step_fixup** on | off where: • `on` — [default] the probe disables any software breakpoint set on the next instruction before performing a single step. The probe restores the breakpoint after the step completes. • `off` — the probe performs single steps without checking the next instruction for a software breakpoint.
step_ints Sets single-stepping behavior when interrupts are enabled. When this option is turned off, asynchronous interrupts are disabled on the target during a single-step. When this option is turned on, interrupts are left alone so a single-step can go to an interrupt handler instead of the next instruction. This option defaults to `on`. To turn **step_ints** on or off, enter the command: **set step_ints** on | off

swbp_type For PowerPC 5xx and 8xx cores only. Sets the software breakpoint method. There is no single interrupt that is guaranteed to be available to the Probe for software breakpoints. You should choose a breakpoint method that does not cause a conflict with the application running on the target. Notice that this setting directly affects some of the bits in the DER register, as described in the following. To set the software breakpoint method, use the command: **set swbp_type** illegal | syscall | trap where the arguments specify the behavior described in the following. • `illegal` — The software emulation interrupt is used to cause software breakpoints by writing an illegal instruction to memory where software breakpoints are used. Specifying this option sets SEIE in DER. • `syscall` — The system call interrupt is used to cause software breakpoints by writing the `syscall` instruction to memory where software breakpoints are used. Specifying this option sets SYSIE in DER. • `trap` — The program interrupt is used to cause software breakpoints by writing a `trap` instruction to memory where software breakpoints are used. Specifying this option sets PRIE in DER. The default setting is `syscall`.
swcrr_value **swcrr_write_enable** Applies to PowerPC 83xx cores only. Please refer to the description of the **sypcr_value** and **sypcr_write_enable** options (below). The same information applies to **swcrr_value** and **swcrr_write_enable**, which are named differently only because on PowerPC 83xx, the watchdog timer is disabled via fields in the SWCRR register instead of the SYPCR register.

sync_cores

Applies to the PowerPC 8641D only.

Specifies whether or not the probe uses special 8641D debug features to enable synchronous debugging. Set this option to **on** if you are debugging a program that shares memory between the two cores (for example, an SMP INTEGRITY Kernel).

You must set the **agent** option correctly when using **sync_cores** in order to maintain memory coherency while debugging.

To change this setting, use the command:

set sync_cores on | off

where:

- `on` — Enables synchronous debugging. The probe couples the run-control operation of the cores so that a **tc** or **th** command issued to one core affects both cores. It also links software breakpoints so that both cores halt when either core hits a breakpoint.
- `off` — [default] Does not enable synchronous debugging. The probe treats the two cores as separate targets.

sync_etpu

Applies to PowerPC 55xx and 56xx cores only.

Configures run control behavior for any enhanced time processor unit (eTPU) cores on the target.

The probe treats all cores independently, so they must be resumed and halted individually. Because the eTPU is a part of the processor, we recommend that you resume an eTPU any time the core is resumed. When you set this option to `on`, the probe keeps the eTPU in sync with the processor.

To change this setting, use the command:

set sync_etpu on | off

where:

- `on` — [default] Configures the probe to resume the processor's eTPU whenever it resumes the processor.
- `off` — Configures the probe so that resuming the processor has no effect on the eTPU. Run control commands execute only on the individual core where the command was issued.

sypcr_value

sypcr_write_enable

For PowerPC 5xx, 8xx and 82xx (except 8240 and 8245) cores only.

Enables the Probe to write a value to the SYPCR register just after reset, which makes debugging easier on these specific targets.

The preceding PowerPC processors include an internal watchdog timer. In the default configuration, debugging is difficult if the watchdog timer is not 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 configuration values enable the probe to write a value to the SYPCR register just after reset.

The syntax for using these options are:

set sypcr_write_enable on | off

set sypcr_write_value *value*

If **sypcr_write_enable** is turned on, the Probe writes the *value* specified by **sypcr_value** to the SYPCR register immediately after reset.

By default, **sypcr_write_enable** is `on` and the value of **sypcr_value** is `0xffffff03`. This default configuration causes the Probe to disable the watchdog timer just after target reset.

target_rev

Currently, for PowerPC 750 and PowerPC 7445/7455 cores only.

Specifies the revision number of a target.

The probe can debug different revisions of PowerPC 750, 7445, and 7455 targets. Because the debug port differs between revisions on some of these processors, the probe needs to know the exact revision number to successfully debug the target. Usually, the probe can determine the revision number of these targets automatically. However, if the probe has difficulty determining the revision number, you can use the following syntax to set the revision number manually:

set target_rev *revision*

where *revision* is one of the following:

- `750-r2` — For a PowerPC 750 Revision 2.2 or earlier target
- `750-r3` — For a PowerPC 750 Revision 3.0 or later target
- `750CX-r1` — For an IBM PowerPC 750CX Revision 1.x target
- `750CX-r2` — For an IBM PowerPC 750CX Revision 2.0 or later target
- `750FX-r1` — For an IBM PowerPC 750FX Revision 1.x target
- `750FX-r2` — For an IBM PowerPC 750FX Revision 2.0 or later target
- `750L-r2` — For an IBM PowerPC 750L Revision 2.x target
- `750L-r3` — For an IBM PowerPC 750L Revision 3.0 or later target
- `750GX-r1` — For an IBM PowerPC 750GX Revision 1.x target
- `rev2` — For a PowerPC 7445/7455 Revision 2.x target
- `rev3` — For a PowerPC 7445/7455 Revision 3.x target

Example:

```
>set target_rev 750CX-r2
```

Tells the probe that the target attached is a PowerPC 750CX Revision 2.

To force the probe to attempt to determine the revision number automatically (if you have previously issued the command `set target_rev` *`revision`*), enter the command:

set target_rev auto

The **detect** command (see "Configuration Commands" on page 175) cannot override the **target_rev** configuration setting. If **detect** senses a particular processor revision, but the **target_rev** setting is set to a different revision, the target may not function properly. Therefore, it is best to set **target_rev** to `auto` before using **detect** with one of these targets.

tlb_handler_entry

When used in conjunction with the **tlb_handler_failure** and **tlb_handler_success** settings, this setting extends the address translation capabilities of the probe when running a supported operating system. Do not set this option manually unless instructed to do so by Green Hills Support.

If this setting is set to `0x0`, the probe issues the following diagnostic when a translation attempt misses in both `TLB1` and `TLB0`:

```
ERROR 77: error translating  virtual address
```

tlb_handler_failure

When used in conjunction with the **tlb_handler_entry** and **tlb_handler_success** settings, this setting extends the address translation capabilities of the probe when running a supported operating system. Do not set this option manually unless instructed to do so by Green Hills Support.

tlb_handler_success

When used in conjunction with the **tlb_handler_entry** and **tlb_handler_failure** settings, this setting extends the address translation capabilities of the probe when running a supported operating system. Do not set this option manually unless instructed to do so by Green Hills Support.

tlb_miss_error

For PowerPC 85xx, P1xxx, and P2xxx cores only.

Specifies whether the probe reports a TLB miss as an error.

set tlb_miss_error on | off

Where:

- `on` — The probe reports error `77` whenever a TLB miss occurs. These errors are only reported when the probe attempts to translate a virtual address, as determined by the **mmu_support** configuration option.
- `off` — When a TLB miss occurs, the address is treated as a physical address.

The default setting is `off`.

use_cache_scratch

For PowerPC 4xx cores only.

Specifies whether a cache line or RAM is used as scratch memory when reading FPU registers.

set use_cache_scratch on | off

Where:

- `on` — Configures the probe to use a cache line for scratch memory.
- `off` — Configures the probe to use RAM for scratch memory.

The default setting is `on`.

user_immr

For PowerPC 82xx cores only. (This option only needs to be set if you plan to access the PCI registers.)

Specifies the IMMR value used by the probe to read IMMR-based registers during a debugging session.

After reset, the probe sets the IMMR to **immr_base**, but does not use **immr_base** to access IMMR-based registers. The target might be running code that changes the IMMR. **user_immr** should be set to match the target IMMR value, rather than the **immr_base** value, if the probe is used to debug the target after IMMR-changing code is run.

To set the value of **user_immr**, enter the command:

set user_immr *address*

The default value of **user_immr** is `0x04700000`.

Notes Regarding Other PowerPC Configuration Options

The following additional notes concern PowerPC configuration options that are described in "Target-Specific Configuration Options" on page 92.

- When debugging a PowerPC 8260 target, you must properly set the **immr_base** configuration option to the base address of the internal memory-mapped registers upon reset. The default address is 0. To change the **immr_base** setting, enter the command:

  set immr_base *address*

- The recommended setting for the **fast_dl** (fast downloading) configuration option (for Power PC 603ev, 7xx, 74xx, and 82xx targets) is `on`. Enabling fast downloading provides a dramatic improvement to download speed on some targets, and also causes the probe to perform a rapid cache invalidation after the download. If the **fast_dl** setting is `off`, the probe uses a slower method to invalidate the L1 instruction cache and the L2 cache, which can take longer than 15 seconds. This might cause MULTI to time out and ask if you want to terminate. If MULTI does time out, do not choose to terminate. Instead, give the probe more time to finish the cache invalidation.

If your target is running slowly, or you are getting timeouts during or immediately after downloads, verify that **fast_dl** is enabled. To verify that enabling **fast_dl** is reliable on your system, run the **vm** test (see “Green Hills Debug Probe Commands” on page 174 for a description of the syntax for the **vm** command).

SH

Unless otherwise specified, the following configuration options are available for the targets specified by the SH device names (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command).

step_ints

Sets single-stepping behavior when interrupts are enabled.

When this option is turned off, interrupts are disabled on the target during a single-step. When this option is turned on, interrupts are left alone so a single-step can go to an interrupt handler instead of the next instruction.

This option defaults to `off`.

To turn **step_ints** on or off, enter the command:

set step_ints on | off

V800

There are no target-specific configuration options for V800 targets.

x86

The following configuration options are available for the targets specified by the x86 device names (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command).

enable_debug

Specifies how debug exceptions are handled. Changes to this option will take effect the next time the probe resumes the target.

set enable_debug on | off | auto

- `on` — [default] Debug exceptions are handled by the probe. This setting is recommended for standard debug operation (that is, for freeze-mode debugging only).
- `off` — Debug exceptions are handled by the code in the processor's debug exception handler. This setting is recommended if the target is running an operating system with debug features, such as INTEGRITY and if you are doing most of your debugging using a run-mode debug agent, and the probe is primarily used to download and run code, not to debug it. While this setting is enabled, you cannot set any freeze-mode breakpoints except software breakpoints on Sandy Bridge processors.
- `auto` — Debug exceptions are handled by the probe or by the code in the processor's debug exception handler, as appropriate.

  The presence or absence of freeze-mode breakpoints determines how exceptions are handled. If present, debug exceptions are handled by the probe. If absent, debug exceptions are handled by the code in the processor's debug exception handler.

  On Sandy Bridge, only hardware breakpoints determine how exceptions are handled, meaning that you can set freeze-mode software breakpoints without affecting run-mode debug functionality.

  The manner in which exceptions are handled may change when:

  - a freeze-mode breakpoint is set
  - all freeze-mode breakpoints are removed

  This setting allows for easier transitions between run-mode and freeze-mode debugging.

scratch_addr

Specifies where the probe can find 128 bytes of scratch memory. This memory is backed up before and restored after use, so it can be shared with the running program. This memory is only used to access floating point registers.

set scratch_addr *address*

Where *`address`* is a 32-bit, 16-byte aligned address that points to usable RAM. If *address* is set to `0`, the probe does not attempt to access floating-point registers.

skip_smm

Allows you to skip System Management Mode (SMM). In SMM, the target cannot access the full register set and has other reduced functionality which may cause unpredictable behavior.

set skip_smm on | off

- `on` — If the target is halted in SMM, the probe resumes the target so that it halts as soon as it has exited SMM. If it does not exit SMM, there is an error.
- `off` — Does not attempt to skip SMM.

use_gtl

Specifies whether the probe enables the GTL drivers on a GTL-capable adapter or bypasses them to use the native CMOS-style drivers. When the GTL drivers are enabled, the **logic_high** option is not available.

set use_gtl on | off

- `on` — Instructs the probe to enable GTL drivers.
- `off` — Instructs the probe to disable the GTL drivers and allows the **logic_high** option to be set.

This option is specific to GTL-capable TE adapters.

XScale

The following configuration options are available for the targets specified by the XScale device names (see “Specifying Your Target” on page 76 for information about specifying the target device name with the **set target** command).

auto_vector_reload

Sets the probe policy for automatically reloading vectors on vector traps.

On XScale targets the probe must verify that the target's exception vector opcodes are the same as those stored in the mini-I-cache for debugging, or else exceptions cannot execute correctly. Enter the command:

set auto_vector_load off | once | always

to specify the policy for reloading vectors where the settings specify the following behavior:

- `off` — Disables automatic reloading of vectors on vector traps.
- `once` — Causes the probe to trap the first exception taken after reset (regardless of the **catch_*exception*** settings), reload the exception vectors, and depending on the value of the **catch_*exception*** setting corresponding to the exception taken, either transparently resume the target or remain halted at the exception vector. Subsequent exceptions are only trapped if the corresponding **catch_*exception*** setting is turned on.
- `always` — Causes all exceptions to be caught and the vectors reloaded. If the corresponding **catch_*exception*** setting for an exception is not enabled, the probe transparently resumes the target.

  Normally, the **always** setting should not be enabled because it lengthens the service time for all exceptions (by as much as several milliseconds).

The default **auto_vector_reload** setting is `once`, which works for most probe-unaware programs.

catch_*exception*

Determines whether or not to halt the core before executing the first instruction of the exception handler for *exception*. The following values for *exception* apply to XScale:

- `abort` — Data Abort
- `fiq` — FIQ
- `irq` — IRQ
- `prefetch` — Prefetch Abort
- `reset` — Reset
- `swi` — Software Interrupt (SWI)
- `undef` — Undefined Instruction

The default setting for all **catch_*exception*** options is `off`.

To turn abort checking on or off, enter the command:

set catch_*exception* on | off

- `on` — Sets up the debug logic so the core halts before executing the first instruction of the exception handler.
- `off` — The probe does not enable the trap logic, and exceptions execute normally.

  For **catch_reset**, the probe still sets `DCSR.TR` to maintain debug control across a reset exception, but transparently resumes the core from trapped reset exceptions.

For example:

set catch_abort on

early_reset_release

Only for targets specified by the `xscale` target device name.

Controls how the NSRST line is used when the probe first takes control of an XScale core.

When the probe target is set to the generic `xscale` type, **early_reset_release** configures whether or not the probe releases the NSRST line before loading the mini-I-cache. If you have configured the probe for a specific type of XScale processor (setting the device type to `i80200`, for instance, rather than using the generic `xscale` parameter), the probe automatically uses the NSRST line correctly for the specified XScale processor, and the **early_reset_release** configuration option is hidden.

To specify the NSRST behavior, enter the command:

set early_reset_release off | on

where the settings have the following effects:

- `off` — Causes the probe to release the NSRST line before loading the mini-I-cache, relying on the `DCSR.HLD_RST` bit to keep the processor in reset
- `on` — Causes the processor to keep the NSRST line asserted while loading the mini-I-cache.

The `off` setting is correct according to the Intel XScale debug specification, but does not work for the i80321 (IOP321) processor. The `on` setting works for the i80321 (IOP321), but does not work on the PXA210 or PXA250 processors. The i80200 processor seems to work equally well with either setting. If the **tr** command returns an error, changing this setting might help.

The option defaults to `on`.

handler_base

Specifies the base address of the debug handler used to facilitate debugging of the XScale.

The Probe requires a small section of memory on the target (approximately 2-KB in size and aligned on a 2-KB boundary). The starting address of this section is specified by the **handler_base** setting. This address can be anywhere within the first 32-MB of memory that does not overlap with any addresses used by the code or data of your application or any other devices on the target, such as flash ROM or memory-mapped I/O devices. This address is actually locked in the XScale mini instruction cache. It can and must be mapped to an address that does not map to real memory or devices.

The default **handler_base** setting is `0x5000`.

To specify a different address, enter the command:

set handler_base *address*

read_after_write_mem

You can set this option to **on** to read back memory after it has been written for XScale cores. This fixes cache errata on some XScale processors, but might also lead to unpredictable behavior when writing to write-only memory.

To enable or disable this option, use the following syntax:

set read_after_write_mem [on | off]

override_mmu

Set this option to bypass the permissions set in the MMU, and access all memory with administrator privileges while debugging. **override_mmu** enables reading and writing protected memory, and setting software breakpoints in protected memory.

To enable or disable this option, use the following syntax:

set override_mmu [on | off]

override_mpu

Set this option to bypass the permissions set in the MPU, and access all memory with full read/write access while debugging. **override_mpu** enables reading and writing protected memory, and setting software breakpoints in protected memory.

To enable or disable this option, use the following syntax:

set override_mpu [on | off]

use_new_memwrite

This option specifies whether or not the probe uses a memory write algorithm that is compatible with the PXA320 processor.

To enable or disable this option, use the following syntax:

set use_new_memwrite [on | off]

- `on` — Use a new memory write algorithm that communicates with the debug handler in a way that is not affected by errata 5.7 to the PXA320 processor. Use this setting if your target is a PXA320.
- `off` — [default] Use an older memory write algorithm. This algorithm violates errata 5.7 to the PXA320 processor, which places restrictions on how the probe can communicate with the debug server.

Target-Specific Trace Options

This section describes the target-specific trace configuration options available in the MULTI IDE. For information about generic trace options, see the *MULTI: Debugging* book.

Note

If you manually configure your target's trace options and their values change each time you download your program, check your setup script to see if it sets the option while configuring your target. If this is the case, update the setup script to use the correct option.

ARM ETM and PTM Target-Specific Options

Each option is described in the following table. For more detailed information about these options, refer to the ETM specification from ARM.

Port Mode

Selects the ETM clocking mode used on the target system.

There are three choices for ETMv1.x and ETMv2.x targets:

- **Normal** — Most targets only support this mode.
- **Multiplexed** — Uses half as many pins to output trace information, but twice the clock rate.
- **Demultiplexed** — Uses twice as many pins and half the clock rate. This mode typically requires a special adapter with dual Mictor connectors.

Multiplexed and **Demultiplexed** modes were eliminated with ETMv3.x. However, many ETMv3.x targets support several ETM clock multipliers relative to the processor core clock. All of the ETMv3.x clocking modes are listed below. Note that the ratios are of ETM data rate to core clock speed. The ETM data rate is twice the ETM clock speed because ETMv3.x always outputs data on both clock edges.

- **2:1**
- **1:1**
- **1:2**
- **1:3**
- **1:4**
- **Implementation Defined**
- **Dynamic**

Port Size/CoreSight Port Size

Selects the ETM port size. This corresponds to the number of bits that will be drained from the ETM FIFO each cycle. Larger port sizes mean that more data can be generated without causing an overflow. It is generally desirable to use the largest port size supported by the target system.

For CoreSight targets, this option displays as **CoreSight Port Size**.

Data Capture

Selects the data access information that will be traced. The ETM can either trace data access addresses and values, addresses only, or values only.

Note: To reduce the number of ETM FIFO overflows in your trace data, you can set this option to **Address Only**. However, this setting reduces the effectiveness of TimeMachine by preventing reconstruction of register and memory values. For more information, see the documentation about incomplete trace data in the *MULTI: Debugging* book.

Data Only Trace

Enables data only mode for ETMv3.0 and greater. In this mode, the target traces data accesses only and does not output PC information.

Cycle Accurate Enables ETM cycle-accurate mode. For ETMv1.x and ETMv2.x targets, the ETM normally outputs one trace packet every cycle when trace is enabled. Often many of these packets contain no useful information and can be discarded by the trace collection device. When this option is enabled on those targets, no packets will be discarded. For ETMv3.x targets, this option enables cycle-accurate mode in the ETM, which then outputs cycle count data. This allows the trace tools to determine the number of cycles spent executing each instruction, but requires extra space in the trace buffer. **Note**: For ETMv1.x and ETMv2.x targets, the ETM continues to output trace packets even when the processor is stopped at a breakpoint. Therefore it is generally not a good idea to enable **Cycle Accurate** mode if you will be hitting breakpoints while collecting trace data from an ETMv1.x or ETMv2.x target.
Break On Trigger Enables halting of the target processor when the trigger event occurs. There is some slip between when the trigger occurs and where the target halts. For more information about configuring a trigger event, see the documentation about configuring trace collection in the *MULTI: Debugging* book. This option does not support external triggers.
Trace Coprocessor Registers Enables tracing of values read from and written to coprocessor registers.
Filter Trace of Coprocessor Registers Enables ETM trace filtering of coprocessor register accesses. For more information about trace filtering, see the documentation about configuring trace collection in the *MULTI: Debugging* book. This option is only available when the **Trace Coprocessor Registers** option is enabled.
Half Rate Clocking Enables ETM half rate clocking mode. In this mode trace packets are output on both the rising and falling edges of the trace clock. This allows the trace clock to run at half the speed of the core clock. **Note**: Some targets do not support half rate clocking and some targets only support half rate clocking. If your target only supports one clocking mode, the **Half Rate Clocking** option is disabled and the appropriate clocking mode is selected automatically. **Note**: This option is not available with ETMv3.x targets because trace data is always output on both clock edges with ETMv3.x.
Timestamps Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI **Profile** window, PathAnalyzer, and EventAnalyzer.

Overflow Threshold Some ETM targets can stall the CPU or suppress data trace to prevent a FIFO overflow when the ETM FIFO is close to full. This option specifies the number of bytes remaining in the FIFO when that action will be done. Setting this option to `0` disables overflow prevention. When using this option with **Prevent Overflow by Stalling CPU**, setting a value greater than or equal to the size of your FIFO may cause errors when running or single-stepping (because the CPU is constantly stalled). FIFO sizes are implementation defined, but are often around 16 to 32 bytes. For more information, see the **Prevent Overflow by** option.
Prevent Overflow by Enables the ETM `FIFOFull` mechanism. The amount of data output by the ETM varies depending on the code being executed and the trace configuration. Code with a large number of indirect branches and data accesses (if data trace is enabled) may generate so much data that the ETM FIFO overflows. Trace data is lost when this happens. If this option is enabled, the ETM attempts to prevent FIFO overflows by using the selected method: • **Stalling CPU** — Attempts to stall the target processor when the FIFO is close to overflowing. This slows execution of code on the target, but can be very helpful in reducing the number of gaps in the trace data. • **Suppressing Data Trace** — Suppresses data trace when the FIFO is close to overflowing. This method is only available with ETMv3.x targets. Suppressing data trace is less effective at preventing the FIFO from overflowing than stalling the CPU, but has no impact on the speed of execution. To configure the overflow threshold, see the **Overflow Threshold** option. For more information, see the documentation about incomplete trace data in the *MULTI: Debugging* book. **Note**: The ETM `FIFOFull` mechanism is not supported by some targets.
Use Embedded Trace Buffer Enables use of the Embedded Trace Buffer (ETB). This option may only be changed if you use a SuperTrace Probe to connect to a target that has an ETB. If you use a Green Hills Probe to connect to a target with an ETB, the ETB is the only method available for collecting trace.
ASIC Control Provides a value for the optional ETM ASIC Control register. This register is implemented by some ASICs and allows configuration of ASIC-specific features.
CoreSight Source ID The CoreSight trace source ID of the ETM. Each CoreSight trace source on a system must have a unique ID between `0x1` and `0x6f`.

Infer Branch Target

Controls the circumstances in which the target emits a branch's target address instead of requiring the decompressor to infer the address. Trace data is more compact when the decompressor infers more branch targets, while the decompressor is more resilient to errors in the collected trace data when fewer branches must be inferred. There are three choices:

- `Never` — The PTM emits the target address for all branches, and the decompressor never needs to infer the target.
- `Direct Branches` — The PTM does not emit the target address for direct branches; the decompressor must infer them instead.
- `Direct Branches and Return Addresses` — The PTM does not emit the target address for direct branches or for some indirect branches that return from a function; the decompressor must infer them instead. Use this option if you are uncertain which option is most appropriate.

Trace Enable (core n)

Enables or disables trace collection on an individual core in the system. This option determines the cores for which trace data is collected when **Enable Trace** is turned on.

ColdFire Target-Specific Options

Each option is described in the following table.

Cycle Accurate

Enables cycle-accurate mode. Often many trace packets contain no useful information and can be discarded by the trace collection device. When this option is enabled, no packets will be discarded. This allows the trace tools to determine the number of cycles spent executing each instruction, but requires extra space in the trace buffer.

Data Capture

Selects the data access information that will be traced:

- **No Data Values (PC Trace Only)**
- **Reads Only**
- **Writes Only**
- **Reads and Writes**

INTEGRITY Interrupts at Address

Specifies the address that the interrupt vectors are copied to. For most ColdFire cores, this option should be set to `0x0`.

INTEGRITY Interrupts at Other Location Enable this option if you are tracing INTEGRITY and the BSP copies the interrupt handlers to a different address than the `.vector` section is mapped to.
Timestamps Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI **Profile** window, PathAnalyzer, and EventAnalyzer.

Nexus e200 Target-Specific Options

Each option is described in the following table.

MDO Data Port Width Selects the number of pins used to output trace data. Some targets support multiple MDO data port widths. Using a larger port width reduces the chance of a processor stall and/or FIFO overflow. We recommend using the largest port width that your target supports. If you specify an unsupported port width, you may get invalid trace data.
Trace Clock Multiplier Specifies the divisor for the trace clock. Not all targets support all divisors listed. In general, a faster trace collect (smaller divisor) allows for more data to be collected and results in fewer FIFO overflows. If you specify a divisor that your target does not support, you may get invalid trace data.
Stall Processor to Avoid Overflows Enables processor stalling to prevent FIFO overflows. The amount of trace data output by the target varies depending on the code being executed and the trace configuration. Code with a large number of indirect branches and data accesses (if data trace is enabled) may generate so much data that the FIFO overflows. Trace data is lost when this happens. If this option is enabled, the processor stalls when the FIFO starts to fill. This is very effective at preventing FIFO overflows, but does not prevent all overflows. For more information, see the documentation about incomplete trace data in the *MULTI: Debugging* book. **Note**: OS-awareness in TimeMachine requires specific parts of OS execution to be reconstructed. If kernel trace data is lost due to FIFO overflows, MULTI may discard trace data for some tasks. In this case, enabling **Stall Processor to Avoid Overflows** may improve OS-awareness.

Use Branch History Messages Enables branch history messaging. Most Nexus targets are capable of generating 2 different types of Nexus program trace messages. Traditional Nexus program trace messages are generated each time a branch is taken (direct or indirect). Branch history messaging only generates a message when 31 direct branch instructions have been executed (taken or not) or when an indirect branch is taken. Advantages of Branch History Messages: • More compact trace data 　◦ Less chance of overflow 　◦ Better utilization of trace buffer space Disadvantages of Branch History Messages: • More likely to have uncorrelated data trace • More data is lost as a result of an overflow 　◦ Less likely that the tools will be able to reconstruct missing instructions • Less precise filtering • Less accurate timestamps • Less precise reporting of trigger locations **Note**: ARM mode instructions on Nexus ARM targets (MAC71xx) are always traced with branch history messages.
Timestamps This option is only supported with SuperTrace Probe v3. Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI **Profile** window, PathAnalyzer, and EventAnalyzer.

Power Architecture QorIQ Nexus Target-Specific Options

Each option is described in the following table.

Branch and link correlation messages Enables trace messages every time a branch and link occurs. When these messages are enabled, timestamps are more precise, but trace data is less efficient.
CoreNet Trace Enables trace output from the CoreNet peripheral. This trace output appears in the trace list, but is not processed in any other way.

Data Path Trace Enables trace output from the Data Path peripheral. This trace output appears in the trace list, but is not processed in any other way.
DDR Trace Enables trace output from the DDR peripheral. This trace output appears in the trace list, but is not processed in any other way.
Enable/Disable program trace with MSR[PMM] When used in conjunction with a mechanism in target software that sets and clears the `MSR[PMM]` bit, this option aids in determining which sections of code are traced. The `MSR[PMM]` bit is the performance monitor mark bit, used for Nexus to provide execution context filtering.
Lite Trace Available for e6500 targets only. When enabled, the processor uses a link register stack optimization to double trace bandwidth efficiency, reducing the risk of processor stalls (if enabled) and FIFO overflows.
OCeaN Trace Enables trace output from the OCeaN peripheral. This trace output appears in the trace list, but is not processed in any other way.
Stall Processor to Avoid Overflows Enables processor stalling to prevent FIFO overflows. The amount of trace data output by the target varies depending on the code being executed and the trace configuration. Code with a large number of indirect branches and data accesses (if data trace is enabled) may generate so much data that the FIFO overflows. Trace data is lost when this happens. If this option is enabled, the processor stalls when the FIFO starts to fill. This is very effective at preventing FIFO overflows, but does not prevent all overflows. For more information, see the documentation about incomplete trace data in the *MULTI: Debugging* book. **Note**: OS-awareness in TimeMachine requires specific parts of OS execution to be reconstructed. If kernel trace data is lost due to FIFO overflows, MULTI may discard trace data for some tasks. In this case, enabling **Stall Processor to Avoid Overflows** may improve OS-awareness.
Stall threshold When **Stall Processor to Avoid Overflows** is enabled, indicates a fraction of trace message queue capacity at which the processor should stall. Smaller values result in more frequent stalls, but fewer FIFO overflows.

Timestamps Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI **Profile** window, PathAnalyzer, and EventAnalyzer. QorIQ targets buffer Nexus trace data before sending it out over high-speed serial trace (HSST). The buffer creates a variation in the time delta between when the packet is emitted by the core and when it actually sends over HSST. The size of the delta is inversely proportional to the HSST bandwidth, and varies depending on how full the buffer is. At minimum bandwidth (1 Aurora lane at 2.5 Gpbs), the maximum error is approximately 65 us.
Trace core *n* Enables or disables trace collection on an individual core in the system. This option determines the cores for which trace data is collected when **Enable Trace** is turned on.

PowerPC 405, 440, and 460 Target-Specific Options

Each option is described in the following table.

Timestamps Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI **Profile** window, PathAnalyzer, and EventAnalyzer.
Cycle Accurate Enables IBM cycle-accurate mode. Often many trace packets contain no useful information and can be discarded by the trace collection device. When this option is enabled, no packets will be discarded. This allows the trace tools to determine the number of cycles spent executing each instruction, but requires extra space in the trace buffer.

Chapter 5

Probe Command Reference

Contents

This chapter documents the commands available to the **mpserv** debug server that supports Green Hills Probe and SuperTrace Probe target connections. The commands are divided into the following sections:

- "Green Hills Debug Probe Commands" on page 174 — Unless otherwise noted, these commands can be entered:
  - In the MULTI Debugger target pane.
  - In the MULTI Debugger command pane or **Command** pane, if prefixed with the **target** command.
  - In a telnet or serial terminal window. For information about the terminal window prompt, see "Green Hills Probe and SuperTrace Probe Terminal Prompts" on page 224.

In addition to these commands, **mpserv** also supports the Green Hills debug server scripting language, which is described in "The Green Hills Debug Server Scripting Language" on page 228.

Green Hills Debug Probe Commands

In addition to the commands accepted by all Green Hills debug servers (see "Generic Debug Server Commands" on page 218 for a complete list of these commands), the Green Hills Debug Probes also accepts additional commands, which are described in the subsequent listed sections. These additional commands are grouped into the following nine command action categories:

- "Configuration Commands" on page 175
- "Front Panel I/O Pin Commands" on page 179
- "Group Commands" on page 181
- "JTAG and SWD Commands" on page 183
- "System Commands" on page 187
- "Target Commands" on page 190
  - "Cache, Memory, and TLB Commands" on page 191
  - "Run Control Commands" on page 202
  - "Other Commands" on page 207
- "Test Commands" on page 212

Note

For an alphabetized list of all of the Green Hills Debug Probe commands, see the heading **commands** in the index.

The commands listed in these sections are supported by all Green Hills Debug Probes unless otherwise noted. The following abbreviations are used to indicate commands that do not apply to all three devices:

- **GHP** — Green Hills Probe
- **STP** — SuperTrace Probe

Unless otherwise noted, the Green Hills Debug Probe commands described in the sections listed above can be entered:

- In the MULTI Debugger target pane.
- In the MULTI Debugger command pane or **Command** pane, if prefixed with the **target** command.
- In a telnet or serial terminal window. For information about the terminal window prompt, see "Green Hills Probe and SuperTrace Probe Terminal Prompts" on page 224.

For information about how to specify numbers when using the commands listed in the following sections, see "Specifying Numbers with Green Hills Debug Probe Commands" on page 215.

Configuration Commands

de
(Not available from **mpserv**.)
Detects whether the currently selected core is running in big endian or little endian mode. If detection is successful, the endianness option is changed accordingly, but the changes are not stored to non-volatile memory. To save the change to non-volatile memory, use the **save** command. This change stays in effect until the next reboot.

detect [-only *option1* [*option2...*] | **-skip** *option1* [*option2...*]] (Not available from **mpserv**.) Automatically detects several options, including **adapter**, **logic_high**, **target** and **endianness**. Additional options might be detected depending on the target. To make these settings persist when rebooting the probe, enter the **save** command after detect has finished. To detect the **target** option, the probe attempts to read the JTAG IDCODE or PVR registers from each device in the scan chain, and use those values to identify each device in the chain. A matching **target** type is selected for each identified device, and unidentified devices are grouped together and configured as generic JTAG devices. Some devices either have a faulty identification register, or have no identification register, in which case they are configured as generic JTAG devices (see "Specifying Your Target" on page 76). If detection is successful, this command displays a table of devices found. If no devices are found, the command displays an error. Successful detection indicates that low-level debug connectivity is functional, and an error typically means that there is a low-level connectivity problem such as no power, an incorrect **logic_high** value, or a miswired debug port connection. **detect** may also fail to identify targets that are not fully JTAG compliant. Because multiple processors within the same family sometimes have the same PVR or IDCODE, **detect** might not set the target to the exact processor type connected. For example, MPC8266 targets will be detected as MPC8260 because the two processors share an identical identification register value. **Note:** For PowerPC 750 and PowerPC 7445/7455 targets, the **target_rev** option must be set to `auto` before using **detect**. To detect only a single option, issue the command: `detect -only` *`optionname`* Some options are not detectable or are not supported as an argument to **-only**. In some cases, additional related options may be detected. To skip detection of a particular option, issue the command (the only options that may be skipped are **adapter**, **logic_high**, **power_detect**, and **use_gtl**): `detect -skip` *`optionname`* Both **-only** and **-skip** may be repeated to detect multiple options, or to skip multiple options. To detect or skip an option on only a single core, prefix the core number. For example: `detect -only 1:endianness` detects only the **endianness** option on core 1.	**Not For ARC Targets**

dlh Detects the voltage on the power sense pin of the adapter, if available. If no power is detected, no changes will be made. If a voltage is detected within the supported range, the **logic_high** setting will be adjusted (to the nearest 0.1 volt), but the change will not be stored to non-volatile memory. Use the **save** command to store the setting in non-volatile memory.	**Not for STP with Legacy Kits**
restore Restores the current configuration settings from non-volatile memory.	
save Saves the current configuration settings to non-volatile memory.	

set [*option* [*value* | default]]
 	
Sets configuration options by name, or displays current option settings.
 	
If no parameters are given, this command displays the values of all options that are not currently hidden. Some options are hidden under some circumstances. For example, on ARM targets, setting the **use_rtck** option to something other than `off` causes the **clock** option to become hidden. Do not change the value of a hidden option.
 	
If only an *option* is specified, the value of that option is displayed.
 	
If an *option* and a *value* are specified, the option is set to the specified value and saved to non-volatile memory. This command saves all non-volatile options, including those that have been previously modified with **tset**.
 	
If an *option* and the **default** argument are specified, the option is reset to its factory default setting.
 	
Examples:
 	
`>set`
 	
Displays all configuration options.
 	
`>set ip`
 	
Shows the value of the IP address.
 	
`>set netmask 255.255.255.0`
 	
Sets the netmask to `255.255.255.0` and stores this setting to non-volatile memory.
 	
`>set rst_settle default`
 	
Resets the **rst_settle** option to its factory default setting.
 	
Green Hills Probe users can use the **setup** command from a telnet or serial terminal window to be guided interactively through setting common options (rather than having to set each option with the **set** command).	

setup [*group* | reset]
 	
(Serial or terminal window only. Not available in the MULTI Debugger **Target** and **Command** panes.
 	
Interactively guides you through setting the probe configuration options.
 	
If no parameters are given, the interactive utility provides prompts to help you set the most common options.
 	
If you specify a *group*, the utility prompts you to set the options in that group. The configuration groups are:
 	
• `net` (network)
• `trg` (target-specific)
 	
If you pass the **reset** argument, all of the configuration settings will revert to their factory defaults.
 	
Examples:
 	
`>setup`
 	
Launches an interactive utility to configure the default group of common configuration options. This is the easiest way to configure a new probe out of the box.
 	
`>setup reset`
 	
Resets all configuration options to their factory defaults.
 	
This **setup** command functions differently than the **mpserv setup** command described in “Generic Debug Server Commands” on page 218.

tracereg *reg_name* **tracereg** *reg_name* = *reg_value* Reads or writes trace registers to configure trace capture on the probe. *`reg_name`* is the name of the register to read or write. *`reg_value`* is the value to write to the trace register. *`reg_name = reg_value`* instructs the probe to set the contents of the register *`reg_name`* to the value specified by *`reg_value`*.

tset [*option* [*value*]]

Temporarily sets configuration options by name, or displays current option settings.

If no parameters are given, all options and their current values are displayed.

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

If an *option* and a *value* are specified, the option is set to the specified value, but is not stored to non-volatile memory. The option setting stays in effect only until the next reboot, unless you use the **save** command to save all configuration changes to non-volatile memory.

Example:

```
>tset clock 100 kHz
```

Sets the debug level to 42 until the next reboot.

Front Panel I/O Pin Commands

gpin

This command is deprecated. Use **iop** instead.

gpincfg [*pin_config*]

This command is deprecated. Use **iop** instead.

iop [*binval*]

iop [*pin* [*.field* [=*value*]]...]

Sets or displays the present state of the front panel I/O pins. There are three basic formats for this command:

If no arguments are given, **iop** displays the current mode and value of each pin.

If a *binval* argument is given, **iop** sets the pins to the specified values. The *binval* argument must be three bits, where each bit corresponds to the setting of a pin and can be either `1` or `0`. Pin 1 is set to the leftmost bit, pin 2 to the middle bit, and pin 3 to the rightmost bit. The correspondence of the bits to pins can change if more front panel pins are added in the future. The number of bits required in the *binval* argument corresponds to the number of GPIO pins.

If one or more `[pin][.field][=value]` arguments are specified, the value of one or more pins or pin fields are displayed or modified, where:

- ***pin*** — Specifies the name of the pin to view or modify, in the form `pin#` or `p#`. For example, to specify pin 1, you can use either `pin1` or `p1` as the *pin* argument. If a *pin* argument is specified without any other arguments, the current value of the pin is displayed. If a *pin* argument is specified with only a ***=value*** argument, the output value of the pin is set to *value* (which must be `1` or `0` in this case).
- ***.field*** — Indicates a specific pin field to modify or display. (You must use a ***pin*** argument also if you specify a *field*. If a *pin.field* is specified, but no ***=value*** argument is given, the current setting of the pin field is displayed. If ***=value*** is specified with the *pin.field* argument, the indicated pin field is set to the new value. Valid fields are listed below, with the possible values that can be assigned to each.
  - **value** — The pin's output value, which can be set to either `0` or `1`.
  - **trigger** or **t** — (For pins configured as input pins only; currently not supported) The input pin's method for triggering the probe, which can be `n` (never), `f` (falling edge), or `r` (rising edge).
  - **mode** or **m** — The pin's I/O mode, which can be set to either `o` (a lowercase letter O, for output mode) or `i` (for input mode).
  - **drive** or **d** — (For pins configured as output pins only) The pin's driving mode, which can be set to either `a` (for active) or `g` (for open drain).

  If no field is specified, but an ***=value*** option is given, the pin's output value is modified.

Example 5.1. Using IOP Commands

The following are examples of **iop** usage:

`>iop 110` — Sets pins 1 and 2 to 1, and sets pin 3 to 0.

`>iop pin2` — Displays all the current settings for pin 2.

`>iop p2=0` — Sets pin 2 to 0.

`>iop pin1.mode=o` — Sets pin 1 to be an output pin.

`>iop p1=1 p2.m=o p3.m=o` — Sets pin 1 to 1, and sets pins 2 and 3 to be output pins.

Group Commands

g [*group_id*] Sets the current group (for group-based operations) to the group with ID *group_id*. A group ID is returned by the **ga** command when a new group is added. To see a list of groups and their IDs, use the **gl** command. If *group_id* is not specified, **g** displays the current group for group-based operations.
ga [*type*] *cores* Adds a new synchronous run-control group and returns a group ID for the new group. *type* optionally specifies the type of run-control group. By default, *type* is set to `generic`. Only set this if directed to do so by Green Hills Support. *cores* specifies one or more core IDs for the new group. For information about specifying *cores*, see **-force_coreid** in “Options for Custom Connection Methods” on page 49. Example: `>ga 0..2,4` — adds a group of type `generic` using the core IDs 0, 1, 2, and 4.

gc [*cores* | *ggroup_id*] Synchronously continues one or more cores. True synchronous control of cores may not be available on all targets. If not available, cores will be resumed sequentially. A group ID is returned by the **ga** command when a new group is added. To see a list of groups and their IDs, use the **gl** command. If *cores* is specified, this command synchronously continues a set of cores. For information about specifying *cores*, see **-force_coreid** in “Options for Custom Connection Methods” on page 49. If `ggroup_id` is specified, this command continues the synchronous run-control group specified by *group_id*. When used with no arguments, **gc** synchronously continues the current run-control group. Examples: `>gc g1` — continues run-control group 1. `>gc 1` — continues core 1. `>gc 0,3..6` — continues cores 0, 3, 4, 5 and 6.
gd *group_id* | *type cores* Deletes the specified synchronous run-control group. If *group_id* is specified, this command deletes group with this ID. For a list of group IDs, use the **gl** command. If *type* is specified, this command deletes any run-control groups in the system of the same *type* that overlap with the provided *cores*. *cores* is the core list to use with *type*. For information about specifying *cores*, see **-force_coreid** in “Options for Custom Connection Methods” on page 49.

gh [*cores* | g*group_id*]

Synchronously halts one or more cores. True synchronous control of cores may not be available on all targets. If not available, cores will be halted sequentially.

A group ID is returned by the **ga** command when a new group is added. To see a list of groups and their IDs, use the **gl** command.

If *cores* is specified, this command synchronously halts a set of cores. For information about specifying *cores*, see **-force_coreid** in "Options for Custom Connection Methods" on page 49.

If `ggroup_id` is specified, this command halts the synchronous run-control group specified by *group_id*.

When used without arguments, this command synchronously halts the current run-control group.

Examples:

`>gh g1` — halts run-control group 1.

`>gh 1` — halts core 1.

`>gh 0,3..6` — halts cores 0, 3 ,4, 5 and 6.

gl

Displays all configured run-control groups, listing the group number, group type and cores in this group.

JTAG and SWD Commands

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

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* of *data* are scanned in. 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 contents of the data register are read as the data is shifted in and is printed in hexadecimal format.

The optional arguments `b` and `l` specify that data scanning starts with the most or least significant bit.

This command prints the bits scanned out from the target in hexadecimal.

ji *instruction* [*rti*] **JTAG only**

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. 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.)

The value printed and returned from this command is the contents of the instruction register as shifted out during the `Shift-IR` state.

Example:

```
>ji 0xff
```

Shifts in bypass instruction for PowerPC 8260 and 8240 targets.

jp [on | off] **JTAG only**

jp *pin_name* mode=*pin_mode*

jp *pin_name* [mode=*pin_mode*] *level*

Displays and controls JTAG pins. If no arguments are specified, all pins and their current status are displayed. Specifying `on` or `off` enables or disables (tristates) all interface pins. The **User** button emulates **jp on** and **jp off**. When the interface pins are disabled, you can safely connect a different target board to the probe.

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

Specifying a *pin_mode*, for adapters that support it, changes the drive mode for the pin. Valid modes are:

- `1|0` — active drive
- `Z|0` — open-collector
- `1|Z` — open-emitter
- `Z|Z` — disabled
- `LOW` — always low
- `HIGH` — always high
- `default` — reverts to the pin's default drive mode

For example:

```
>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.

```
>jp NTRST mode=Z|0 1
```

Changes the drive mode of `NTRST` to be open-collector.

```
>jp HRST mode=1|0 1
```

Changes the drive mode of `HRST` to be active and pulls it high.

jr Resets the JTAG TAP or SWD controller. We recommend that you do not use this command on XScale targets, because the probe will lose control of the target, due to the way the XScale debug architecture works.	**Not for BDM targets**
swd *sequence* Scans an arbitrary SWD sequence *sequence*, made up of the characters `0`, `1`, and `i`. For each `0` or `1`, that value is transmitted. For each `i`, the probe reads one bit.	**SWD only**
swdread ap | dp *addr* Reads an SWD register at the address *addr*. If you specify `ap`, the probe reads an `AP` register. If you specify `dp`, the probe reads a `DP` register.	**SWD only**
swdwrite ap | dp *addr value* Writes the value *value* to an SWD register at the address *addr*. If you specify `ap`, the probe writes an `AP` register. If you specify `dp`, the probe writes a `DP` register.	**SWD only**

System Commands

alias [*alias*] [*expansion*]
 	
Sets command alias or displays current aliases.
 	
If no parameters are given, all aliases and their current values are listed.
 	
If only an alias name is specified, the expansion value of the alias is displayed.
 	
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 dumps the memory at `0xff004020` when run.

exit (Not available in the MULTI Debugger **Target** and **Command** panes.) Exits a telnet session. When run from the serial port, this command deactivates the serial console until another command is entered. The serial terminal no longer prints the target status updates generated by the checker thread. You cannot exit an RS232 serial terminal session.

help [*group* | *topic* [*subtopic*]]
 	
Displays help messages.
 	
If no arguments are specified, a brief overview of the most commonly used commands are displayed, and lists which *groups* are available.
 	
If a *group* is specified, all help topics within that group are displayed. If a *topic* is specified, detailed help on that topic (and optional *subtopic*) is displayed.
 	
This **help** command is specific to the Green Hills Debug Probes and differs from a **help** command issued through **mpserv**. For information about the **mpserv help** command, see “Generic Debug Server Commands” on page 218.
 	
Examples:
 	
`>help bs`
 	
Displays help on the **breakpoint set** command.
 	
`>help set ip`
 	
Displays help on setting the IP address.

info

Displays basic Green Hills Probe or SuperTrace Probe information.

print [*argument*]...

Prints all arguments separated by spaces and followed by a newline to the current terminal.

Example:

```
>print "Hello, World!" "Hello, World!"
```

pterminal [[*port_name*]|[-baud *baud_rate*]|[-parity *parity*]|[-databits *data_bits*]|[-stopbits *stop_bits*]|[-flowcontrol *flow_control*]]

Opens a terminal session to a port on the probe. To exit the session, press the following three keys, in order

1. Enter
2. Tilde (~)
3. Period (.)

To list available ports, run this command without any arguments. `serial` (the RS-232 port on the back of the probe) is always listed, but is only available when **serial_terminal** is set to `off`. Some targets list an on-chip debug port.

The arguments for this command are:

- *`port_name`* — The name of the port to open.
- *`baud_rate`* — The baud rate. Valid values are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200. The default is 9600.
- *`parity`* — The number and type of parity bits. Valid values are `none|even|odd`. The default is `none`.
- *`data_bits`* — The number of data bits in each word. Valid values are 5, 6, 7, and 8. The default value is 8.
- *`stop_bits`* — The number of stop bits. Valid values are 1 or 2. The default is 1.
- *`flow_control`* — Type of flow control. Valid values are `none` and `xonxoff`. The default is `none`.

Examples:

```
pterminal serial -baud 115200
```

Opens the probe's serial port at a baud rate of 115200.

reboot

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

Reboots the probe.

You must reboot your probe after changing certain configuration options, such as `ip`, `netmask`, `gateway`, or `dhcp`.

support

Displays information that is useful to Green Hills support.

tbtemp **STPv1 only**

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

Displays the temperature inside the SuperTrace Probe. Green Hills support may ask for the output of this command to help diagnose problems.

w

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

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

- The connection type, displayed as one of the following:
  - `Ethernet Debugger` — The MULTI Debugger is connected to the probe through an Ethernet connection
  - `Telnet` — A telnet session is connected to the probe console
  - `Serial Console` — A serial terminal is connected to the probe console
  - `USB Debugger` — The MULTI Debugger is connected to the probe through a USB connection
- (Ethernet connections only) The IP address of the client
- The number of seconds since the user last used the connection

xswitch [-defer | -nosave] [-reset] [[+|-]*$switch*]

Sets or modifies the xswitch *$switch*. Do not use this option unless instructed to do so by Green Hills support.

If you do not specify any arguments for **xswitch**, it displays all modified xswitches.

When you update your probe's firmware, it resets all xswitches to their default values.

- `-defer` — Wait to modify the xswitch until the probe reboots.
- `-nosave` — Do not store the change to non-volatile memory. The change will revert when the probe reboots.
- `-reset` — Restore *$switch* to its default value. If you do not specify *$switch*, this option resets all xswitches.
- *`$switch`* — The xswitch you want to modify. If you precede this argument with `+` or `-`, it is set or cleared respectively. Otherwise, the current value of the xswitch is displayed.

For example, to set the `example.switch` xswitch immediately, but have that change revert the next time you reboot the probe, type the following command:

```
xswitch -nosave +example.switch
```

Target Commands

Address Suffixes

Many target commands require an address parameter. How an address is accessed can be modified with the following suffix characters:

`v` — Indicates the specified address is virtual.

`p` — Indicates the specified address is physical.

The `v` and `p` suffixes are mutually exclusive.

`r` — Indicates that a raw memory access should be performed, bypassing any caches even if they are valid or dirty. This suffix is ignored on targets that do not support raw memory access.

`i` — Indicates that an instruction is being written and any necessary steps should be taken (such as flushing the instruction cache) to ensure that the data written can be executed by the target. This suffix only applies to memory writes.

Examples:

`0x10000v` represents the virtual address `0x10000`.

`0x4abcdp` represents the physical address `0x4abcd`.

```
>md 0x4000r
```

Dump raw memory at `0x4000`.

```
>mw 0x700i 0x60000000
```

Writes `0x60000000` to address `0x700`, and invalidates any corresponding instruction cache entries.

Cache, Memory, and TLB Commands

clf [*cache*] *address*

Searches a cache for a valid entry corresponding to *address*. *cache* specifies the cache to be searched and can be any one of the following:

- **i** — Specifies the instruction cache.
- **d** — Specifies the data cache.
- **1** — Specifies the unified L1 cache.
- **2** — Specifies the L2 cache.
- **3** — Specifies the L3 cache.

If *cache* is not specified, all caches are searched.

If a valid entry is found, it is displayed. If a valid entry is not found, the set number(s) searched are displayed.

clop *op cache set* [*way*]

clop *op cache* address=*address*

Performs a cache operation on one or more cache lines. Not all operations are supported on all targets. It is possible that, for a given target, the set/way form of addressing is implemented while the address form is not and vice versa.

The *op* argument specifies the operation to perform, and can be any one of the following:

- **fill**
- **writeback**
- **invalidate**
- **flush**
- **init**
- **lock**
- **unlock**

The *cache* argument specifies the cache to operate on, and can be any one of the following:

- **i** — Specifies the instruction cache.
- **d** — Specifies the data cache.
- **1** — Specifies the unified L1 cache.
- **2** — Specifies the L2 cache.
- **3** — Specifies the L3 cache.

set specifies the range of sets on which to perform the operation. For a fully-associative cache, it specifies which lines to change. Use `*` to specify all sets.

way specifies the range of ways in the sets on which to perform the operation. The default is `0`. Use `*` to specify all ways.

address specifies the address to perform the operation on.

For example, to initialize the entire i-cache:

```
clop init i * *
```

To flush the data cache line containing address `0x100`, if it exists:

```
clop flush d address=0x100
```

clr [s*grouping*] *cache set*

Reads a range of cache sets.

The optional `sgrouping` argument specifies the number of bytes into which memory is grouped for display. If the `sgrouping` argument is not specified, memory is displayed in 4-byte groupings.

The *cache* argument specifies the cache to read, and can be any one of the following:

- **i** — Specifies the instruction cache.
- **d** — Specifies the data cache.
- **1** — Specifies the unified L1 cache.
- **2** — Specifies the L2 cache.
- **3** — Specifies the L3 cache.

The *set* argument specifies the range of sets to read. For a fully associative cache, *set* specifies which lines to read.

Examples:

```
>clr i 3
```

Reads the third set of the instruction cache.

```
>clr s2 d 6
```

Reads the sixth set of the data cache and displays the data grouped in 16-bit units.

clsa *cache set* [*way*] *address*

Sets the address on a single cache line. The least significant bits of the address for each cache line are determined by the set to which that line belongs, and are not affected by this command.

The *cache* argument specifies which cache to change and can be any one of the following:

- **i** — Specifies the instruction cache.
- **d** — Specifies the data cache.
- **1** — Specifies the unified L1 cache.
- **2** — Specifies the L2 cache.
- **3** — Specifies the L3 cache.

The *set* argument specifies the range of sets to change. For a fully associative cache, *set* specifies which lines to change.

The optional ***way*** argument specifies the range of ways in the sets to change. If not specified, this value defaults to 0.

The *address* argument specifies the address to set. This address may be a physical address or a virtual address, depending upon the type of target processor.

Examples:

```
>clsa i 3 0x00020030
```

Sets the address of icache set 3 way 0 to `0x00020030`.

```
>clsa d 6 2 0x560
```

Sets the address of dcache set 6 way 2 to `0x560`.

clst *cache set* [*way*] *tag*

Sets the tag bits on one or more cache lines.

The *cache* argument specifies which cache to change and can be any one of the following:

- **i** — Specifies the instruction cache.
- **d** — Specifies the data cache.
- **1** — Specifies the unified L1 cache.
- **2** — Specifies the L2 cache.
- **3** — Specifies the L3 cache.

The *set* argument specifies the range of sets to change. For a fully associative cache, *set* specifies which lines to change.

The optional ***way*** argument specifies the range of ways in the sets to change. If not specified, this value defaults to 0.

The *tag* argument specifies the tag bits to set. The tag bits can be specified either as a single number or as a sequence of `key=value` pairs (see “Bitfield Types” on page 217 for more information). To see which fields are supported by your target, perform a **clr** command. Unspecified fields default to `0`. (For tables listing CPU-specific bit information, see Appendix C, “CPU-Specific Bit Tables” on page 299.)

Examples:

```
>clst i 3 Valid=0xf,LRF=1
```

Sets the tag of icache line 3 way 0. The unspecified `L` bit will be `0`.

```
>clst d 6 2 1
```

Sets the tag of dcache line 6 way 2 to `1`.

```
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. To clear memory access permissions, use the **mca** command. All regions with undefined permissions allow both read and write access.

permission specifies any combination of `r`, `w`, `f`, `0`, and `1` to allow reading, writing, flash breakpoints and two features specific to the architecture, respectively. If you specify `-` for this argument, all access to the memory region is disallowed. If you do not specify the argument, `rw` is used by default. The features `0` and `1` have different meanings depending on your target. Support for flash breakpoints varies by target. Check the usage notes for more information. Regions with `*` in the permission field of the memory range list cannot be modified by the user due to hardware limitations.

The optional *match* argument 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 *access_list* argument specifies a comma-separated list of allowable access sizes in bits. An asterisk (`*`) indicates that all access sizes are permissible, while the letter `b` allows block accesses (which can use any access size).

If no arguments are specified, **ma** displays the current memory access permissions.

Examples:

```
>ma
```

Displays current debug memory access permissions.

```
>ma 0xz 0xh -
```

Prevents any access.

```
>ma 0x80000000 0x8001ffff rw
```

Specifies the memory range as regular RAM.

```
>ma 0xbf000c00 0xbf000cff rw 8
```

Specifies a device which has 8-bit registers.

```
>ma 0xbf000d00 0xbf000dff rw 0x0/0x3 16
```

Specifies a device that has 16-bit registers, but only at a 32-bit aligned address.

```
>ma 0xbf000e00 0xbf000eff rw 8,16
```

Prevents 32-bit accesses.

mat ***virtual_address***

Attempts to translate the given *virtual_address* to the physical address it corresponds to, given the current state of the processor.

The probe applies TLBs, page tables, or any other mechanism that the processor might use to define the mapping of the specified address.

If the probe is not able to perform the translation, the probe assumes a direct mapping, and returns the same address.

md [s*grouping*] *start_address* [*length*]

Displays bytes of target memory.

The optional `sgrouping` argument specifies the number of bytes into which memory is grouped for display. If this argument is not specified, memory is displayed in 4-byte groupings.

The *start_address* argument specifies the address to start reading memory.

The optional *length* argument specifies the number of bytes to display. If this argument is not specified, 64 bytes are displayed.

Examples:

```
>md 0xa0000000
```

Reads 64 bytes starting at `0xa0000000`.

```
>md 0x1000 200
```

Reads 200 bytes starting at `0x1000`.

```
>md s1 0xfff00000
```

Reads 64 bytes starting at `0xfff00000`, and displays the result in single bytes.

mf *start_address length pattern*

(Not available from **mpserv**.)

Fills the block of target memory beginning at *start_address* and continuing for *length* bytes with the 4-byte pattern, *pattern*.

Examples:

```
>mf 0xa0000000 0x500 0x401e00e0
```

Fills `0x500` bytes starting at `0xa0000000` with the 4-byte pattern `0x401e00e0`.

```
>mf 0x0 0x8000 0x21212121
```

Fills `0x8000` bytes starting at `0x00000000` with the 1-byte pattern `0x21`.

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, you must verify 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, you must verify that the *start_address* is aligned in a compatible manner.

Examples:

```
>mw 0xa0000000 0x23
```

Performs a 4-byte memory write of `0x00000023` to `0xa0000000`.

```
>mw 1 0x1000 0xfd
```

Performs a 1-byte memory write of `0xfd` to `0x1000`.

regdcr *reg_num* [=*value*] | *reg_num value*

(For PowerPC 4xx targets only.)

Accesses a PowerPC 4xx DCR register. If *value* is specified, **regdcr** writes the value to the DCR register specified by *reg_num*. If no value is specified, **regdcr** prints the value of the specified DCR register.

Examples:

```
>regdcr 0x12
```

Prints the value of the first peripheral bank configuration register (EBC0_B0CR).

```
>regdcr 0x12 0x0
```

Writes the value `0x0` to EBC0_B0CR.

```
>regdcr 0x12=0x0
```

Writes the value `0x0` to EBC0_B0CR.

rr [*name* | *group*]...

Reads target registers.

If no parameters are specified, the most commonly accessed registers are displayed. Alternatively, you can use any number of *name* and *group* arguments to specify registers, where:

- *name* specifies a register name, such as `r0` or `pc`. You can also use wildcards when specifying names:
  - `*` — matches zero or more characters
  - `?` — matches exactly one character

  All matching registers are displayed. If you specify an asterisk by itself (`rr *`), all registers and groups are displayed.
- *group* specifies a group of registers.

For example, on an ARM target:

```
rr cp15
```

Reads the system coprocessor (`cp15`).

For more information about special registers available for the **rr** command, see "Target-Specific Special Registers" on page 209.

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`.

For more information about special registers available for the **rw** command, see "Target-Specific Special Registers" on page 209.

tlbr [-valid] [i | d | ds | dl | t0 | t1 | l2 | lockdown] *entry*

Reads and prints out a range of entries or sets from the target's translation lookaside buffer (TLB), where:

- The optional **-valid** option only prints valid TLB entries (if an entry has a valid bit).
- The optional **i** or **d** option specifies the instruction TLB or data TLB. This argument is ignored on targets without separate instruction and data TLBs. On targets with separate instruction and data TLBs, the default TLB used if neither **i** or **d** is specified depends upon the specific implementation.
- The optional **ds** or **dl** option specifies the data store TLB and the data load TLB. (Cortex-A15)
- The optional **t0** or **t1** option specifies the target's TLB 0 or TLB 1 TLBs. This argument is ignored on targets without TLB 0 and TLB 1. **t0** and **t1** are used only for PowerPC 85xx and QorIQ targets, with **t1** being the default if no option is specified.
- The optional **l2** option specifies the `L2` TLB. (Cortex-A15)
- The optional **lockdown** option specifies Lockdown TLB. (ARM926)
- ***entry*** specifies the range of entries or sets to read, with `0` specifying the first entry or set. To specify a range from *x* to *y*, inclusive, use the format `x..y`. To read the entire TLB, use an asterisk (`*`).

Entries are printed for fully-associative or direct-mapped TLBs, while sets are printed for set-associative TLBs.

Examples:

```
>tlbr i 3
```

Prints the fourth TLB entry or set. On a PowerPC 440gx (fully-associative TLB), this command prints:

```
entry 3: vaddr=0x03000000 vtag=0x02700007 (!U0,!U1,!U2,!U3,V,!TS,SIZE=0x7,
TID=0x0,!W,!I,!M,!G,!E,!UX,!UW,!UR,SX,SW,SR) ->
         paddr=0x0:03000000 ptag=0x0 (ERPN=0x0)
```

```
>tlbr 3..8

idx w vaddr      U0U1U2U3VTsSizeTidWIMGEUxUwUrSxSwSr    paddr        Erpn
  3 0 0x03000000 --------V--   7  0-----------SxSwSr -> 0x0:03000000    0
  4 0 0x04000000 --------V--   7  0-----------SxSwSr -> 0x0:04000000    0
  5 0 0x05000000 --------V--   7  0-----------SxSwSr -> 0x0:05000000    0
  6 0 0x06000000 --------V--   7  0-----------SxSwSr -> 0x0:06000000    0
  7 0 0x07000000 --------V--   7  0-----------SxSwSr -> 0x0:07000000    0
  8 0 0xe0000000 --------V--   1  0-I-G---------SwSr -> 0x1:40000000    1
```

```
>tlbr *

idx w vaddr      U0U1U2U3VTsSizeTidWIMGEUxUwUrSxSwSr    paddr        Erpn
  3 0 0x03000000 --------V--   7  0-----------SxSwSr -> 0x0:03000000    0
  4 0 0x04000000 --------V--   7  0-----------SxSwSr -> 0x0:04000000    0
  5 0 0x05000000 --------V--   7  0-----------SxSwSr -> 0x0:05000000    0
  6 0 0x06000000 --------V--   7  0-----------SxSwSr -> 0x0:06000000    0
  7 0 0x07000000 --------V--   7  0-----------SxSwSr -> 0x0:07000000    0
  8 0 0xe0000000 --------V--   1  0-I-G---------SwSr -> 0x1:40000000    1
...
  63 0 0x00024000 --------VTs   2  5-----Ux--Ur------ -> 0x0:01fc0000    0
```

tlbw [i | d | t0 | t1 | lockdown] *entry* [*way*] *vaddr vtag paddr ptag*

Writes an entry in the target's translation lookaside buffer (TLB) with CPU-specific TLB settings and information, where:

- The optional **i** or **d** option specifies the instruction TLB or data TLB. This argument is ignored on targets without separate instruction and data TLBs. On targets with separated instruction and data TLBs, the default TLB used if neither **i** or **d** is specified depends upon the specific implementation.
- The optional **t0** or **t1** option specifies the target's TLB 0 or TLB 1 TLBs. This argument is ignored on targets without TLB 0 and TLB 1. **t0** and **t1** are used only for PowerPC 85xx and QorIQ targets, with **t1** being the default if no option is specified.
- The optional **lockdown** option specifies Lockdown TLB. (ARM926)
- The *entry* argument specifies the entry or range of entries in the TLB to be written.
- The optional *way* argument can be used to specify the way in the entry to be written, if applicable. Generally, this argument only needs to be specified for CPUs with set-associative TLBs. If this argument is not specified, it defaults to `0`.
- The *vaddr* argument specifies the virtual address to write to *entry*.
- The *vtag* argument specifies CPU-specific TLB information to write for *vaddr*'s entry.
- The *paddr* argument specifies the physical address to write to *entry*.
- The *ptag* argument specifies CPU-specific TLB information to write for *paddr*'s entry.

In the syntax above, *vtag* and *ptag* are bitfield types. The bits can be specified either as a single number or as a sequence of *`key=value`* pairs (see “Bitfield Types” on page 217 for more information). For CPU-specific tag information, see the tables in Appendix C, “CPU-Specific Bit Tables” on page 299.

Run Control Commands

bc *id* Clears the breakpoint with the ID number *id*.
bca Clears all breakpoints.
bl Lists all breakpoints in the same format as specified by the **bs** command (see the following table entry). For information about number formats in the list, see “Specifying Numbers with Green Hills Debug Probe Commands” on page 215 and “Address Suffixes” on page 190.

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. If none of these parameters (`r`, `w`, or `x`) are included, **bs** sets a software breakpoint. 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.

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

You can 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*.

Note that a mask of `0xffffffff` means that all bits are significant. Whereas a mask of `0xffffff00` means that 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.

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

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

Examples:

```
>bs x 0x200000
```

Sets a hardware breakpoint at `0x200000`.

```
>bs 0xa00000000
```

Sets a software breakpoint at `0xa0000000`.

```
>bs rw 0x1000/0xffffff00 2 d=0x1234 c=3
```

Sets a data hardware breakpoint 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
```

Sets a hardware breakpoint when a 32-bit value with the most significant bits equal to `0x1234` is written to `0x1000`.

cr pre | tap | post | full

Performs, on the current core, one or all of the phases that make up a typical reset sequence, where:

- **pre** — Prepares the core for reset.
- **tap** — Performs all steps required while the CPU reset line (for example, `nHRESET` or `nRESET`) is asserted. On JTAG targets, this performs the JTAG Test Access Port (TAP) reset, including toggling the TAP reset line (for example, `nTRST`). This step works only if **target_reset_pin** is set to `independent`, otherwise, it should be skipped.
- **post** — Performs any necessary post-reset initialization.
- **full** — Performs all three reset phases and toggles the CPU reset line as appropriate. This option assumes that the probe can control the reset line.

t *cores* [*command*]
 	
Changes the current core ID to *cores* or issues a detailed command for all specified cores.
 	
cores is a comma separated list of core groups. For information about specifying *cores*, see **-force_coreid** in “Options for Custom Connection Methods” on page 49.
 	
If no *command* is specified, **t** changes the current core to the core with the ID *cores* and all subsequent commands apply to the new core until another **t** command is executed. This form of the command is not available from within the MULTI Debugger.
 	
If a *command* is specified, the current core ID is not changed, and the command is issued for all cores listed in *cores*. In this case, *cores* can consist of a single core, a list of cores separated by commands, a range of cores, or an asterisk (`*`) to indicate all cores.
 	
Examples:
 	
`>t 0`
 	
Selects core 0, or the first core.
 	
`>t 9`
 	
Selects core 9, or the tenth core.
 	
`>t 1 th`
 	
Halts core 1, the second core, but does not change the default core setting to 1.
 	
`>t 0,3 ti`
 	
Prints the status for cores 0 and 3.
 	
`>t 3..5 tc`
 	
Resumes cores 3, 4, and 5.

tc Continues running the target from the current `PC` (program counter).
th Halts a running target.
ti Displays current target status information.
tl Lists all targets in the system and their status.

tr [-s] [d | r]

Resets the target system using JTAG reset pins.

If `d` is specified, or if neither `d` nor `r` is specified, the target remains in debug mode immediately after the reset. For targets that do not support this behavior, a halt request is sent immediately.

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

Passing the `-s` option causes **tr** to perform a soft reset that does not wiggle the JTAG reset lines. (The `-s` option is not supported for all targets.)

On a multi-core system, **tr** is only guaranteed to reset one core. Other cores can be reset or experience other side effects as a result of **tr**. This depends on how the cores are connected to the core being 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 might result in more than one instruction being executed. For instance, the branch delay slot can also be executed when single stepping a branch instruction on a MIPS target.

Other Commands

addkey *featurename key* Adds or changes a feature key on this device. Feature keys turn on probe functionality that is only available as an add-on or is still in beta. Feature keys are provided by Green Hills support.	
checkkey *featurename* Reports the current stored value of a feature key, and whether or not that key is valid. Feature keys are provided by Green Hills Support, and can be added with the **addkey** command.	
dp *filename* Programs a target device with the file specified by *filename*. Currently, the only supported target device is the Xilinx FPGA on ARC evaluation boards. In that case, the file should be a .xbf file provided by ARC. Note that there is no way for the debug probe to verify correct device programming.	**GHP only; available only in MPserv; ARC targets only**

prm status *logentries*

prm start [**-invalidate**] [*addr* | -] [*connstr* | -]

prm stop

prm subscribe *msgtype*[,...] | all | none

Controls the Probe Run Mode Proxy on the probe. This command can start, stop, get status, and subscribe to messages.

prm status – Gets the status of the Proxy. *logentries* specifies the maximum number of log entries to print with the status.

prm start – Starts the Probe Run Mode Proxy. If the Proxy is already running it will be stopped and restarted.

- `-invalidate` invalidates the current header so the Probe Agent can tell when the target has started. This should not be used if the kernel has already booted.
- *addr* specifies the address of the `.gipctarget` section. If none is specified and **mpserv** knows this address, that address is used by default.
- *connstr* specifies the connection string. It defaults to the connection string **mpserv** used to connect to the probe, or the probe's hostname.

prm stop — Stops the Probe Run Mode Proxy. `stop *` stops any and all instances.

prm subscribe — Subscribes to one or more message types or clear subscriptions. Setting the subscription list replaces any previous subscription list.

- *msgtype* is one of `error`, `info`, `debug`, or `signal`. The signal type shows any output generated by the interrupt script.
- `all` subscribes to all types.
- `none` clears all subscriptions.

Examples:

- `prm` Queries the Proxy status and displays recent log entries.
- `prm start -invalidate` When run from **mpserv**, uses the `.gipctarget` section address of the loaded program to start the Proxy, invalidating the header on the target in the process.
- `prm start 0x80001240` Starts the Proxy with base address `0x80001240`.
- `prm start - -` Starts the Proxy with the address from the link map and with no connection string (to disable automatic partnering).
- `prm subscribe info,error` Subscribes to both `info` and `error` messages from the Proxy.

syscalls [on | off] Enables or disables system calls. System calls are on by default. If no argument is specified, this command reports the current state of system calls. System calls for performing input and output on the host are implemented with a software breakpoint. The **syscalls** command can be used to disable and enable this breakpoint and therefore system call servicing by the host. When running code from ROM, it may not be possible to set the system call breakpoint and you may need to use **syscalls off** to disable it.	
time [-noprint] [-repeat *num*] [command] Executes the probe command *command* and outputs the amount of time it took to execute it. To run the command multiple times, pass the **-repeat** option, followed by the number of repetitions.	
trace_registers [on | off] Enables or disables support for trace registers. Should be used in combination with **-no_trace_registers** argument. For information, see "Options for Custom Connection Methods" on page 49.	**STP, ARM only; available only in MPserv**
trace_state Prints `1` if trace is supported and enabled. Otherwise prints `0`.	**MPserv only**
trace_triggers [on | off] Enables or disables support for trace triggers. If no argument is specified, this command reports whether or not trace triggers are enabled. If you specify `on`: • all hardware and software breakpoints are disabled • you cannot set new breakpoints • before downloading your program, you must disable system calls with the `syscalls off` command For more information, see "Using Triggers with PowerPC 405, 440, and 460" on page 362.	**STP, PPC 405, 440, and 460 only; available only in MPserv**

Target-Specific Special Registers

The following sections provide detailed listings of target-specific special registers that can be used with the **rr** and **rw** commands.

ARM

This section contains information about special registers for ARM targets.

If your target is an ARM 11, ARMv7-A, or ARMv7-R, you can use the **rr** and **rw** commands to access an arbitrary register that is not available by name. To do this, follow the command with:

```
coproc,opcode_1,CRn,CRm,[opcode_2]
```

Where *coproc*, *opcode_1*, *CRn*, *CRm*, and *opcode_2* refer to the same fields as an ARM MCR or MRC instruction.

Only use this syntax if the a register is not available by name. If you use this syntax with a register that is available by name, it might corrupt the register.

When using this syntax:

- Do not put spaces between any of the arguments.
- The maximum value for *opcode_1* and *opcode_2* is 7. The maximum value for the other parameters is 15.
- If you omit *opcode_2*, it defaults to zero.

For example, the following two commands are equivalent. They both read the coprocessor 15 ID register:

```
rr cp15,0,c0,c0,0
rr p15,0,c0,c0
```

On ARM Cortex targets, you can use the **rr** and **rw** commands with the following special registers. Such transactions are always 32-bit and must be 4-byte aligned.:

- `ahb[0x`*`addr`*`]` performs transactions on the AHB-AP.
- `apb[0x`*`addr`*`]` performs transactions on the APB-AP.
- `axi[0x`*`addr`*`]` performs transactions on the AXI-AP.

For example:

```
rr ahb[0x80024000]
```

Reads the value at address `0x80024000` on the AHB-AP.

Power Architecture

This section contains information about special registers for Power Architecture targets.

On Power Architecture QorIQ targets, you can use **rr** and **rw** commands to directly access CCSR space. To do this, follow the command with:

```
ccsr[0xaddr]
```

When using this syntax:

- You can add a suffix of `.1`, `.2`, or `.4` to specify the access size in bytes. If the access size is not specified, the default is 4 bytes.
- `0x`*`addr`* must be aligned to the access size.

For example:

```
rr ccsr[0xc14].4
```

Reads the 4-byte value at a `0xc14` offset into CCSR space. On the QorIQ P4080, this is the LAW_LAWBARL1 register.

Test Commands

selftest Runs diagnostic tests on the probe's trace components. These tests take several seconds, and take longer for probes that have additional trace memory. If any of the tests fail, ensure the pod is securely attached to the probe. If **selftest** still fails, contact Green Hills Support.	**STPv3 Only**
vb [*count*] Verifies that the JTAG bypass register can be scanned properly. This test scans several predefined patterns, and continues to scan in *count* number of random patterns. The default value of *count* is 1000. This is a good first step to verify 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 for BDM (PowerPC 5xx, PowerPC 8xx, and ColdFire) targets**
vbp *start_address* [*count* [-i[*inc*]] | -fix | -ignore | -nocheck] 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. This command takes the same arguments as the **vc** command. For more information, see the **vc** documentation.	**Not for eTPU cores**
vbph *start_address* [*count* [-i[*inc*]] | -fix | -ignore | -nocheck] Tests hardware 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. This command takes the same arguments as the **vc** command. For more information, see the **vc** documentation.	**Not for eTPU cores**

vc *start_address* [*count* [-i[*inc*]] | -fix | -ignore | -nocheck]

Not for eTPU cores

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.

Before performing this test, the probe checks to determine if the current state of the target could interfere with the tests. If so, it lists a description of each issue and aborts the test.

The **vc** command accepts the following arguments:

- *start_address* — A 1 KB area of memory to which the test program is written. You must be able to read, write, and execute from this memory.
- *count* — The number of times to run the test. The default value is `1`. The maximum value is `1000`.

It also accepts the following options (not available in the MULTI Debugger **Cmd** or **Target** panes):

- `-fix` — If any problems are found by the pre-test check, try to fix those problems and run the test.
- `-i[`*`inc`*`]` — On each successive run, increase the address where the test is performed by *inc*. If you do not specify *inc*, the address is incremented by the size of the test program.
- `-ignore` — Do not abort the test when the pre-test check finds potential problems.
- `-nocheck` — Do not perform the pre-test check.

For example:

```
>vc 0xa0000000
```

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

```
>vc 0x8000 0x10 -i0x100
```

Performs the test 16 times, starting at address `0x8000` and adding `0x100` to the address on each subsequent test.

ve *start_address* [*count* [-i[*inc*]] | -fix | -ignore | -nocheck]

Verifies that the byte order (endianness) configuration setting matches the actual setting on the target.

This command takes the same arguments as the **vc** command. For more information, see the **vc** documentation.

vle **STP only**

Verifies the communication link with the active trace pod. This command is useful for ensuring that the probe can communicate with the active trace pod, and that there are no loose connections or bad cables.

This command prints an error message if there is a problem with your SuperTrace setup. If your cable, trace pod, and connection are functioning properly, the command will not print anything.

vm [*access_size*] *start_address length*

Tests memory by writing a random sequence of values to the specified memory range and then reading it back, comparing the values.

- *access_size* — Specifies the size, in bytes, of the memory accesses. If you do not specify this argument, the probe uses a *block* access, which is the fastest way to access memory. Because the size of a block access varies by target and the access method may differ from normal accesses, only use it if you want the test to be fast, but do not care how the access is performed.
- *start_address* — Specifies the first address in the memory range.
- *length* — A 32-bit value that specifies the number of bytes to test.

Examples:

```
>vm 0xa0000000 0x10000
```

Tests memory `0xa0000000` to `0xa000ffff`.

```
>vm 1 0x0 0x7fff
```

Tests byte access `0x0` to `0x8000`.

vr [*$count*]

Verifies that the processor's general purpose registers can be read and written properly. This test writes random values into the general purpose registers, reads the values back, and verifies that the values read back match the values written.

This test requires that the target is halted. It is a good test after **vb** to verify that more complicated probe-target communications are working. It is simpler than the **vm** test because it does not require any memory to be present or initialized.

The optional argument *$count* specifies the number of times the test is run. The default value is 1.

Examples:

```
vr 50
```

Performs the test 50 times. Each test uses a different set of random values to write to the registers for verification.

vrh *start_address* [*count* [-i[*inc*]] | -fix | -ignore | -nocheck] Verifies asynchronous run/halt control by writing a test program to memory and executing it. This test verifies register and memory reads and writes between every run and halt sequence. This command takes the same arguments as the **vc** command. For more information, see the **vc** documentation.	**Not for eTPU cores**
vsi *start_address* [*count* [-i[*inc*]] | -fix | -ignore | -nocheck] 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. This command takes the same arguments as the **vc** command. For more information, see the **vc** documentation.	**Not for eTPU cores**
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.	

Specifying Numbers with Green Hills Debug Probe Commands

For many Green Hills Debug Probe commands, you can specify numbers in regular decimal form (for example, `1024`) or in hexadecimal form with a leading `0x` (for example, `0x400`). Decimal and hexadecimal numbers can be as large as your application requires. Leading zeros in hexadecimal numbers are significant. For example, `0x01234_5678` is considered a 36-bit number, and would be rejected in a context that requires a 32-bit number.

To make numbers less cumbersome, you can use the special fill digits `h` (for high) and `z` (for zero) when the number of bits of a number is known (for example, when specifying a register value).

In some circumstances, you may also need to use delimiters with these fill digits to avoid ambiguities. For instance, you cannot replace both the `0`s and `1`s in the number `0xa001ffff` with `z`s and `h`s, because this would be ambiguous. To overcome this problem, place delimiters in the number. You can place an underscore (_) every 16 bits, and a colon (**:**) every 32 bits.

The subsequent table demonstrates the use of fill digits and delimiters to provide shorthand representations of hexadecimal numbers.

Hexadecimal number	Number with fill digits and delimiters
`0xa0000000`	`0xaz`
`0xa001ffff`	`0xaz1ffff`
`0xa001ffff`	`0xa001h`
`0xa001ffff`	`0xaz1_h`
`0xffffffffa0010000`	`0xh:a001z`

Data Types for Green Hills Debug Probe Commands

Some of the Green Hills Debug Probe commands use specific range and bitfield types, which are described in the following.

Range Type

The syntax:

```
[x][..][y] | *
```

specifies an inclusive range of unsigned integers from `x` to `y`. `x` must be less than or equal to `y`, and both `x` and `y` are unsigned integers. If `x` is omitted, `x = 0` is assumed. If `y` is omitted, the highest possible value is assumed for `y`.

Using an asterisk in place of the `[x][..][y]` syntax specifies a range containing all possible values.

A single number is used to specify a range of one value.

The following table lists several examples that demonstrate the syntax for specifying ranges. These examples assume a 32-bit range.

Syntax	Meaning
`3..5`	3, 4, 5
`3.5`	invalid range
`3...5`	invalid range
`5..3`	invalid range

Syntax	Meaning
`3..3`	3
`3`	3
`..7`	0, 1, 2, 3, 4, 5, 6, 7
`18..`	18, 19, 20, ... 4294967295
`*`	0, 1, 2, 3, ... 4294967295

Bitfield Types

The syntax:

`number | key[=value][,key[=value]...]`

specifies a value and is typically used with commands for writing cache and TLB tags. The value can be specified numerically (with the *number* argument), or with a set of comma-separated keys that correspond to individual bitfields. Keys not specified default to `0`. Key names are case-insensitive.

The bitfield keys are specific to each command and each CPU architecture. For CPU-specific bit information, see Appendix C, “CPU-Specific Bit Tables” on page 299.

Example 5.2. Bitfields and Bit Positions

The PowerPC 440's TLB virtual tags consist of the following bitfields and bit positions.

Bit	Name
0	SR
1	SW
2	SX
3	UR
4	UW
5	UX
6	E
7	G

Bit	Name
8	M
9	I
10	W
12..19	TID
20..23	SIZE
24	TS
25	V

These bit positions do not correspond to actual TLB virtual tag bit positions. They are translated by the probe to the actual hardware's bit positions when written.

The key specification:

```
v,!ts,size=0x9,tid=0x0,!w,i,m,g,!e,ux,uw,ur,sx,sr,sr
```

corresponds to a number value of `0x29003bf`.

Other Green Hills Debug Server Commands

In addition to the commands described in “Green Hills Debug Probe Commands” on page 174, the Green Hills Probe also accepts the generic Green Hills debug server commands listed in this section.

You can enter all of the commands listed below directly into the **mpserv Target** pane. You can also enter these commands into the MULTI Debugger command pane using the **target** command. These commands cannot be entered through telnet or a serial terminal. All of the Green Hills debug server commands are case-insensitive.

Generic Debug Server Commands

You can enter all of these commands directly into the debug server **Target** pane. You can also enter these commands into the MULTI Debugger command pane using the **target** command. All of the Green Hills debug server commands are case-insensitive.

addressof *symbol* Returns the address of *symbol*. This command requires that an image be loaded in the MULTI Debugger.
amask [*mask*] [*value*] If no arguments are specified, the current settings of the download mask and value are displayed. The default settings of *mask* and *value* are `0xffffffffffffffff` and `0`, respectively. If *mask* and *value* are specified, **amask** sets the download mask and value to *mask* and *value*. When **mpserv** downloads a program, it will bitwise AND *mask* and bitwise OR *value* with the download addresses, as follows: `address = (address & mask) | value` The **amask** command is useful for shifting download target addresses without relinking a program. However, the program being downloaded still retains its original relocations and might not run correctly at its new destination address without further support on the target.
close *fd* Closes the specified file descriptor, *fd*.
debug *n* Sets the debugging bit flags. Do not use this command unless directed to do so by Green Hills Technical Support.
delrange *addr* Deletes a checked address range starting at *addr*. See **setrange** for more information about checked memory ranges.
delrangeall Deletes all checked address ranges. See **setrange** for more information about checked memory ranges.
echo [on | off] If no argument is specified, the current echo mode is printed. If **on** or **off** is specified, printing of commands executed in a script are enabled or disabled.
fprint *fd string* Prints *string* to the specified file, *fd*, with script variable expansion.
fprintb *fd integer* Prints *integer* to the specified file, *fd*, in binary mode.
fread *fd identifier* Reads one line of text from the specified file, *fd*. 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, *fd*, 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.

help [*command* | *group*]

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

If a *command* is specified, detailed help information about the specified *command* is printed.

If a *group* is specified, a list of all of the commands in the group with information about the arguments each command takes is printed. Valid command groups include **target**, **server**, and **scripting**.

Example 1:

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

Example 2:

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

listrange

Lists all checked ranges. See **setrange** for more information about checked memory ranges.

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 one or more arguments are specified, **load** instructs which sections to include in host-to-target downloads. `.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.

You can combine the **load** command with the **noload** command.

Green Hills startup code clears all `.bss` sections so you do not need to download them. In most cases, the default setting is `text data`, but the default varies according to your debug server.

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, the indicated memory *address* on the target is read.

If `=val` is specified, *val* to the specified memory *address*on the target is written.

Memory addresses and values must be specified in hexadecimal (with or without a leading `0x`).

The optional **-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 argument is specified, the current **noload** settings are displayed.

If one or more arguments are specified, **noload** 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. Setting the **all** option excludes all of these sections from the download.

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; therefore, downloading them to the target is usually unnecessary.

---

open *file*

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

---

print *string*

Prints *string* with script variable expansion.

Example:

```
> print Test
Test
```

---

random *max*

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

---

reg [*reg* [=*val*]]

Reads or writes to the specified registers.

If no arguments are specified: Lists the names and values of all target registers.

If a register name is specified: Prints the value of the specified register.

If a value (`=val`) is specified for a register: Sets the register to the specified value. Values should be given in hexadecimal form with no leading `0x`.

Example:

```
 reg r1
reg r1 is 0x0001a000
> reg r1=1
reg r1 is now 0x00000001
```

regnum *reg*[=*val*]

If =*val* is not specified, the specified register, *reg* is read and returned.

If `=val` is specified, **regnum** writes *val* to the specified register, *reg*.

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

Example:

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

rg *regname*[=*val*]

If *val* is not specified: Reads and returns the specified register, *regname*.

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

script *file*

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

setrange *addr length*

Sets a checked address range beginning at *addr* and extending for *length* bytes.

A checked address range is a range of addresses that are considered inaccessible. Any memory access, read, or write to a checked address range can fail.

The **setrange** command is useful for restricting access to target memory that might be sensitive or volatile.

setup *file*

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

This **setup** command functions differently than the Green Hills Probe **setup** command.

sleep *n*

Suspends **mpserv** for *n* 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 (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 *variable* and releases any memory associated with it.
 	
Example:
 	
`> x=5`
`> undef x`
`> print $x`
`Error: variable undefined!`

Green Hills Probe and SuperTrace Probe Terminal Prompts

At a terminal prompt, a probe indicates the target's type, position in the scan chain, and current status. Commands are directed to the core identified by the prompt. The command line prompt for single-core target systems is:

```
coretype[status]
```

The command line prompt for multi-core target systems uses the following syntax:

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

where:

- *coretype* indicates the CPU core type.
- *position* indicates the position in the JTAG scan, with `0` indicating the first position (for multi-core systems only).
- *status* indicates the current status. Values for *status* are listed in the table below.

Value of *status*	Meaning
?	The status of the target is currently unknown, either because the probe has not requested target information or because an error has occurred.
!	The target is not in a recognizable state. This can be caused by incorrect probe configuration or faulty hardware.
-	The target is currently held in reset, meaning that the probe was attempting to reset and halt the processor, but the processor appeared to stay in reset instead of entering debug mode. This status is common on systems with multiple processors, where a primary processor resets successfully, but the other processors are held in reset until they are released by software running on the primary processor.
x	The target is powered off.
B	The target is currently halted because of a hardware execution breakpoint or data watchpoint.
C	The probe has requested that the target resume, but the target had not successfully left debugging mode when the probe last queried for the target status.
E	Target is currently halted because of an unknown exception.
H	The probe has requested that the target halt, but the target had not successfully entered debugging mode when the probe last queried for the target status.
R	The target is currently stopped at the reset vector after a reset.
Y	The target is busy and status cannot be received.
b	The target is currently halted because of a software breakpoint.
c	The target is running under control of the probe.
f	The target is running freely. (This can occur if the target is reset external to the probe.)
h	The target is currently halted because of a halt request from the probe.
n	The target is not connected and the pins are tri-stated.

Value of *status*	Meaning
s	The target is currently halted after completing a single instruction step.

Example 5.3. Terminal Prompt for Single-Core Target

The terminal prompt:

```
mips32_4kep[h] %
```

indicates that the probe is connected to a single MIPS Technologies MIPS32 4KEp core that is presently halted.

Example 5.4. Terminal Prompt for a Multi-Core Target

The terminal prompt:

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

indicates that probe commands are directed to the ARM7TDMI core, which is the first device in the scan chain and is currently halted because of a software breakpoint.

Appendix A

Legacy Scripting Reference

Contents

This chapter documents the legacy Green Hills debug server scripting language.

The Green Hills Debug Server Scripting Language

In addition to the commands listed in “Generic Debug Server Commands” on page 218, 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 entering a script line by line, nothing is 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.
- Storing commands in a script file and then using the **-setup** option or **setup** command (or the **Target Setup Script** field in some Connection Editors) to ensure that the script file is automatically executed immediately prior to every host-to-target download.

General Notes

When writing scripts, keep the following points in mind:

- There can be no 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 the Green Hills debug server scripting language.

Expressions

Expressions in the Green Hills debug server scripting language are similar to C language expressions. Unlike C, each operator has its own precedence (`10/2*5` evaluates to 1, not 25), and there are no unary operators (such as `~` and unary `-`). 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
`( )`	Grouping of operators to ensure desired evaluation	Grouping of 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
`>=`	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 underscore (_) character, but cannot begin with a number, or have the same name as a command.

Arrays

Arrays are indexed lists of variables. Each cell in an array can contain a string, an integer, or an 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, an integer, or an array to one cell of an array. To reference a cell, follow the array identified 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. After an array element has been allocated, it can only be deallocated by 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 *expression* evaluates to zero, nothing happens. Otherwise, the block of statements between the **if** and **endif** lines are executed.

Syntax	Effect

if *expression*
statements
else
statements
endif
	If *expression* evaluates to zero, the debug server executes the block of statements between the **else** and **endif** lines. Otherwise, the block of statements between the **if** and **else** lines are executed.

if *expression1* *statements* **elif *expression2*** *statements* [**elif *expressionX*** *statements*]... **endif**	If *expression1* does not evaluate to zero, the debug server executes the block of statements between the **if** and **elif** lines. If *expression1* does evaluate to zero and *expression2* does not evaluate to zero, the debug server executes the block of statements between the **elif** and **endif** lines. If both *expression1* and *expression2* evaluate to zero, the debug server continues evaluating each of the subsequent **elif** expressions (if any) that occur before the **endif** statement and will execute the block of statement associated with each **elif** that does not evaluate to zero. If none of the **elif** expressions evaluates to non-zero, the debug server 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 are executed as long as *expression* does not evaluate to zero.

do *statements* **dowhile *expression***	The statements between **do** and **while** are executed one time, then continue to execute as long as *expression* does not evaluate to zero.

Be careful to avoid infinite loops. If an infinite loop occurs, you must shut down and restart the debug server.

Variable Expansion

To use script variables as arguments to Green Hills debug server 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 `$` character. This passes a decimal string for integer variables and passes the string itself for string variables. Integers are expanded as two's complement signed 32-bit integers. An entire array cannot be given as an argument to a command.
- To pass an integer variable to a command as a hexadecimal string, prepend the variable name with two `$` characters. Use this method with commands such as **m** that require arguments in hexadecimal form. The hexadecimal string does not include a leading `0x`.

Variable expansion must be unambiguous. 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 Green Hills debug server scripting language as a guide when writing your own scripts.

Example A.1. Testing Script File Access in ASCII Mode

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 Linux/Solaris 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 A.2. Accessing a File

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 A.3. Calculating a CRC32 Value

The following script calculates the CRC32 value of the memory range `0x010000` to `0x010100` and prints the result in the **Target** pane. 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
```

Appendix B

Pin Tables

Contents

Overview

The debug interface for most CPU types is essentially a synchronous serial port: serial data in, serial data out, and bit clock. This applies to JTAG, BDM, and SWD debug ports. Additional pins have been added for system reset, debug interruption, and status notification. Different manufacturers have standardized their own connector and pin configuration schemes, but a common set of design rules apply to all:

1. Route the debug interface signals together, maintaining approximately equal signal length and termination.
2. Provide a continuous ground return path adjacent to the debug signals.
3. Place the debug interface connector as close as reasonably possible to the CPU, and keep track lengths short.This will reduce the antenna area of these signals.
4. Pull all CPU debug input signals to a high or low state when no debug probe is connected. Choose an appropriate value for the pull-up or pull-down resistors to reduce crosstalk and to meet valid logic level requirements.
5. Clearly mark the pin orientation of the debug connector to avoid backward probe insertion. Use a keyed, shrouded header if possible.
6. Series-terminate the CPU debug output signals as needed for improved impedance matching.
7. Use the standard debug connector and assignments for your target processor family. Although the Green Hills debug probes can use nearly any standard connector on any supported processor, other probe manufacturers cannot support this connection.

Note

The information published in this document is an extract from CPU product data sheets and application notes, and is provided for information purposes only. For detailed information please check the most recent version of the relevant documentation from the target device manufacturer. Green Hills Software cannot be held responsible for any incorrect or incomplete information.

Green Hills Probe Connector Pin Assignments

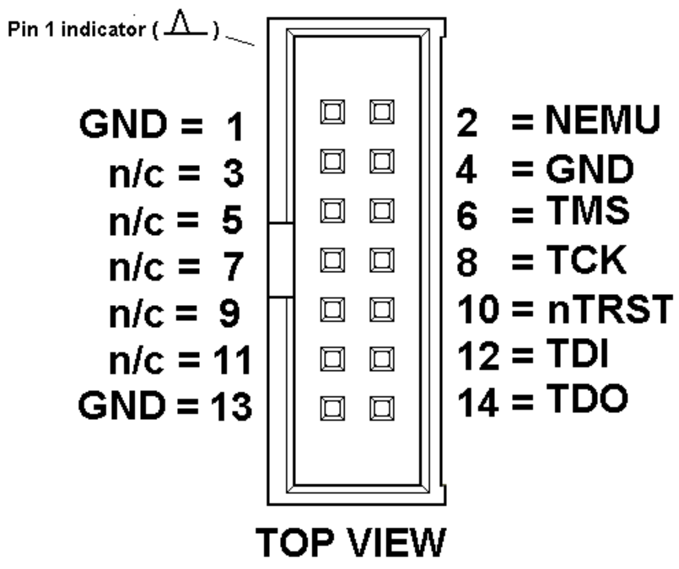
The reference tables below identify the pin numbers and pin names for the connectors supported by the Green Hills Probe.

CPU Families and Debug Connectors

CPU Family	Debug Family	Connector	See
Analog Devices Blackfin	Analog Devices DSP	14-pin 0.100" (2.54 mm) dual-row header	"Analog Devices DSP" on page 239
ARM Licensees (Legacy 14-pin)	ARM (14-pin) (For all ARM & XScale Targets)	14-pin 0.100" (2.54 mm) dual-row header	"ARM Legacy 14-Pin" on page 241
ARM Licensees (Legacy 20-pin)	ARM (20-pin) (For all ARM & XScale Targets)	20-pin 0.100" (2.54 mm) dual-row header	"ARM Legacy 20-Pin" on page 243
ARM Licensees (CoreSight 20-pin)	ARM (20-pin) (For all CoreSight Targets)	20-pin 0.050" (1.27 mm) dual-row header	"ARM CoreSight 20-Pin" on page 245
ARM Licensees (CoreSight 10-pin)	ARM (10-pin) (For all CoreSight Targets)	10-pin 0.050" (1.27 mm) dual-row header	"ARM CoreSight 10-Pin" on page 247
MIPS Licensees (IDT323xx, others)	MIPS EJTAG V2.0 and lower	12-pin (minimum) 0.050" (1.27 mm) dual-row header	"MIPS EJTAGV2.0 and lower" on page 255
MIPS 4K/5K/20K,Licensees	MIPS EJTAG V2.5 and higher	14-pin 0.100" (2.54 mm) dual-row header	"MIPS EJTAG V2.5 and higher" on page 257
Freescale ColdFire V2, V3, V4, V4e, V5, 52xx, 53xx, 54xx	ColdFire BDM	26-pin 0.100" (2.54 mm) dual-row header	"ColdFire BDM" on page 259
Freescale and IBM PowerPC 4xx	PowerPC 4xx	16-pin 0.100" (2.54 mm) dual-row header	"PowerPC 4xx Targets" on page 269
Freescale PowerPC 5xx, 8xx	PowerPC BDM	10-pin 0.100" (2.54 mm) dual-row header	"PowerPC BDM" on page 264

CPU Family	Debug Family	Connector	See
PowerPC 51xx, 52xx, 6xx, 7xx, 74xx, 82xx, 83xx, 85xx, and QorIQ	PowerPC COP	16-pin 0.100" (2.54 mm) dual-row header	“PowerPC COP” on page 266
Intel x86	Intel XDP-60	60-pin MICTOR connector	“Intel XDP-60” on page 249
Intel x86	Intel XDP-SSA	31-pin dual-row header	“Intel XDP-SSA” on page 252
Intel x86	Intel XDP-SFF-24	24-pin flat flex	“Intel XDP-SFF-24” on page 254
Freescale PowerPC 55xx/56xx/57xx	PowerPC 55xx/56xx/57xx	38-pin MICTOR connector	“Generic Nexus” on page 271
Renesas SH-2A and SH2A-FPU	Renesas H-UDI	14-pin 0.100" (2.54 mm) dual-row header	“Renesas H-UDI” on page 277
PowerPC 55xx/56xx57xx	EOnCE	14-pin 0.100" (2.54 mm) dual-row header	“EOnCE” on page 273
Texas Instruments OMAP	Texas Instruments DSP	14-pin 0.100" (2.54 mm) dual-row header	“Texas Instruments DSP” on page 275

Analog Devices DSP



Pin description and termination

Pin	Name	Description	Direction	Notes
2	`NEMU`	Emulation Pin	From Target	
3	`n/c`			
5	`n/c`			
6	`TMS`	JTAG TAP Machine State	To Target	2
7	`n/c`			
8	`TCK`	JTAG TAP Clock	To Target	2
9	`n/c`			
10	`nTRST`	JTAG TAP Reset, active low	To Target	1
11	`n/c`			
12	`TDI`	JTAG Test Data In	To Target	2
14	`TDO`	JTAG Test Data Out	From Target	
1, 4, 13	`GND`	Ground reference		

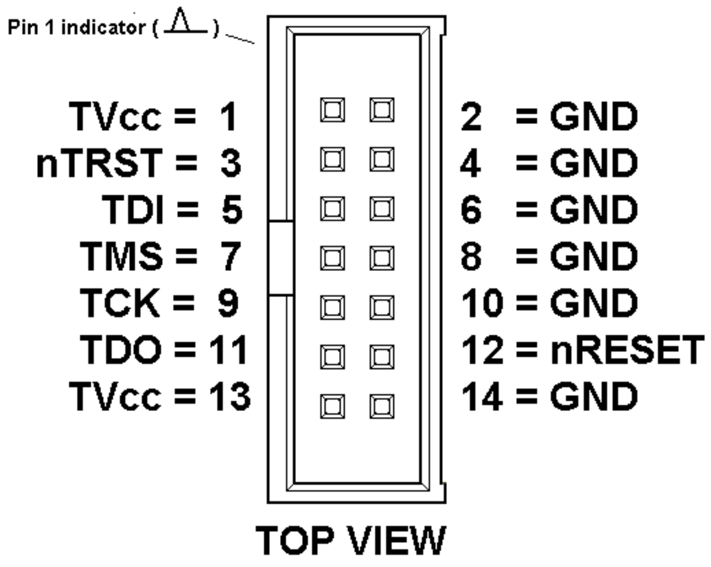
Notes:

1. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
2. Pull up to a logical high with a resistor.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-2`	`103311-2`
3M [http://www.3m.com]	`2514-6002UB`	`2514-5002UB`

ARM Legacy 14-Pin



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TVcc`	Target voltage reference	From Target	1
3	`nTRST`	JTAG TAP Reset, active low	To Target	2
5	`TDI`	JTAG Test Data In	To Target	3
7	`TMS`	JTAG TAP Machine State	To Target	3
9	`TCK`	JTAG TAP Clock	To Target	3
11	`TDO`	JTAG Test Data Out	From Target	
12	`nRESET`	Target Reset, active low	Bidirectional	3, 4
13	`TVcc`	Target voltage reference	From Target	1
2, 4, 6, 8, 10, 14	`GND`	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1000` ohms. The power supply voltage should be representative of the logic levels used in the JTAG, BDM, or SWD interface. You may connect this pin directly to the target power, although high

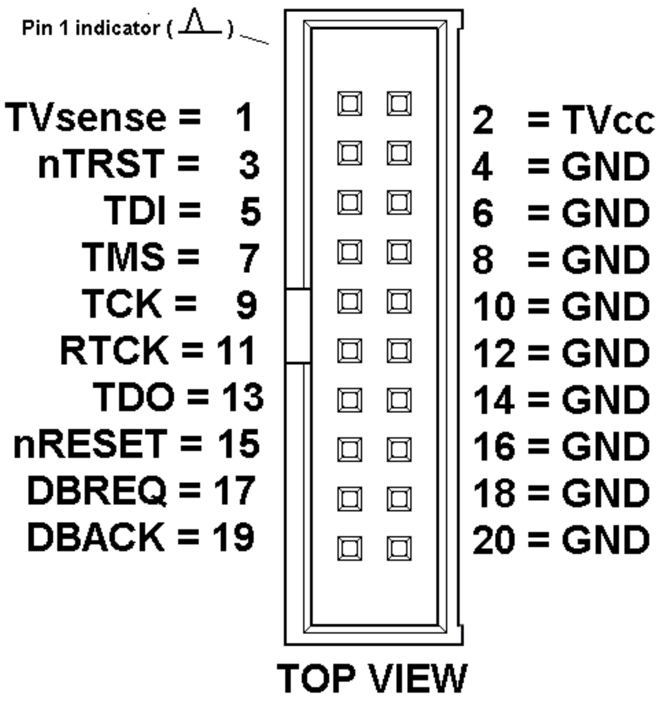
currents might be encountered if the pin is accidentally shorted to ground. The probe draws negligible current from these pins.

2. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
3. Pull up to a logical high with a resistor.
4. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-2`	`103311-2`
3M [http://www.3m.com]	`2514-6002UB`	`2514-5002UB`

ARM Legacy 20-Pin



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TVsense`	Target voltage reference	From Target	1
2	`TVcc`	Target power	From Target	1
3	`nTRST`	JTAG TAP Reset, active low	To Target	2
5	`TDI`	JTAG Test Data In	To Target	3
7	`TMS/SWDIO`	JTAG TAP Machine State	To Target	3
9	`TCK`	JTAG TAP Clock	To Target	3
11	`RTCK`	Return `TCK` (reclocked)	From Target	5
13	`TDO/SWO`	JTAG Test Data Out	From Target	
15	`nRESET`	Target Reset, active low	Bidirectional	3, 4
17	`DBREQ`	Probe Debug Request	To Target	6
19	`DBACK`	Probe Debug Acknowledge	From Target	6

Pin	Name	Description	Direction	Notes
4, 6, 8, 10, 12, 14, 16, 18, 20	`GND`	Ground reference		

In SWD mode, `nTRST` and `TDI` are unused, `TMS` becomes `SWDIO` and is bidirectional, and `TDO` becomes `SWO` and is unused.

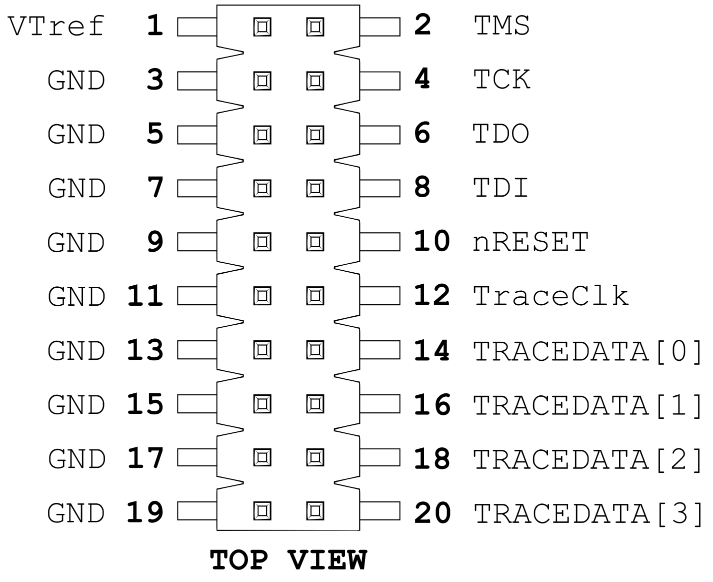
Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage should be representative of the logic levels used in the JTAG, BDM, or SWD interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.
2. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
3. Pull up to a logical high with a resistor.
4. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.
5. `RTCK` is a reclocked `TCK` signal, used by the debugging probe to determine the CPU's core operating frequency, useful for debugging CPU targets with variable clock frequencies. Please consult your CPU documentation - this pin is optional. Typical target termination is a low-ohm series termination resistor (`22` to `33` ohms).
6. `DBREQ` and `DBACK` are optional pins. These pins are seldom included in commercial ARM CPU implementations.

Example connector part numbers:

Manufacturer	Vertical	Right Angle
AMP/Tyco [http://www.amp.com]	`103309-5`	`103311-5`
3M [http://www.3m.com]	`2520-6002UB`	`2520-5002UB`

ARM CoreSight 20-Pin



Pin	Name	Description	Direction	Notes
1	`VTref`	Target voltage reference	From Target	1
2	`TMS`	JTAG TAP machine state	To Target	
4	`TCK`	JTAG TAP Clock	To Target	2
6	`TDO/SWO`	JTAG Test Data Out	From Target	
8	`TDI/SWDIO`	JTAG Test Data In	To Target	2
10	`nRESET`	Target Reset, active low	Bidirectional	2, 3
12	`TraceClk`	Trace Clock	To Target	
14	`TRACEDATA[0]`	Trace Data	From Target	
16	`TRACEDATA[1]`	Trace Data	From Target	
18	`TRACEDATA[2]`	Trace Data	From Target	
20	`TRACEDATA[3]`	Trace Data	From Target	
3, 5, 7, 9, 11, 13, 15, 17, 19	`GND`	Ground reference		

In SWD mode, `nTRST` and `TDI` are unused, `TMS` becomes `SWDIO` and is bidirectional, and `TDO` becomes `SWO` and is unused.

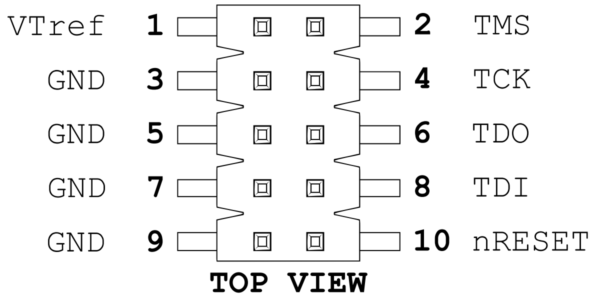
Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between 33 and 1000 ohms. The power supply voltage should be representative of the logic levels used in the JTAG, BDM, or SWD interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.
2. Pull up to a logical high with a resistor.
3. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**
Samtec [http://www.samtec.com]	FTSH-110–01–L-DV-K-A

ARM CoreSight 10-Pin



Pin	Name	Description	Direction	Notes
1	`VTref`	Target voltage reference	From Target	1
2	`TMS`	JTAG TAP machine state	To Target	
4	`TCK`	JTAG TAP Clock	To Target	2
6	`TDO/SWO`	JTAG Test Data Out	From Target	
8	`TDI/SWDIO`	JTAG Test Data In	To Target	2
10	`nRESET`	Target Reset, active low	Bidirectional	2, 3
3, 5, 7, 9	`GND`	Ground reference		

In SWD mode, `nTRST` and `TDI` are unused, `TMS` becomes `SWDIO` and is bidirectional, and `TDO` becomes `SWO` and is unused.

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between 33 and 1000 ohms. The power supply voltage should be representative of the logic levels used in the JTAG, BDM, or SWD interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.
2. Pull up to a logical high with a resistor.
3. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	Vertical
Samtec [http://www.samtec.com]	FTSH-105-01-L-DV-K-A-TR

Intel XDP-60

Pin	Name	Description	Direction	Notes
1	GND	Ground reference		
2	GND	Ground reference		
3	OBSFN_A0	OBS A control/strobe	To Target	1
4	OBSFN_C0	OBS C control/strobe		
5	OBSFN_A1	OBS A control/strobe		
6	OBSFN_C1	OBS C control/strobe		
7	GND	Ground reference		
8	GND	Ground reference		
9	OBSDATA_A[0]	OBS A data	Bidirectional	
10	OBSDATA_C[0]	OBS C data		
11	OBSDATA_A[1]	OBS A data	From Target	
12	OBSDATA_C[1]	OBS C data		
13	GND	Ground reference		
14	GND	Ground reference		
15	OBSDATA_A[2]	OBS A data	From Target	
16	OBSDATA_C[2]	OBS C data		
17	OBSDATA_A[3]	OBS A data	From Target	
18	OBSDATA_C[3]	OBS C data		
19	GND	Ground reference		
20	GND	Ground reference		
21	OBSFN_B0	OBS B control/strobe		2
22	OBSFN_D0	OBS D control/strobe		
23	OBSFN_B1	OBS B control/strobe		
24	OBSFN_D1	OBS D control/strobe		
25	GND	Ground reference		
26	GND	Ground reference		
27	OBSDATA_B[0]	OBS B data		
28	OBSDATA_D[0]	OBS D data		
29	OBSDATA_B[1]	OBS B data		

Pin	Name	Description	Direction	Notes
30	OBSDATA_D[1]	OBS D data		
31	GND	Ground reference		
32	GND	Ground reference		
33	OBSDATA_B[2]	OBS B data		
34	OBSDATA_D[2]	OBS D data		
35	OBSDATA_B[3]	OBS B data		
36	OBSDATA_D[3]	OBS D data		
37	GND	Ground reference		
38	GND	Ground reference		
39	HOOK0	Target power	From Target	
40	ITPCLK/HOOK4	System clock frequency reference	From Target	
41	HOOK1	Reserved		
42	ITPCLK#/HOOK5	System clock frequency reference	From Target	
43	VCC_OBS_AB	OBS termination		
44	VCC_OBS_CD	OBS termination		
45	HOOK2	Split-VTT / reserved	From Target	
46	HOOK6	Detect debug reset	From Target	
47	HOOK3	Reserved		
48	HOOK7	Control debug reset	To Target	
49	GND	Ground reference		
50	GND	Ground reference		
51	SDA	I2C ITP-XDP interface	Bidirectional	
52	TDO	JTAG Test Data Out	From Target	
53	SCL	I2C ITP-XDP interface	Bidirectional	
54	TRSTn	JTAG TAP Reset	To Target	
55	TCK1	JTAG TAP clock	To Target	
56	TDI	JTAG Test Data In	To Target	
57	TCK0	JTAG TAP clock	To Target	
58	TMS	JTAG TAP machine state	To Target	

Pin	Name	Description	Direction	Notes
59	`GND`	Ground reference		
60	`GND`	Ground reference		

Notes

- Used for the `PREQ0` signal.
- Used for the `PREQ1` signal.

Example connector part numbers:

Manufacturer	Name
Samtec [http://www.samtec.com]	60-pin BSH-030–01

Intel XDP-SSA

Pin	Name	Description	Direction	Notes
1	`OBSFN_A0`	OBS A control/strobe	Bidirectional	
2	`GND`	Ground reference		
3	`OBSFN_A1`	OBS A control/strobe	Bidirectional	
4	`OBSDATA_A[0]`	OBS A data	Bidirectional	
5	`GND`	Ground reference		
6	`OBSDATA_A[1]`	OBS A data	Bidirectional	
7	`OBSDATA_A[2]`	OBS A data	Bidirectional	
8	`GND`	Ground reference		
9	`OBSDATA_A[3]`	OBS A data	Bidirectional	
10	`HOOK0`	Target power	From Target	
11	`GND`	Ground reference		
12	`HOOK1`	Reserved		
13	`HOOK4`	System clock frequency reference	From Target	
14	`VCCOBS_AB`	OBS termination		
15	`HOOK5`	System clock frequency reference	From Target	
16	`HOOK2`	Split-VTT / reserved	From Target	
17	`GND`	Ground reference		
18	`HOOK3`	Reserved		
19	`HOOK6`	Detect debug reset	From Target	
20	`GND`	Ground reference		
21	`HOOK7`	Control debug reset	To Target	
22	`SCL`	I2C ITP-XDP interface	Bidirectional	
23	`TDO`	JTAG Test Data Out	From Target	
24	`SDA`	I2C ITP-XDP interface	Bidirectional	
25	`TRSTn`	JTAG TAP Reset	To Target	
26	`GND`	Ground reference		
27	`GND`	Ground reference		
28	`TCK1`	JTAG TAP clock	To Target	
29	`TDI`	JTAG Test Data In	To Target	

Pin	Name	Description	Direction	Notes
30	`TCK0`	JTAG TAP clock	To Target	
31	`TMS`	JTAG TAP machine state	To Target	

Example connector part numbers:

Manufacturer	Name
Hirose [http://www.hirose.com]	DF9C-31S

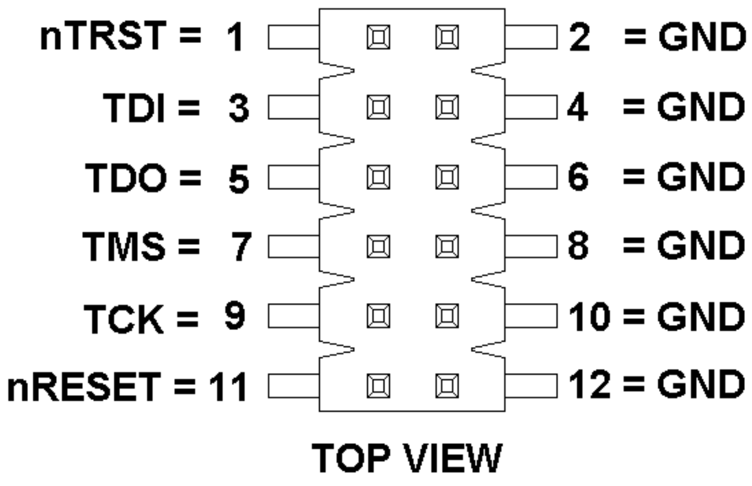
Intel XDP-SFF-24

Pin	Name	Description	Direction	Notes
1	`OBSFN_A0`	OBS A control/strobe	Bidirectional	
2	`OBSFN_A1`	OBS A control/strobe	Bidirectional	
3	`GND`	Ground reference		
4	`OBSDATA_A[0]`	OBS A data	Bidirectional	
5	`OBSDATA_A[1]`	OBS A data	Bidirectional	
6	`GND`	Ground reference		
7	`OBSDATA_A[2]`	OBS A data	Bidirectional	
8	`OBSDATA_A[3]`	OBS A data	Bidirectional	
9	`GND`	Ground reference		
10	`HOOK0`	Target power	From Target	
11	`HOOK2`	Split-VTT / reserved	From Target	
12	`HOOK4`	System clock frequency reference	From Target	
13	`HOOK5`	System clock frequency reference	From Target	
14	`VCCOBS_AB`	OBS termination		
15	`HOOK6`	Detect debug reset	From Target	
16	`HOOK7`	Control debug reset	To Target	
17	`GND`	Ground reference		
18	`TDO`	JTAG Test Data Out	From Target	
19	`TRSTn`	JTAG TAP Reset	To Target	
20	`TDI`	JTAG Test Data In	To Target	
21	`TMS`	JTAG TAP machine state	To Target	
22	`TCK1`	JTAG TAP clock	To Target	
23	`GND`	Ground reference		
24	`TCK0`	JTAG TAP clock	To Target	

Example connector part numbers:

Manufacturer	Name
Molex [http://www.molex.com]	52435-2472

MIPS EJTAGV2.0 and lower



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`nTRST`	JTAG TAP Reset, active low	To Target	1
3	`TDI`	JTAG Test Data In	To Target	2
5	`TDO`	JTAG Test Data Out	From Target	
7	`TMS`	JTAG TAP Machine State	To Target	2
9	`TCK`	JTAG TAP Clock	To Target	2
11	`nRESET`	Target Reset, active low	Bidirectional	2, 3
2, 4, 6, 8, 10, 12	`GND`	Ground reference		

Notes:

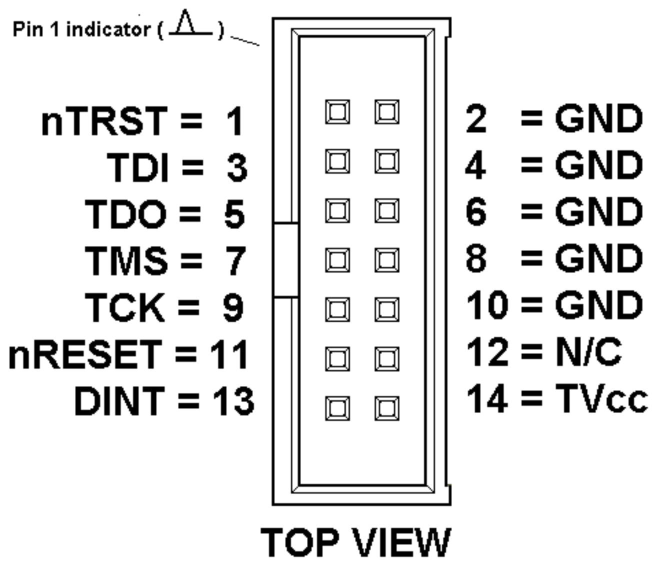
1. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
2. Pull up to a logical high with a resistor.

3. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**
Samtec [http://www.samtec.com]	`FTSH-106-F-DV`

MIPS EJTAG V2.5 and higher



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`nTRST`	JTAG TAP Reset, active low	To Target	2
3	`TDI`	JTAG Test Data In	To Target	3
5	`TDO`	JTAG Test Data Out	From Target	
7	`TMS`	JTAG TAP Machine State	To Target	3
9	`TCK`	JTAG TAP Clock	To Target	3
11	`nRESET`	Target Reset, active low	Bidirectional	3, 4
12	`Key (n/c)`	Optional key (pin removed)		
13	`DINT`	Debug interrupt, active high	To Target	
14	`TVcc`	Target voltage reference	From Target	1
2, 4, 6, 8, 10	`GND`	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage should be representative of the logic levels used in the `JTAG/BDM` interface.

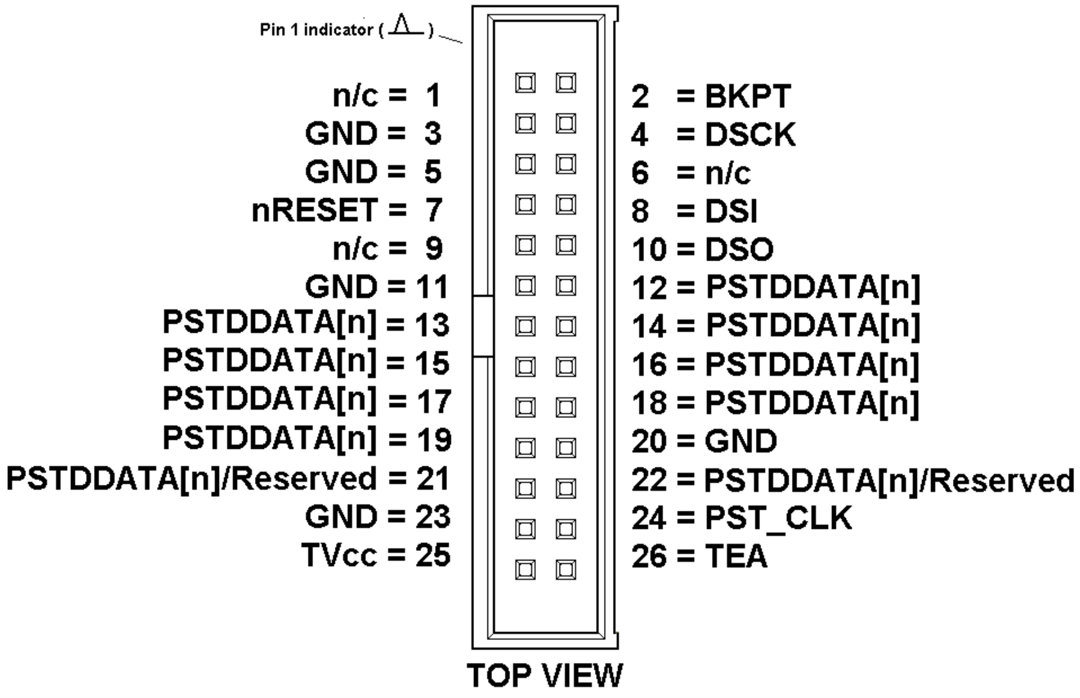
You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.

2. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
3. Pull up to a logical high with a resistor.
4. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-2`	`103311-2`
3M [http://www.3m.com]	`2514-6002UB`	`2514-5002UB`

ColdFire BDM



Pin description and termination

ColdFireV5

Pin	Name	Description	Direction	Notes
1	`n/c`	No connection		
2	`BKPT`	Hardware Breakpoint Input	To Target	
4	`DSCK`	Debug serial clock	To Target	
6	`n/c`	No connection		
7	`nRESET`	System reset, active low	Bidirectional	2, 3
8	`DSI`	Serial data input	To Target	
9	`Pad_Vdd`	Power sense		
10	`DSO`	Serial data output	From Target	
12	`PSTDDATA [8]`	Processor status/data (real time trace)	From Target	
13	`PSTDDATA [7]`	Processor status/data (real time trace)	From Target	

Pin	Name	Description	Direction	Notes
14	`PSTDDATA [6]`	Processor status/data (real time trace)	From Target	
15	`PSTDDATA [5]`	Processor status/data (real time trace)	From Target	
16	`PSTDDATA [3]`	Processor status/data (real time trace)	From Target	
17	`PSTDDATA [2]`	Processor status/data (real time trace)	From Target	
18	`PSTDDATA [1]`	Processor status/data (real time trace)	From Target	
19	`PSTDDATA [0]`	Processor status/data (real time trace)	From Target	
21	`PSTDDATA [4]`	Processor status/data (real time trace)	From Target	
22	`PSTDDATA [9]`	Processor status/data (real time trace)	From Target	
24	`PST_CLK`	Processor clock	From Target	
25	`TVcc`	Target voltage reference	From Target	1
26	`TEA`	Transfer Error Acknowledge Input	To Target	
3, 5, 11, 20, 23	`GND`	Ground reference		

ColdFireV4(e)

Pin	Name	Description	Direction	Notes
1	`n/c`	No connection		
2	`BKPT`	Hardware Breakpoint Input	To Target	
4	`DSCK`	Debug serial clock	To Target	
6	`n/c`	No connection		
7	`nRESET`	System reset, active low	Bidirectional	2, 3
8	`DSI`	Serial data input	To Target	
9	`n/c`	No connection		
10	`DSO`	Serial data output	From Target	
12	`PSTDDATA [7]`	Processor status/data (real time trace)	From Target	
13	`PSTDDATA [6]`	Processor status/data (real time trace)	From Target	
14	`PSTDDATA [5]`	Processor status/data (real time trace)	From Target	
15	`PSTDDATA [4]`	Processor status/data (real time trace)	From Target	
16	`PSTDDATA [3]`	Processor status/data (real time trace)	From Target	
17	`PSTDDATA [2]`	Processor status/data (real time trace)	From Target	
18	`PSTDDATA [1]`	Processor status/data (real time trace)	From Target	
19	`PSTDDATA [0]`	Processor status/data (real time trace)	From Target	
21	`n/c`			
22	`n/c`			
24	`PST_CLK`	Processor clock	From Target	
25	`TVcc`	Target voltage reference	From Target	1
26	`TEA`	Transfer Error Acknowledge Input	To Target	
3, 5, 11, 20, 23	`GND`	Ground reference		

ColdFireV2/V3

Pin	Name	Description	Direction	Notes
1	n/c	No connection		
2	BKPT	Hardware Breakpoint Input	To Target	
4	DSCK	Debug serial clock	To Target	
6	n/c	No connection		
7	nRESET	System reset, active low	Bidirectional	2, 3
8	DSI	Serial data input	To Target	
9	n/c	No connection		
10	DSO	Serial data output	From Target	
12	PST[3]	Processor status (real time trace)	From Target	
13	PST[2]	Processor status (real time trace)	From Target	
14	PST[1]	Processor status (real time trace)	From Target	
15	PST[0]	Processor status (real time trace)	From Target	
16	DDATA[3]	Data (real time trace)	From Target	
17	DDATA[2]	Data (real time trace)	From Target	
18	DDATA[1]	Data (real time trace)	From Target	
19	DDATA[0]	Data (real time trace)	From Target	
21	n/c		From Target	
22	n/c		From Target	
24	PST_CLK	Processor clock	From Target	
25	TVcc	Target voltage reference	From Target	1
26	TEA	Transfer Error Acknowledge Input	To Target	
3, 5, 11, 20, 23	GND	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between 33 and 1.0k ohms. The power supply voltage should be representative of the logic levels used in the JTAG/BDM interface.

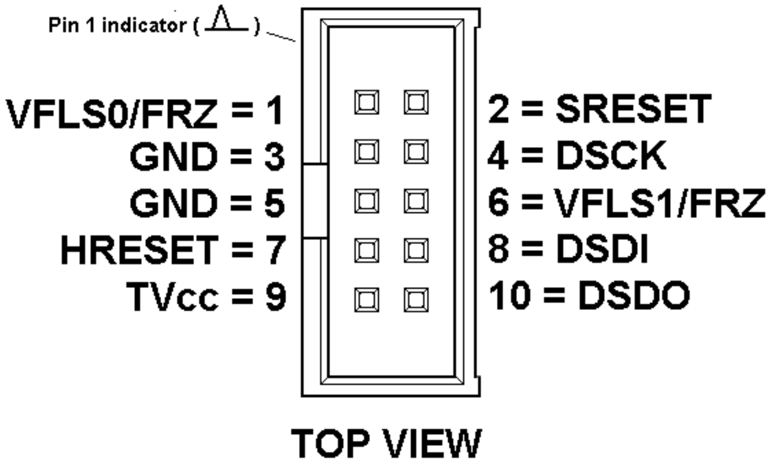
You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.

2. Pull up to a logical high with a resistor.
3. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-6`	`103311-6`
3M [http://www.3m.com]	`2526-6002UB`	`2526-5002UB`

PowerPC BDM



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`VFLS0/FRZ`	Processor Status	From Target	
2	`SRESET`	Target soft reset, active low	Bidirectional	2, 3
4	`DSCK`	Debug serial clock	To Target	2
6	`VFLS1/FRZ`	Processor Status	From Target	
7	`HRESET`	Target hard reset, active low	Bidirectional	2, 3
8	`DSDI`	Debug serial data input	To Target	2
9	`TVcc`	Target voltage reference	From Target	1
10	`DSDO`	Debug serial data output	From Target	
3, 5	`GND`	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage should be representative of the logic levels used in the `JTAG`/`BDM` interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.
2. Pull up to a logical high with a resistor.

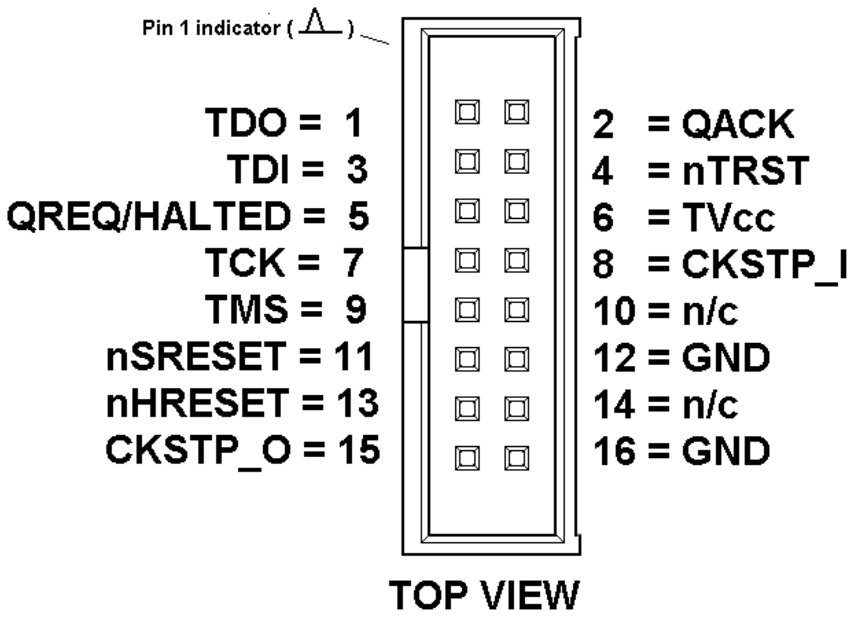
3. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-1`	`103311-1`
3M [http://www.3m.com]	`2510-6002UB`	`2510-5002UB`

PowerPC COP

PowerPC 603ev, 7xx, 74xx, 82xx, 83xx, 85xx, and QorIQ targets.



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TDO`	JTAG Test Data Output	From Target	
2	`QACK`	CPU `QACK`	Bidirectional	5, 10
3	`TDI`	JTAG Test Data Input	To Target	1
4	`nTRST`	JTAG TAP Reset, active low	To Target	3, 11
5	`QREQ`	CPU `QREQ`	From Target	7, 10
6	`TVcc`	Target voltage reference	From Target	2
7	`TCK`	JTAG TAP Clock	To Target	1, 8
8	`CKSTP_I`	CPU checkstop input	To Target	10
9	`TMS`	JTAG TAP Machine State	To Target	1
10	`n/c`			
11	`nSRESET`	Target soft reset	Bidirectional	4, 5, 10

Pin	Name	Description	Direction	Notes
12	`n/c`			9
13	`nHRESET`	Target hard reset	Bidirectional	4, 5, 11
14	`Key`	Optional keying pin, removed on some targets		
15	`CKSTP_O`	Target checkstop output	From Target	10
16	`GND`	Ground reference		

Notes:

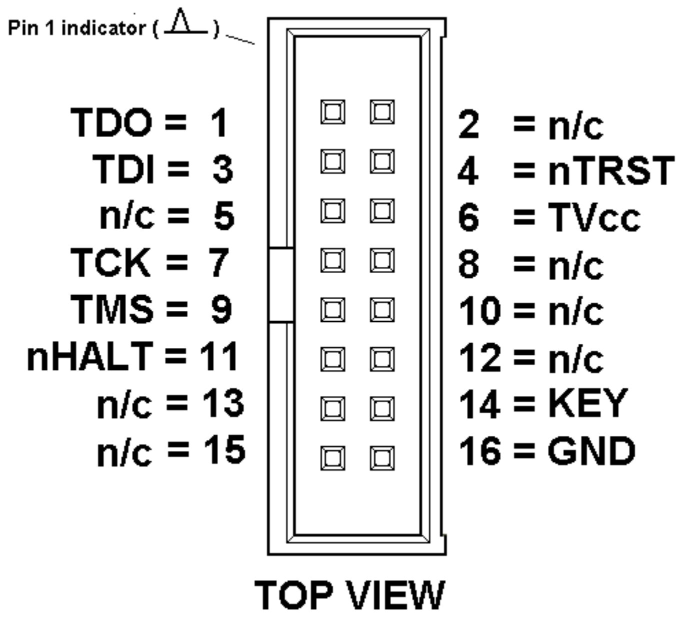
1. To avoid spurious noise affecting the CPU when the probe is not connected, pull this signal to a logical high level with a pull-up resistor on the target. Typical value range will be approximately `4.7k` to `47k` ohms. Please consult the datasheet of your target CPU for proper resistor value.
2. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage should be representative of the logic levels used in the `JTAG` interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.
3. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Consult your target CPU's user manual for specific details.
4. Pull up to a logical high with a resistor.
5. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.
6. While the probe does not require connection to this signal to fully debug the target CPU, certain processors require this signal to be pulled down on the board in order for the probe to debug the processor. Consult your target CPU's user manual for specific details.
7. `QREQ` is sometimes called `HALTED`.
8. `TCK` connection on the board should be as direct as possible, and routed over continuous ground when possible.

9. Pin 12 was specified to be grounded in older COP design documents, whereas newer documents specify pin 12 as no-connect. If your board design requires pin 12 to be no-connect, you may remove pin 12 on the probe's target adapter. For example, if your board design presents power on pin 12, you should disconnect pin 12 from the probe.
10. This signal is optional, and does not need to be connected to the probe for correct debugging operation, however this signal may require some form of termination. Consult your target CPU's user manual for specific details.
11. `nHRESET` and `nTRST` must not be wired, logically ORed, or otherwise tied together on the board. For more information, see “Diagnosing Reset Line Problems” on page 319.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-3`	`103311-3`
3M [http://www.3m.com]	`2516-6002UB`	`2516-5002UB`

PowerPC 4xx Targets



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TDO`	JTAG Test Data Output	From Target	
2	`n/c`			
3	`TDI`	JTAG Test Data Input	To Target	3
4	`nTRST`	JTAG TAP Reset, active low	To Target	2
5	`n/c`			
6	`TVcc`	Target voltage reference	From Target	1
7	`TCK`	JTAG TAP Clock	To Target	3
8	`n/c`			
9	`TMS`	JTAG TAP Machine State	To Target	3
10	`n/c`			
11	`nHALT`	Halt input	To Target	3, 4
12	`n/c`			
13	`n/c`			

Pin	Name	Description	Direction	Notes
14	`Key`	Optional keying pin, removed on some targets		
15	`n/c`			
16	`GND`	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage should be representative of the logic levels used in the `JTAG/BDM` interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.
2. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
3. Pull up to a logical high with a resistor.
4. When attempting to perform a freeze-mode reset, the probe asserts the `nHALT` pin. If the **reset_type** option is set to `chip` (the default) or `core`, this pin is optional. If it is set to `system`, a precise reset is only possible if this pin is connected.

Note

Do not tie the `nHRESET` and `nTRST` pins together. The probe does not use `HALTED` (pin 5), `Checkstop_in` (pin 8), or `Checkstop_out` (pin 15). The pins can be left unconnected.

Example connector part numbers:

Manufacturer	Vertical	Right Angle
AMP/Tyco [http://www.amp.com]	`103309-3`	`103311-3`
3M [http://www.3m.com]	`2516-6002UB`	`2516-5002UB`

Generic Nexus

When debugging with a Green Hills Probe, an adapter is required to connect to a Nexus interface. We recommend that you use a COP to MICTOR adapter. To purchase one of these adapters, contact your Green Hills Sales Representative.

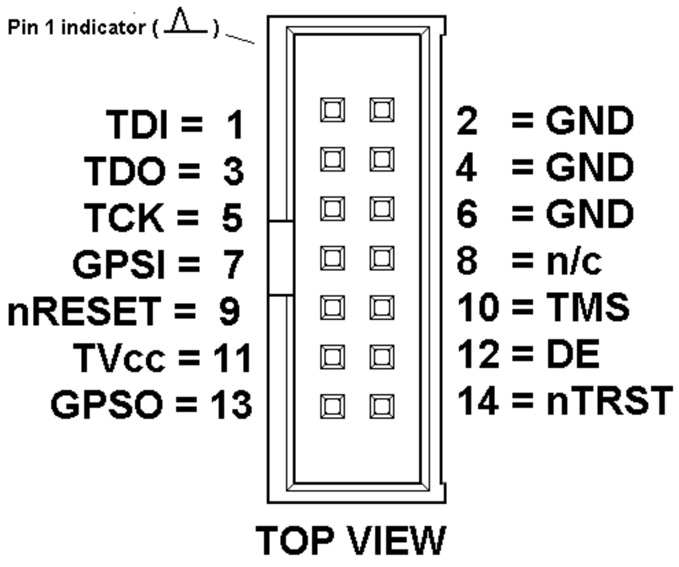
Some hardware implements connections that deviate slightly from the Nexus standard. If you are using a SuperTrace Probe to collect trace data, check your hardware manual to determine if this is the case. If you are using a PowerPC 55xx, see “PowerPC 55xx/56xx/57xx Nexus” on page 289.

This connector uses a bus connection that runs along the middle of the connector for ground; it does not have a pin for this purpose.

Pin	Signal Name	Direction	Notes
1	`n/c`		
2	`n/c`		
3	`n/c`		
4	`n/c`		
5	`VEND_IO0`	Bidirectional	
6	`CLKOUT`	From Target	
7	`VEND_IO2`	Bidirectional	
8	`VEND_IO3`	Bidirectional	
9	`RESET`	To Target	
10	`EVTI`	To Target	
11	`TDO`	Bidirectional	
12	`VREF`	From Target	
13	`VEND_IO4`	Bidirectional	
14	`RDY`	Bidirectional	
15	`TCK`	To Target	
16	`MDO7`	From Target	
17	`TMS`	To Target	
18	`MDO6`	From Target	
19	`TDI`	To Target	
20	`MDO5`	From Target	

Pin	Signal Name	Direction	Notes
21	`TRST`	To Target	
22	`MDO4`	From Target	
23	`VEND_IO1`	To Target	
24	`MDO3`	From Target	
25	`TOOL_IO3`	Bidirectional	
26	`MDO2`	From Target	
27	`TOOL_IO2`	Bidirectional	
28	`MDO1`	From Target	
29	`TOOL_IO1`	Bidirectional	
30	`MDO0`	From Target	
31	`UBATT`	From Target	
32	`EVTO`	From Target	
33	`UBATT`	From Target	
34	`MCKO`	From Target	
35	`TOOL_IO0`	Bidirectional	
36	`MSEO1`	From Target	
37	`VALTREF`	From Target	
38	`MSEO0`	From Target	

EOnCE



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TDI`	JTAG Test Data In	To Target	2
3	`TDO`	JTAG Test Data Out	From Target	
5	`TCK`	JTAG TAP Clock	To Target	2
7	`n/c`			
8	`n/c`			
9	`nRESET`	Target Reset, active low	Bidirectional	2, 3
10	`TMS`	JTAG TAP Machine State	To Target	2
11	`TVcc`	Target voltage reference	From Target	1
12	`n/c`			
13	`n/c`			
14	`nTRST`	JTAG TAP Reset, active low	To Target	2
2, 4, 6	`GND`	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage

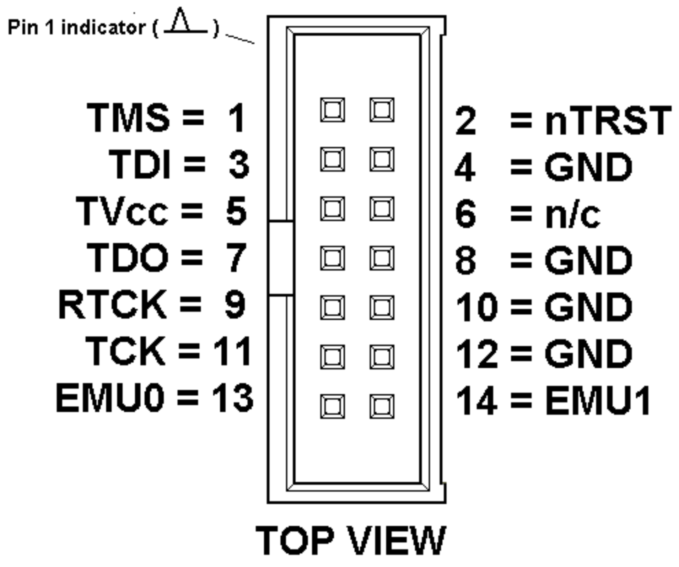
should be representative of the logic levels used in the `JTAG/BDM` interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.

2. Pull up to a logical high with a resistor.
3. This signal is typically used with open-drain drivers and pull-up resistors, hence the bidirectional notation. This pin can also be configured as an input-only to the target, however in this configuration the probe will not be able to directly detect a target-asserted reset.

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-2`	`103311-2`
3M [http://www.3m.com]	`2514-6002UB`	`2514-5002UB`

Texas Instruments DSP



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TMS`	JTAG TAP Machine State	To Target	3
2	`nTRST`	JTAG TAP Reset, active low	To Target	2
3	`TDI`	JTAG Test Data In	To Target	3
5	`TVcc`	Target voltage reference	From Target	1
6	`n/c`	No connection		
7	`TDO`	JTAG Test Data Out	From Target	
9	`RTCK`	Return `TCK` (reclocked)	From Target	4
11	`TCK`	JTAG TAP Clock	To Target	3
13	`EMU0`	Emulation pin 0	From Target	
14	`EMU1`	Emulation pin 1	From Target	
4, 8, 10, 12	`GND`	Ground reference		

Notes:

1. Connect to the target power supply through a current-limiting resistor. The value range should be between `33` and `1.0k` ohms. The power supply voltage

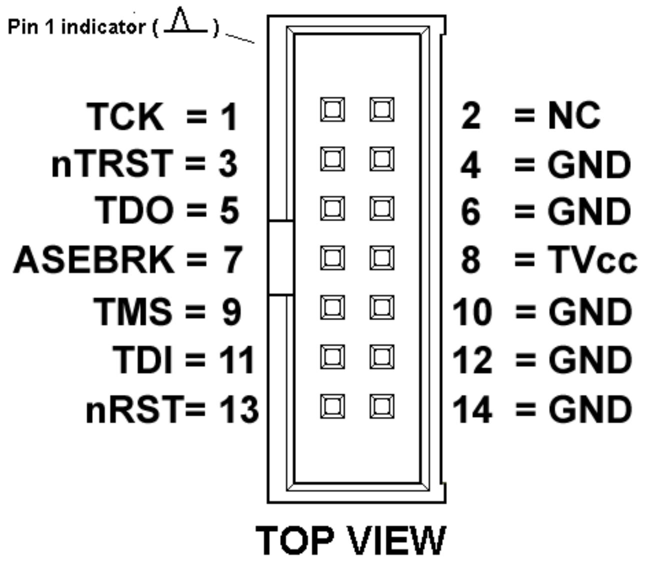
should be representative of the logic levels used in the `JTAG/BDM` interface. You may connect this pin directly to the target power, although high currents might be encountered if the pin is accidentally shorted to ground.

2. Check the target CPU datasheet for treatment of the `nTRST` pin. Common practice is to pull this pin to a defined state (high or low) with a resistor, however some processors require this pin to be asserted during a cold reset for proper CPU initialization. Please consult the documentation for your target CPU.
3. Pull up to a logical high with a resistor.
4. `RTCK` is a reclocked `TCK` signal, used by the debugging probe to determine the CPU's core operating frequency, useful for debugging CPU targets with variable clock frequencies. Typical target termination is a low-ohm series termination resistor (`22` to `33` ohms). Please consult your CPU documentation.(this pin is optional.)

Example connector part numbers:

Manufacturer	**Vertical**	**Right Angle**
AMP/Tyco [http://www.amp.com]	`103309-2`	`103311-2`
3M [http://www.3m.com]	`2514-6002UB`	`2514-5002UB`

Renesas H-UDI



Pin description and termination

Pin	Name	Description	Direction	Notes
1	`TCK`	JTAG TAP Clock	To Target	
2	`NC`	No Connection		
3	`nTRST`	JTAG TAP machine reset	To Target	
5	`TDO`	JTAG Test Data Out		
7	`ASEBRK`	Run/Break mode indicator	From Target	
8	`TVcc`	Target voltage reference	From Target	
9	`TMS`	JTAG TAP Machine State	To Target	
11	`TDI`	JTAG Test Data In	To Target	
13	`nRST`	Reset Status	From Target	1
4, 6, 10, 12, 14	`GND`	Ground reference		

Notes:

The H-UDI specification requires a reset line from the target that indicates if the target is currently in a reset state (signal `nRST` from pin 13). If a target instead presents a `nRST` pin to the probe that can be used as an input to the board to cause a target reset, the probe can be reconfigured to use that signal to cause a system-wide reset by setting the **allow_hard_rst** option to **on**. For more information about this option, see “CPU Families and Debug Connectors” on page 237

SuperTrace Probe Trace Pod Pin Assignments

This section describes pin assignments and electrical characteristics for the SuperTrace Probe trace pods.

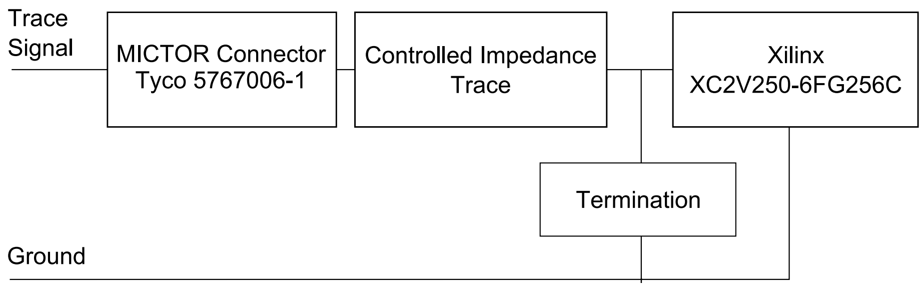
The input pins on SuperTrace Probe V1 active pods have the following electrical characteristics:

- The input capacitance is 10 pF.
- Data lines are routed on 0.005 inch tracks on FR4 over a continuous plane. The length of the data lines is 1.152 +/- 0.030 inches.
- Data and clock signals are AC terminated to circuit ground with 330 Ohms + 5.6 pF in series. This is subject to modification or removal.

The input pins on SuperTrace Probe V3 active pods have the following electrical characteristics:

- The input capacitance is 10 pF.
- Data and clock lines are routed on impedance-controlled, length-matched microstrip and stripline structures with a target impedance of 50 Ohms +/- 10%. Total track lengths are 37 mm +/- 2 mm.
- All data and clock signals are resistively terminated to ground with a 10 K pull-down.

Trace Input Equivalent Load



All probe inputs have light pull-downs to avoid completely floating signals. The strength of these pull-downs varies slightly between different models of Green Hills debug hardware, but is 500K to 1Mohm on current hardware models of STPv3 and Green Hills Probe v3. For more specific information about pull-downs of older hardware models, contact Green Hills support.

ARM ETM/PTM/CoreSight MICTOR

The following table describes pin assignments for the SuperTrace Probe trace pod for ARM ETM/PTM/CoreSight MICTOR. Connect unused trace inputs to circuit ground. For example, if you are implementing an 8-bit trace port, connect pins 23, 25, 27, 29, 31, 33, 35, and 37 to ground.

Pin	Signal Name		Type	Notes
	ETMv1	CoreSight/ETMv3		
1	`n/c`	`n/c`		
2	`n/c`	`n/c`		
3	`n/c`	`n/c`		
4	`n/c`	`n/c`		
5	`GND`	`GND`		3
6	`TRACECLK`	`TRACECLK`	From Target	
7	`DBGREQ`	`DBGREQ`	To Target	4
8	`DBGACK`	`DBGACK`	From Target	4
9	`nSRST`	`nSRST`	Open-drain	
10	`EXTTrig`	`EXTTrig`	To Target	4
11	`TDO`	`TDO/SWO`	From Target	6
12	`VTref`	`VTref`	From Target	5
13	`RTCK`	`RTCK`	From Target	
14	`Vsupply`	`Vsupply`	Bidirectional	5
15	`TCK`	`TCK`	To Target	2
16	`TRACEPKT[7]`	`TRACEDATA[7]`	From Target	
17	`TMS`	`TMS`	To Target	2, 6
18	`TRACEPKT[6]`	`TRACEDATA[6]`	From Target	
19	`TDI`	`TDI/SWDIO`	To Target	
20	`TRACEPKT[5]`	`TRACEDATA[5]`	From Target	
21	`nTRST`	`nTRST`	To Target	
22	`TRACEPKT[4]`	`TRACEDATA[4]`	From Target	
23	`TRACEPKT[15]`	`TRACEDATA[15]`	From Target	1
24	`TRACEPKT[3]`	`TRACEDATA[3]`	From Target	

Pin	Signal Name		Type	Notes
	ETMv1	CoreSight/ETMv3		
25	`TRACEPKT[14]`	`TRACEDATA[14]`	From Target	1
26	`TRACEPKT[2]`	`TRACEDATA[2]`	From Target	
27	`TRACEPKT[13]`	`TRACEDATA[13]`	From Target	1
28	`TRACEPKT[1]`	`TRACEDATA[1]`	From Target	
29	`TRACEPKT[12]`	`TRACEDATA[12]`	From Target	1
30	`TRACEPKT[0]`	`GND`	From Target	3
31	`TRACEPKT[11]`	`TRACEDATA[11]`	From Target	1
32	`TRACESYNC`	`GND`	From Target	3
33	`TRACEPKT[10]`	`TRACEDATA[10]`	From Target	1
34	`PIPESTAT[2]`	`PULL-UP`	From Target	3
35	`TRACEPKT[9]`	`TRACEDATA[9]`	From Target	1
36	`PIPESTAT[1]`	`TRACECTL`	From Target	
37	`TRACEPKT[8]`	`TRACEDATA[8]`	From Target	1
38	`PIPESTAT[0]`	`TRACEDATA[0]`	From Target	

Notes:

1. For ETMv1, trace inputs in this group can be used for single-core tracing, or for the second CPU in a compatible dual-core target.
2. On STPv1, legacy STPv3 pod, and Green Hills Probe v2/v3, the probe adds serial termination of 33 ohms when using the ARM-20 legacy TTM and ARM-20-to-MICTOR pin adapter. The STPv3 TE pod has variable termination of these signals (firmware-controlled by the documented **jtag_drive** option). The Green Hills Probe v3 TE system has no series termination of these signals. Customers do not need to add termination of their own to these signals, however if the device does not have suitable pull-up/down resistors on-device, we recommend a very light (>100k ohms) pull-up on `TMS` and a similarly light pull-down on `TCK` to avoid unexpected debug logic activity when no probe is connected.
3. For CoreSight and direct ETMv3 systems, this pin is not connected and may display as `n/c` in the output of the **jp** command. The designation listed in the table is recommended by the specification.

4. `DBGACK` and `DBGREQ` are unnecessary and typically not used. Customers should normally connect `DBGACK` to ground and leave `DBGREQ` unconnected. `EXTTrig` is also not commonly used, and can be left unconnected.
5. New probes draw negligible current from both `VTRef` and `VSupply`. The STPv1 and STPv3 legacy pods may draw as much as 5mA from `VSupply`, but as long as `VTRef` is connected, probe operation will not be impacted if `VSupply` cannot meet that current demand. Only one of `VTRef` or `VSupply` needs to be connected via pull-up to the JTAG/SWD reference voltage of the board in order to enable power and voltage level detection on Green Hills debug hardware, but the ARM specifications indicate that both should be connected for maximum debug device compatibility.
6. In SWD and cJTAG two-wire modes, `TDI` and `TDO` are unused and `TMS` becomes bidirectional.

The public ARM documentation for the CoreSight TPIU has further information about SoC-level design considerations for trace of this type.

ARM ETMv3/PTM/CoreSight MIPI-60

The following table describes pin assignments for the SuperTrace Probe trace pod for ARM CoreSight/ETMv3/PTM using a MIPI-60 connector. Connect unused trace inputs to circuit ground. For example, if you are implementing an 8-bit trace port, connect pins 35, 37, 39, 41, 43, 45, 47, and 49 to ground.

Pin	Signal Name		Direction	Notes
	MIPI-60	CoreSight		
1	`VREF_DEBUG`	`VREF_DEBUG`	From Target	
2	`TMS`	`TMS`	To Target	1
3	`TCK`	`TCK`	To Target	1
4	`TDO`	`TDO/SWO`	From Target	2
5	`TDI`	`TDI/SWDIO`	To Target	2
6	`nRESET`	`nRESET`	To Target	3
7	`RTCK`	`RTCK`	From Target	2
8	`TRST_PD`	`n/c`		
9	`nTRST`	`nTRST`	To Target	
10	`TRIGIN`	`TRIGIN`	To Target	
11	`TRIGOUT`	`TRIGOUT`	From Target	
12	`VREF_TRACE`	`VREF_TRACE`	From Target	
13	`TRC_CLK0`	`TRC_CLK`	From Target	
14	`TRC_CLK1`	`n/c`		
15	`PRESENCE_DET`	`n/c`		
16	`GND`	`GND`		
17	`TRC_DATA[0]`	`n/c`		4
18	`TRC_DATA[20]`	`n/c`		
19	`TRC_DATA[1]`	`TRC_DATA[0]`	From Target	
20	`TRC_DATA[21]`	`n/c`		
21	`TRC_DATA[2]`	`TRC_DATA[1]`	From Target	
22	`TRC_DATA[22]`	`n/c`		
23	`TRC_DATA[3]`	`TRC_DATA[2]`	From Target	
24	`TRC_DATA[23]`	`n/c`		

Pin	Signal Name		Direction	Notes
	MIPI-60	CoreSight		
25	TRC_DATA[4]	TRC_DATA[3]	From Target	
26	TRC_DATA[24]	n/c		
27	TRC_DATA[5]	TRC_DATA[4]	From Target	
28	TRC_DATA[25]	n/c		
29	TRC_DATA[6]	TRC_DATA[5]	From Target	
30	TRC_DATA[26]	n/c		
31	TRC_DATA[7]	TRC_DATA[6]	From Target	
32	TRC_DATA[27]	n/c		
33	TRC_DATA[8]	TRC_DATA[7]	From Target	
34	TRC_DATA[28]	n/c		
35	TRC_DATA[9]	TRC_DATA[8]	From Target	
36	TRC_DATA[29]	n/c		
37	TRC_DATA[10]	TRC_DATA[9]	From Target	
38	TRC_DATA[30]	n/c		
39	TRC_DATA[11]	TRC_DATA[10]	From Target	
40	TRC_DATA[31]	n/c		
41	TRC_DATA[12]	TRC_DATA[11]	From Target	
42	TRC_DATA[32]	n/c		
43	TRC_DATA[13]	TRC_DATA[12]	From Target	
44	TRC_DATA[33]	n/c		
45	TRC_DATA[14]	TRC_DATA[13]	From Target	
46	TRC_DATA[34]	n/c		
47	TRC_DATA[15]	TRC_DATA[14]	From Target	
48	TRC_DATA[35]	n/c		
49	TRC_DATA[16]	TRC_DATA[15]	From Target	
50	TRC_DATA[36]	n/c		
51	TRC_DATA[17]	n/c		
52	TRC_DATA[37]	n/c		
53	TRC_DATA[18]	n/c		

Pin	Signal Name		Direction	Notes
	MIPI-60	**CoreSight**		
54	`TRC_DATA[38]`	`n/c`		
55	`TRC_DATA[19]`	`n/c`		
56	`TRC_DATA[39]`	`n/c`		
57	`GND`	`GND`		
58	`GND`	`GND`		
59	`TRC_CLK3`	`n/c`		
60	`TRC_CLK2`	`n/c`		

See the CoreSight, ETM, PTM, or MIPI specification for more information.

In SWD mode, `nTRST` and `TDI` are unused, `TMS` becomes `SWDIO` and is bidirectional, and `TDO` becomes `SWO` and is unused.

Notes:

1. Terminated with 33 ohms in series with output signal.
2. Weak pull-up resistors enabled on these inputs in FPGA.
3. Configured as an open-drain output.
4. For ETMv3, this pin is not connected, but is in the MIPI specification as `TRACE_CTL`.

PowerPC 405/440/460 MICTOR

The following table describes pin assignments for the SuperTrace Probe trace pod for PowerPC. Connect unused trace inputs to circuit ground.

Pin	Signal Name		Direction	Notes
	PPC405	PPC440/460		
1	n/c	n/c		
2	n/c	n/c		
3	n/c	n/c		
4	n/c	n/c		
5	GND	GND		7
6	TrcClk	TrcClk	From Target	
7	HALT	HALT	To Target	6
8	n/c	n/c	From Target	
9	n/c	n/c	To Target	
10	n/c	n/c	To Target	
11	TDO	TDO	From Target	3
12	Vref	Vref	From Target	4, 5
13	n/c	n/c	From Target	
14	n/c	n/c	From Target	
15	TCK	TCK	To Target	2, 6
16	n/c	n/c	From Target	
17	TMS	TMS	To Target	2, 6
18	n/c	n/c	From Target	
19	TDI	TDI	To Target	2, 6
20	n/c	n/c	From Target	
21	nTRST	nTRST	To Target	
22	n/c	n/c	From Target	
23	n/c	n/c	From Target	
24	TS1O	ES4	From Target	
25	n/c	BS0/BR0	From Target	1
26	TS2O	TS0	From Target	

Pin	Signal Name		Direction	Notes
	PPC405	**PPC440/460**		
27	`n/c`	`BS1/BR1`	From Target	1
28	`TS1E`	`TS1`	From Target	
29	`n/c`	`BS2/BR2`	From Target	1
30	`TS2E`	`TS2`	From Target	
31	`n/c`	`ES0`	From Target	1
32	`TS3`	`TS3`	From Target	
33	`n/c`	`ES1`	From Target	1
34	`TS4`	`TS4`	From Target	
35	`n/c`	`ES2`	From Target	1
36	`TS5`	`TS5`	From Target	
37	`n/c`	`ES3`	From Target	1
38	`TS6`	`TS6`	From Target	

1. Trace inputs in this group can be used for single-core tracing, or for the second CPU in a compatible dual-core target.
2. Terminated with 33 ohms in series with output signal.
3. Weak pull-up resistors enabled on these inputs in FPGA.
4. Terminated with 48.5k ohms resistor to circuit ground.
5. Connect to target I/O power supply through a 1k Ohms (or less) resistor.
6. Include a 10k Ohms pull-up resistor on the target board for this signal.
7. The probe makes no use of this pin. It is internally disconnected.

PowerPC 405 20-Pin Header

The following table describes pin assignments for the SuperTrace Probe trace pod for the 20-pin PowerPC 405 header connector.

Pin	Signal Name	Direction
1	`n/c`	
2	`n/c`	
3	`TrcClk`	From Target
4	`n/c`	
5	`n/c`	
6	`n/c`	
7	`n/c`	
8	`n/c`	
9	`n/c`	
10	`n/c`	
11	`n/c`	
12	`TS1O`	From Target
13	`TS0/TS2O`	From Target
14	`TS1/TS2E`	From Target
15	`TS2/TS2E`	From Target
16	`TS3`	From Target
17	`TS4`	From Target
18	`TS5`	From Target
19	`TS6`	From Target
20	`GND`	

PowerPC 55xx/56xx/57xx Nexus

PowerPC Nexus is the standard connector used to debug PowerPC 55xx/56xx/57xx targets. This connector deviates slightly from the Nexus standard. For information about standard Nexus connections, see “Generic Nexus” on page 271.

Pin	Name	Description	Direction	Notes
1	`n/c`			
2	`n/c`			
3	`n/c`			
4	`n/c`			
5	`MDO9`	Trace Data	From Target	
6	`CLKOUT`	System Clock	From Target	
7	`n/c`			
8	`MDO8`	Trace Data	From Target	
9	`NRESET`	Target Reset, Active Low	To Target	
10	`EVTI`	Nexus event in	To Target	
11	`TDO`	JTAG Test Data Out	From Target	
12	`VREF`	Target Voltage Reference	From Target	
13	`MDO10`	Trace Data	From Target	
14	`NRDY`	Nexus Ready	Bidirectional	
15	`TCK`	JTAG TAP Clock	To Target	1
16	`MDO7`	Trace Data	From Target	
17	`TMS`	JTAG TAP Machine State	To Target	1
18	`MDO6`	Trace Data	From Target	
19	`TDI`	JTAG Test Data In	To Target	1
20	`MDO5`	Trace Data	From Target	
21	`NTRST`	JTAG TAP Reset, Active Low	To Target	2
22	`MDO4`	Trace Data	From Target	
23	`MDO11`	Trace Data	From Target	
24	`MDO3`	Trace Data	From Target	
25	`n/c`			

Pin	Name	Description	Direction	Notes
26	`MDO2`	Trace Data	From Target	
27	`n/c`			
28	`MDO1`	Trace Data	From Target	
29	`n/c`			
30	`MDO0`	Trace Data	From Target	
31	`n/c`			
32	`EVTO`	Nexus Event Out	From Target	
33	`n/c`			
34	`MCKO`	Trace Clock	From Target	
35	`n/c`			
36	`MSEO1`	Nexus Message Start/End	From Target	
37	`n/c`			
38	`MSEO0`	Nexus Message Start/End	From Target	

Notes:

1. Pull up to a logical high with a resistor.
2. The `NTRST` signal connects to the processor's `JCOMP` signal. The MPC5500 microprocessors include a weak internal pull-down on the `/JCOMP` pin. Board designers can optionally strengthen this pull-down by adding a resistor of 4.7K from `JCOMP` to ground.

Example connector part numbers:

Manufacturer	Vertical Mount
AMP/Tyco [http://www.amp.com]	`767054-1`

ColdFire

For information about SuperTrace Probe ColdFire connectors, see "ColdFire BDM" on page 259.

SuperTrace Probe TE Trace Pod and Green Hills Probe TE Adapter Pin Assignments

Nexus HP50

The following table describes pin assignments for the SuperTrace Probe TE trace pod and Green Hills Probe TE adapter for Nexus HP50, which uses a variation of Samtec ERF8/ERM8-025.

Pin	Signal Name	Direction	Notes
1	`MSEO0`	From Target	1
2	`TVREF`	From Target	
3	`MSEO1`	From Target	1
4	`TCK`	To Target	
5	`GND`		
6	`TMSC`	Bidirectional	
7	`MDO0`	From Target	1
8	`TDI`	To Target	
9	`MDO1`	From Target	1
10	`TDO`	From Target	
11	`GND`		
12	`NTRST`	To Target	
13	`MDO2`	From Target	1
14	`NRDY`	From Target	7
15	`MDO3`	From Target	1
16	`EVTI`	To Target	5
17	`GND`		
18	`EVTO`	From Target	6
19	`MCKO`	From Target	1
20	`HRST`	Open-drain	
21	`MDO4`	From Target	1
22	`TOOLIO0[GEN_IO0]`	Bidirectional	3

Pin	Signal Name	Direction	Notes
23	GND		
24	GND		
25	MDO5	From Target	1
26	CLKOUT	From Target	4
27	MDO6	From Target	1
28	MDI1	To Target	2
29	GND		
30	GND		
31	MDO7	From Target	1
32	MDI2	To Target	2
33	MDO8	From Target	1
34	MDI3	To Target	2
35	GND		
36	GND		
37	MDO9	From Target	1
38	BOOTCFG[GEN_IO4]	Bidirectional	3
39	MDO10	From Target	1
40	TOOLIO5[GEN_IO5]	Bidirectional	3
41	GND		
42	GND		
43	MDO11	From Target	1
44	MDO13	From Target	1
45	MDO12	From Target	1
46	MDO14	From Target	1
47	GND		
48	GND		
49	MDO15	From Target	1
50	n/c		

Notes:

1. Trace inputs (the Nexus parallel AUX channel of signals `MSEOx`, `MCKO`, and `MDOx`) are only used when the adapter is connected to the TE trace pod, and are unused when the adapter is connected to a Green Hills Probe. Connect any unused trace inputs to circuit ground.
2. The `MDIx` AUX channel to the target is unused by Green Hills hardware devices and can be left unconnected.
3. Pins named `GEN_IOx` in the Nexus standard are given more specific names by some board/SoC implementations. Green Hills connector definitions use the specific names by default, with the `GEN_IOx` names given in brackets. These pins are available as GPIOs for use with all probe types, but are not driven by the probe by default.
4. `CLKOUT` is not used by Green Hills hardware devices and can be tied to ground, left unconnected, or connected to the `CLKOUT` pin (present only on older Nexus devices) at the board implementer's option.
5. `EVTI` can be triggered through GPIO commands, but is not used by the firmware or tools software.
6. `EVTO` output from the target is used to detect triggers in some cases.
7. `NRDY` is not used by the firmware and should be grounded if not connected to the equivalent target signal.

SuperTrace Probe High-Speed Serial Trace Pin Assignments

ARM CoreSight HSSTP (ERF8-020)

Description	Dir	Name	Number	Number	Name	Dir	Description
Aurora Lane 4	<-	TXP4	1	2	VREF	->	Voltage Reference
Aurora Lane 4	<-	TXN4	3	4	TCK	<-	JTAG Clock
		GND	5	6	GND		
Aurora Lane 2	<-	TXP2	7	8	TMS	<-	JTAG State
Aurora Lane 2	<-	TXN2	9	10	TRST	<-	JTAG Reset
		GND	11	12	GND		
Aurora Lane 0	<-	TXP0	13	14	TDI	<-	JTAG Test Data
Aurora Lane 0	<-	TXN0	15	16	TDO	->	JTAG Test Data
		GND	17	18	GND		
Not Used	->	REFCLKP	19	20	RESET	<-	Target Reset
Not Used	->	REFCLKN	21	22	DBGRQ	<-	Debug Request
		GND	23	24	GND		
Aurora Lane 1	<-	TXP1	25	26	DBGACK	->	Debug Acknowledge
Aurora Lane 1	<-	TXN1	27	28	RTCK	->	Target Clock
		GND	29	30	GND		
Aurora Lane 3	<-	TXP3	31	32	TRIGIN	<-	Trigger In
Aurora Lane 3	<-	TXN3	33	34	TRIGOUT	->	Trigger out
		GND	35	36	N/C		
Aurora Lane 5	<-	TXP5	37	38	N/C		
Aurora Lane 5	<-	TXN5	39	40	N/C		

QorIQ Nexus Trace (ERF35)

Description	Dir	Name	Number		Name	Dir	Description
Aurora Lane 0	<-	`TXP0`	1	2	`VREF`	->	Voltage Reference
Aurora Lane 0	<-	`TXN0`	3	4	`TCK`	<-	JTAG Clock
		`GND`	5	6	`TMS`	<-	JTAG State
Aurora Lane 1	<-	`TXP1`	7	8	`TDI`	<-	JTAG Test Data
Aurora Lane 1	<-	`TXN1`	9	10	`TDO`	->	JTAG Test Data
		`GND`	11	12	`TRST`	<-	JTAG Reset
Aurora Lane 0	->	`RXP0`	13	14	`HALT`	<-	Target Halt
Aurora Lane 0	->	`RXN0`	15	16	`EVTI`	<-	Event In
		`GND`	17	18	`EVTO`	->	Event Out
Aurora Lane 1	->	`RXP1`	19	20	`GEN_IO_3`	<->	Generic IO
Aurora Lane 1	->	`RXN1`	21	22	`RESET`	<-	Target Reset
		`GND`	23	24	`GND`		
Aurora Lane 2	<-	`TXP2`	25	26	`REFCLKP`	<-	Not Used
Aurora Lane 2	<-	`TXN2`	27	28	`REFCLKN`	<-	Not Used
		`GND`	29	30	`GND`		
Aurora Lane 3	<-	`TXP3`	31	32	`RDY`	<->	Not Used
Aurora Lane 3	<-	`TXN3`	33	34	`NSRST`	<-	Not Used
		`GND`	35	36	`GND`		
Aurora Lane 2	->	`RXP2`	37	38	`N/C`		
Aurora Lane 2	->	`RXN2`	39	40	`N/C`		
		`GND`	41	42	`N/C`		
Aurora Lane 3	->	`RXP3`	43	44	`N/C`		
Aurora Lane 3	->	`RXN3`	45	46	`N/C`		
		`GND`	47	48	`N/C`		
Aurora Lane 4	<-	`TXP4`	49	50	`N/C`		
Aurora Lane 4	<-	`TXN4`	51	52	`N/C`		
		`GND`	53	54	`N/C`		
Aurora Lane 5	<-	`TXP5`	55	56	`N/C`		
Aurora Lane 5	<-	`TXN5`	57	58	`N/C`		

Description	Dir	Name	Number	Number	Name	Dir	Description
		`GND`	59	60	`N/C`		
Aurora Lane 6	<-	`TXP6`	61	62	`N/C`		
Aurora Lane 6	<-	`TXN6`	63	64	`N/C`		
		`GND`	65	66	`N/C`		
Aurora Lane 7	<-	`TXP7`	67	68	`N/C`		
Aurora Lane 7	<-	`TXN7`	69	70	`N/C`		

QorIQ Nexus Trace (ERF11)

Description	Dir	Name	Number	Number	Name	Dir	Description
Aurora Lane 0	<-	`TXP0`	1	2	`VREF`	->	Voltage Reference
Aurora Lane 0	<-	`TXN0`	3	4	`TCK`	<-	JTAG Clock
		`GND`	5	6	`TMS`	<-	JTAG State
Aurora Lane 1	<-	`TXP1`	7	8	`TDI`	<-	JTAG Test Data
Aurora Lane 1	<-	`TXN1`	9	10	`TDO`	->	JTAG Test Data
		`GND`	11	12	`TRST`	<-	JTAG Reset
Aurora Lane 0	->	`RXP0`	13	14	`GEN_IO_0`	<->	Generic IO
Aurora Lane 0	->	`RXN0`	15	16	`EVTI`	<-	Event In
		`GND`	17	18	`EVTO`	->	Event Out
Aurora Lane 1	->	`RXP1`	19	20	`GEN_IO_3`	<->	Generic IO
Aurora Lane 1	->	`RXN1`	21	22	`RESET`	<-	Target Reset

Appendix C

CPU-Specific Bit Tables

Contents

The tables in this appendix list TLB and cache tag information according to CPU type. The tables are divided into two sections:

- "Virtual and Physical Tags" on page 300
- "Cache Tags" on page 309

In each of these tables, bit 0 refers to the least significant bit. For more information about the *vtag* and *ptag* parameters, see the **tlbw** command in "Cache, Memory, and TLB Commands" on page 191.

Virtual and Physical Tags

Cortex-A15					
L1-I		**L1-D**		**L2**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
`97`	`V`	`101`	`Sec`	`99`	`Par`
`96..95`	`Share`	`98`	`Hyp`	`97`	`Hyp`
`85`	`NG`	`91..90`	`Share`	`96`	`Sec`
`83..82`	`VAMSID`	`89..82`	`MAIR`	`95`	`NS`
`81..74`	`VMID`	`81..74`	`VMID`	`65`	`DnS`
`73..66`	`ASID`	`73..66`	`ASID`	`94..87`	`VMID`
`65..58`	`MAIR`	`65..62`	`DomID`	`86..79`	`ASID`
`55..52`	`DomID`	`49`	`V`	`33..31`	`Size`
`48`	`NS`	`48`	`NS`	`29`	`XN`
				`28`	`PXN`
				`14`	`nG`
				`13..10`	`Dom`
				`9`	`OS`
				`8`	`IS`
				`7..0`	`MAIR`

MIPS32			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`39..32`	`ASID`	`5..3`	`C`
		`2`	`D`
		`1`	`V`
`31..0`	`PageMask`	`0`	`G`

MIPS64			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`95..94`	`R`	`5..3`	`C`
		`2`	`D`
`39..32`	`ASID`	`1`	`V`
`31..0`	`PageMask`	`0`	`G`

MIPS 4KS			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`39..32`	`ASID`	`31`	`RI`
		`30`	`XI`
		`5..3`	`C`
		`2`	`D`
		`1`	`V`
`31..0`	`PageMask`	`0`	`G`

PowerPC 405			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`13..11`	`SIZE`	`9`	`EX`
		`8`	`WR`
		`7..4`	`ZSEL`
`10`	`V`	`3`	`W`
`9`	`E`	`2`	`I`
`8`	`U0`	`1`	`M`
`7..0`	`PID`	`0`	`G`

PowerPC 440			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
31	U0*		
30	U1*		
29	U2*		
28	U3*		
25	V		
24	TS		
23..20	SIZE		
19..12	TID		
10	W		
9	I		
8	M		
7	G		
6	E		
5	UX		
4	UW		
3	UR		
2	SX		
1	SW		
0	SR	3..0	ERPN

* These tags are only available with probe firmware version 1.6 and later.

PowerPC 567x			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
26	VLE*		
25	V		
24	TS		
23..19	SIZE		
18..11	TID		
10	W		
9	I		
8	M		
7	G		
6	E		
5	UX		
4	SX	4	U3
3	UW	3	U2
2	SW	2	U1
1	UR	1	U0
0	SR	0	IPROT

PowerPC 55xx, 563x, and 566x			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
25	VLE*		
24	V		
23	TS		
22..19	SIZE		
18..11	TID		
10	W		
9	I		
8	M		
7	G		
6	E		
5	UX		
4	SX	4	U3
3	UW	3	U2
2	SW	2	U1
1	UR	1	U0
0	SR	0	IPROT

*This field only appears in the TLB entry for processors that support VLE.

PowerPC 85xx, P1xxx and P2xxx			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`24`	`V`		
`23`	`TS`		
`22..19`	`SIZE`		
`18..11`	`TID`		
`10`	`W`		
`9`	`I`		
`8`	`M`		
`7`	`G`	`7`	`SHAREN`
`6`	`E`	`6`	`X1`
`5`	`UX`	`5`	`X0`
`4`	`UW`	`4`	`U3`
`3`	`UR`	`3`	`U2`
`2`	`SX`	`2`	`U1`
`1`	`SW`	`1`	`U0`
`0`	`SR`	`0`	`IPROT`

QorIQ e500mc and e5500 cores (e.g., P4080, P3041, P5020)							
TLB1				TLB0			
Virtual tag (*vtag*)		Physical tag (*ptag*)		Virtual tag (*vtag*)		Physical tag (*ptag*)	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**	**Bit**	**Name**
`31`	`GS`			`29`	`GS`		
`25..30`	`LPID`			`27..28`	`LRU`		
`24`	`V`			`21..26`	`LPIDR`		
`23`	`TS`			`20`	`V`		
`19..22`	`SIZE`			`19`	`TS`		
`11..18`	`TID`			`11..18`	`TID`		
`10`	`W`			`10`	`W`		
`9`	`I`			`9`	`I`		
`8`	`M`			`8`	`M`		
`7`	`G`	`7`	`VF`	`7`	`G`		
`6`	`E`	`6`	`X0`	`6`	`E`	`6`	`VF`
`5`	`UR`	`5`	`X1`	`5`	`UR`	`5`	`X0`
`4`	`UW`	`4`	`U0`	`4`	`UW`	`4`	`X1`
`3`	`UX`	`3`	`U1`	`3`	`UX`	`3`	`U0`
`2`	`SR`	`2`	`U2`	`2`	`SR`	`2`	`U1`
`1`	`SW`	`1`	`U3`	`1`	`SW`	`1`	`U2`
`0`	`SX`	`0`	`IPROT`	`0`	`SX`	`0`	`U3`

QorIQ e6500-based targets (e.g., T4240)							
TLB1				**TLB0**			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**		**Virtual tag (*vtag*)**		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**	**Bit**	**Name**
33	GS			29	GS		
27..32	LPID			27..28	LRU		
26	V			21..26	LPIDR		
25	TS			20	V		
24	IND						
19..23	SIZE			19	TS		
11..18	TID			11..18	TID		
10	W			10	W		
9	I			9	I		
8	M			8	M		
7	G	7	VF	7	G		
6	E	6	X0	6	E	6	VF
5	UR	5	X1	5	UR	5	X0
4	UW	4	U0	4	UW	4	X1
3	UX	3	U1	3	UX	3	U0
2	SR	2	U2	2	SR	2	U1
1	SW	1	U3	1	SW	1	U2
0	SX	0	IPROT	0	SX	0	U3

Cache Tags

ARM 920, 926, and 946	
tag	
Bit	**Name**
`4`	`L`
`4`	`V`
`3`	`V3`
`3..2`	`Dirty`
`2`	`V2`

ARM 920			
TLB **Virtual tag (*vtag*)**		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`31..28`	`SIZE_C`		
`27`	`V`		
`26`	`PROT_FAULT`		
`24`	`PROT_FAULT`		
`23`	`DOMAIN_FAULT`		
`22`	`TLB_MISS`		
`21..6`	`Domain`		
`5`	`nC`		
`4`	`nB`		
`3..0`	`AP`	`3..0`	`SIZE_R2`

ARM 926			
TLB Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`4`	`V`	`7..4`	`Domain`
`3..0`	`SIZE`	`3..2`	`AP`
		`1`	`C`
		`0`	`B`

ARM1136			
Virtual tag (*vtag*)		**Physical tag (*ptag*)**	
Bit	**Name**	**Bit**	**Name**
`25`	`Global`		
`24..16`	`ASID`		
`15..14`	`AP3`		
`13..12`	`AP2`		
`11..10`	`AP1`		
`9`	`SPV`		
`8..5`	`Domain`	`9..6`	`SZ`
`4`	`XN`	`5..4`	`XRGN`
`3..1`	`RGN`	`3..1`	`AP`
`0`	`S`	`0`	`V`

ARM1136			
Data tag (*dtag*)		**Instruction tag (*itag*)**	
Bit	**Name**	**Bit**	**Name**
`2`	`D1`		
`1`	`D0`		
`0`	`V`	`0`	`V`

Cortex-A15					
Data Tag (*dtag*)		**Instruction Tag (*i2tag*)**		**Level 2 Tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
`28..27`	`MESI`	`28`	`NS`	`1..0`	`MESI`
`26`	`NS`	`29`	`V`	`27`	`NS`
`25..18`	`PAHIGH`	`30`	`P`	`28`	`I`
`38..32`	`ECC`	`27..20`	`PAHIGH`	`29`	`P`
				`26..19`	`PAHIGH`
				`32..32`	`ECC`

Cortex-A15	
CCN504 L3 Cache	
Bit	**Name**
`1..0`	`MESI`
`8`	`NS`
`39..32`	`PAHIGH`

LSI MIPS	
tag	
Bit	**Name**
`4`	`L`
`3`	`V3`
`2`	`V2`
`1`	`V1`
`0`	`V0`

MIPS 4K	
tag	
Bit	**Name**
`7..4`	`VALID`
`2`	`L`
`1`	`LRF`

MIPS 4KE	
Data tag (*dtag*)	
Bit	**Name**
7	V
6	D
5	L

MIPS 5K	
Data tag (*dtag*)	
Bit	**Name**
7..6	PState
5	L
0	P

MIPS 20K			
Data tag (*dtag*)		**Instruction tag (*itag*)**	
Bit	**Name**	**Bit**	**Name**
7..6	PState	49	BE
		48	G
		47..40	ASID
		7	PState
5	L	5	L
4	F	4	F

PowerPC 405			
Data tag (*dtag*)		**Instruction tag (*itag*)**	
Bit	**Name**	**Bit**	**Name**
2	D		
1	V	1	V
0	LRU	0	LRU

PowerPC 440			
Data tag (*dtag*)		**Instruction tag (*itag*)**	
Bit	**Name**	**Bit**	**Name**
`12`	`V`	`10`	`V`
`11..8`	`Dir`	`9`	`TS`
`7`	`U3`	`8`	`TD`
`6`	`U2`	`7..0`	`TID`
`5`	`U1`		
`4`	`U0`		
`3..0`	`TERA`		

PowerPC 55xx, 563x, and 566x	
Unified L1 tag (*tag*)	
Bit	**Name**
`2`	`L`
`1`	`D`
`0`	`V`

PowerPC 567x			
Data tag (*dtag*)		**Instruction tag (*itag*)**	
Bit	**Name**	**Bit**	**Name**
`2`	`L`		
`1`	`D`	`1`	`L`
`0`	`V`	`0`	`V`

PowerPC 603ev, 7xx, 7400, 7410, 82xx			
Data tag (*dtag*)		**Instruction tag (*itag*)**	
Bit	**Name**	**Bit**	**Name**
`1`	`V`		
`0`	`D`	`0`	`V`

PowerPC 7400, 7410					
2 MB Level 2 tag (*l2tag*)		**1 MB Level 2 tag (*l2tag*)**		**512 KB or 256 KB Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
`11`	`M3`	`5`	`M1`	`2`	`M`
`10`	`M2`	`4`	`M0`	`1`	`S`
`9`	`M1`	`3`	`S1`	`0`	`V`
`8`	`M0`	`2`	`S0`		
`7`	`S3`	`1`	`V1`		
`6`	`S2`	`0`	`V0`		
`5`	`S1`				
`4`	`S0`				
`3`	`V3`				
`2`	`V2`				
`1`	`V1`				
`0`	`V0`				

PowerPC 744x, 745x, 86xx							
Data tag (*dtag*)		**Instruction tag (*itag*)**		**Level 2 tag (*l2tag*)**		**Level 3 tag (*l3tag*) (For 7455 and 7457 targets only)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**	**Bit**	**Name**
`2`	`S`	`0`	`V`	`5`	`A1`	`15..12`	`sec3`
`1`	`V`			`4`	`A0`	`11..8`	`sec2`
`0`	`D`			`3..2`	`MESI1`	`7..4`	`sec1`
				`1..0`	`MESI0`	`3..0`	`sec0`

PowerPC 75x			
1 MB Level 2 tag (*l2tag*)		**512 KB or 256 KB Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**
7	D3	3	D1
6	D2	2	D0
5	D1	1	V1
4	D0	0	V0
3	V3		
2	V2		
1	V1		
0	V0		

PowerPC 85xx					
Data tag (*dtag*)		**Instruction tag (*itag*)**		**Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
3	V	1	V	4	V
2	L	0	L	3	IL
1	S			2	DL
0	D			1	T
				0	S

PowerPC QorIQ P1xxx and P2xxx					
Data tag (*dtag*)		**Instruction tag (*itag*)**		**Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
3	V	1	V	3	V
2	L	0	L	2	IL
1	S			1	DL
0	D			0	T

QorIQ e500mc and e5500 cores (e.g., P4080, P3041, P5020)					
Data tag (*dtag*)		**Instruction tag (*itag*)**		**Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
3	V	1	V	4	V
2	L	0	L	3	L
1	S			2	S
0	D			1	D
				0	N

QorIQ e6500-based targets (e.g., T4240)					
Data tag (*dtag*)		**Instruction tag (*itag*)**		**Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
1	V	1	V	12	V
0	L	0	L	11	L
				10	E
				9	S
				8	SO
				7	M
				6	N

IXP2350					
Data tag (*dtag*)		**Instruction tag (*itag*)**		**Level 2 tag (*l2tag*)**	
Bit	**Name**	**Bit**	**Name**	**Bit**	**Name**
12	valid	12	valid	31	valid
10..8	lock	10..8	lock	30	used
6..4	Iru	6..4	Iru	29	lock
3..0	Dirty			27..25	state

Appendix D

Troubleshooting and Usage Notes

Contents

Troubleshooting

If your probe is not communicating with your target properly, consult the steps provided in the *Getting Started* book for your probe. If you are still having problems communicating with your target, consult the information in one of the following sections.

Operating Ranges

The probe has the following temperature and humidity tolerances:

- Operating temperature: 32–122 F (0–50 C)
- Storage temperature: -4–149 F (-20–65 C)
- Relative humidity: less than 90%, non-condensing

Using vb to Diagnose Connection Problems

The **vb** command helps diagnose JTAG problems by:

1. putting JTAG into bypass mode
2. sending data through the target's JTAG test data in (TDI) line
3. checking for matching data returned on the target's test data out (TDO) line

Note

The **vb** command works with JTAG targets only. It does not work on PowerPC 5xx, PowerPC 8xx and ColdFire targets.

If **vb** is successful, the result is:

```
Test passed.
```

When **vb** fails, it issues the following error:

```
Error 13 (test failed): Test Failed: in=input, out=output, i=iter
```

Different values for *output* indicate different potential problems with the probe's configuration:

- `0x0` — Indicates that TDO is held low. Usually, this means that your target's power is off, or your probe's logic high level is set incorrectly. For information about solving these issues, see the troubleshooting section in the *Getting Started* book for your probe.
- `0xffffffff` — Indicates that TDO is held high. Usually, this means that the TAP reset line is asserted or `logic_high` is set too low.
- another number — Indicates that the probe is not reliably detecting transitions between logic high and logic low. Try the following:
  - adjusting the JTAG clock with the **set clock** command
  - adjusting the logic high level with the **set logic_high** command
  - detecting your target with the `detect` command
  - checking for line noise or hardware connection problems

Diagnosing Reset Line Problems

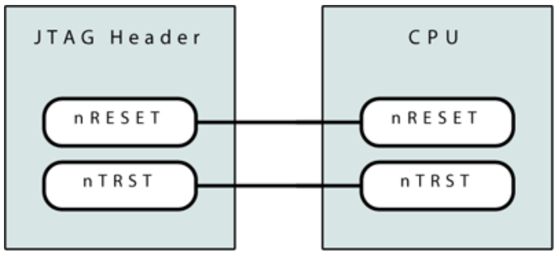
Green Hills Debug Probes are aware of two different reset lines:

- CPU reset line (for example, `nRESET`) — resets the core
- TAP reset line (for example, `nTRST`) — resets the JTAG TAP controller

These reset lines should not be wired, logically ORed, or otherwise tied together, because the probe needs to be able to control them individually to perform a precise reset. For example:

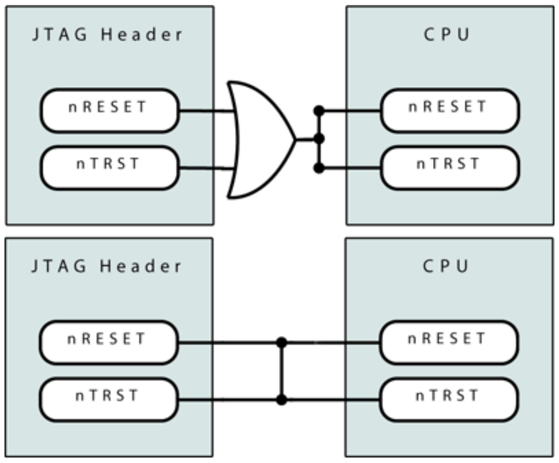
Correct

The TAP reset line on the target's JTAG header is wired directly to the TAP reset line on the CPU, and the CPU reset line on the header is wired directly to the CPU reset line on the CPU.



Incorrect

The TAP reset line and CPU reset line must not be logically ORed or wired together.



If you are getting errors when trying to reset your board and believe this may be the problem, run the following commands in a serial terminal:

```
> jp cpu_reset_line 0
> vb
> jp cpu_reset_line 1
> vb
```

where *cpu_reset_line* is the name of your target's CPU reset line. If **vb** fails the first time but passes the second time, that most likely indicates that the lines are tied together.

If you cannot fix this problem in your hardware, use the following command to tell the probe that the lines are tied together:

```
> set target_reset_pin resets_tap
```

For more information about this setting, see “Setting Reset Pin and JTAG TAP Interaction” on page 87.

Some Board Components Retain Power After Target Power is Removed

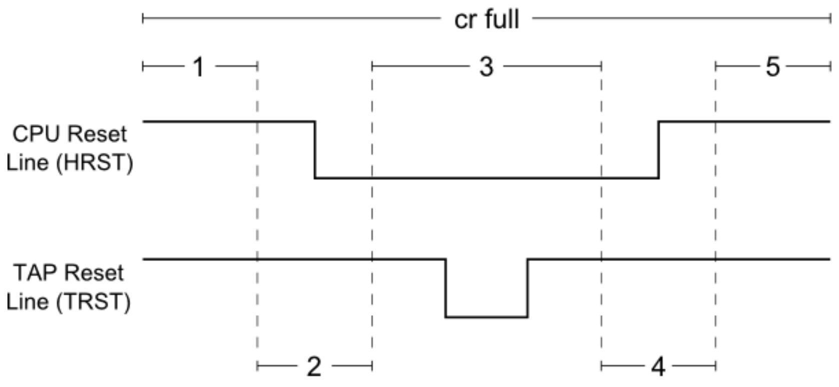
If you remove power from your target and some of its components are still powered on, it is possible that the probe is supplying power to those components over the debug connection. To fix this problem, always tri-state your probe before turning on your target's power (see “The User Button” on page 5).

Resetting a Single Core in a Multi-Core System

A typical reset sequence has the following five phases:

1. Prepare the core for reset.
2. Assert the CPU reset line (for example, `nHRESET` or `nRESET`).
3. Perform all actions required while the CPU reset line is asserted. On JTAG targets, this performs the JTAG test access port (TAP) reset, including toggling the TAP reset line (for example, `nTRST`).
4. De-assert the CPU reset line.
5. Perform any post-reset initialization.

When you reset a core with the **tr d** or **cr full** commands, the probe performs all five of these phases on the selected core:



While this works well on targets with a single core, many multi-core targets require you to use special commands to ensure that only the CPU reset line of the correct core is asserted.

If your board design allows the CPU reset line to be asserted on individual cores, use the following sequence of commands to reset a single core:

```
> t core     ; select a core
> cr pre     ; phase 1 (prepare)
> cmd_a      ; phase 2 (assert CPU reset line for that core)
> cr tap     ; phase 3 (actions while reset is asserted)
> cmd_b      ; phase 4 (de-assert CPU reset line for that core)
> cr post    ; phase 5 (post-reset initialization)
```

Where the commands *cmd_a* and *cmd_b* depend on your design.

If your board design allows you to tie a core's CPU reset line to the probe's CPU reset line, you can simplify these steps to:

```
> t core      ; select a core
> cmd_c       ; tie probe CPU reset line to core CPU reset line
> cr full     ; phases 1-5
```

Recovering From Accidental USB Disconnection

You should not unplug the probe's USB cable while connected to the probe with **mpserv** over USB. If the cable is accidentally disconnected, use the following procedure to reconnect:

1. Disconnect **mpserv** from within MULTI.
2. If the USB cable is still plugged into your probe, unplug it.
3. Plug the USB cable into your host machine, then into your probe.
4. Reconnect to your probe with **mpserv**.

Grounding Your Probe

Electrical noise might affect your probe if you are only using the Ethernet host connection, and connected to a target that is not referenced to earth ground. If your probe is not properly grounded, you may experience the following problems:

- Ambiguity when detecting your target adapter. This is usually seen when using `set adapter auto` with ARM, ARC, and ADI adapters.
- Disrupted JTAG operations

To ensure that your probe is properly grounded, connect a USB or RS-232 cable from your probe to an earth grounded host machine, or use an earth ground referenced power supply for the probe. For more information, contact Green Hills support.

Changing the Target Communication Protocol

On targets that support multiple target communication protocols, the **debug_type** probe configuration option controls which protocol is in use. While the **debug_type** option can be changed at anytime, some targets cannot switch protocols without cycling the power. If the probe loses control of the target after changing **debug_type**, power cycle the target.

ERROR 71 After Changing the Probe TTM

This problem can have several symptoms:

- Most probe commands return an ERROR 71.
- The LED on the TE adapter is red.
- Two LEDs on the front of the probe blink regularly, indicating that the outputs are disabled, and any attempt to enable them has no effect.

This problem is likely due to a misconfigured adapter setting. If the adapter setting was not `auto` for a previously used (legacy) TTM adapter, and then a TE TTM is connected, the outputs will not be enabled until the probe's adapter setting is set to `auto`. Running **detect** will also attempt to configure your adapter setting correctly.

Usage Notes

Breakpoint Usage Notes

Setting Multiple Breakpoint Types on a Single Assembly Statement

When there is more than one type of breakpoint (for example, a software breakpoint and a hardware data breakpoint) set on a single assembly statement, the probe only hits one of the breakpoints. When continuing execution from that point, the probe does not hit the others. Because the type of breakpoint that is hit depends on your target, we do not recommend that you rely on this behavior.

Target-Specific Hardware Breakpoint Support

This section describes target-specific hardware breakpoint support, including but not limited to:

- The number of available breakpoints.
- Support for range breakpoints:
  - *arbitrary range* — Covers any number of consecutive bytes and can begin at any address.
  - *aligned range* — Has a length which is a power of two, and begins at an address aligned to its length.
- Support for mask breakpoints:
  - *arbitrary mask* — Selects any set of addresses, contiguous or non-contiguous. For example, `0xffffff0f` or `0xfffffff0`.
  - *consecutive mask* — Selects a contiguous range of addresses. For example, `0xfffffff0`, but not `0xffffff0f`.
- Breakpoint hit detection:
  - *address* — The address of the access itself must fall within the mask, range, or address of the breakpoint to hit that breakpoint. For example, a 4-byte access to address `0x20` will hit a 2-byte breakpoint set at `0x1F` or `0x20`, but it will not hit a breakpoint set at `0x21`.

  - *range* — Any address affected by the access that falls within the mask, range, or address of the breakpoint will hit the breakpoint. For example, a 4-byte access to address `0x20` will hit any 2-byte breakpoint set between `0x1F` and `0x23`, inclusive.

If a breakpoint specifies both a range and a mask, the behavior is unspecified.

Data Hardware Breakpoint Support

The following table describes data hardware breakpoint support for a variety of targets:

Target	#	Ranges	Masks	Hit Detection	Notes
ARM7, ARM9	1	not supported	arbitrary	address	1, 2, 7
ARM11, Cortex-A8, Cortex-R4, PJ4	2	aligned	consecutive	range	2
Cortex-A5, Cortex-A9	2	aligned	consecutive	address	2, 11
Cortex-A15	4	not supported	not supported	address	2
Cortex-M0, Cortex-M3, Cortex-M4	4	aligned	consecutive	range	2, 4
Cortex-R5	8	aligned	consecutive	range	2
ColdFire 52xx, 522xx, 53xx	1	arbitrary	not supported	address	3
ColdFire 54xx, 54455	2	arbitrary	not supported	address	2
MIPS	2	aligned	arbitrary	address	
PPC 405	2	not supported	not supported	address	
PPC 440	2	not supported	arbitrary	address	5
PPC 55xx, 56xx	2	arbitrary	arbitrary	address	10
PPC 57xx	4	arbitrary	arbitrary	address	10
PPC 5200, 8247, 8248, 827x, 8280	2	not supported	not supported	address	6
PPC 5121, 83xx	2	arbitrary	not supported	address	6
PPC 7xx, 74xx, 86xx	1	not supported	not supported	address	6
PPC 603, 8240, 8241, 8245, 825x, 826x	0	n/a	n/a	n/a	
PPC BDM	1	aligned	not supported	address	8

Target	#	Ranges	Masks	Hit Detection	Notes
PPC e500, e500mc, e5500, e6500	2	arbitrary	arbitrary	address	10
x86	4	not supported	not supported	range	7, 9
XScale	1	aligned	arbitrary	range	

Notes

1. If you disable software breakpoints, 2 hardware breakpoints are available.
2. If more than one hardware breakpoint is set, the probe cannot always report which breakpoint is hit.
3. ColdFire 5206, 5272, 5249, 5282, 5235, 5271, 5275, 5251, and 5253 targets share a single register for data and execution breakpoints.
4. Hardware breakpoints are imprecise; the target stops several instructions after the access that hits the breakpoint.
5. Using a mask consumes both hardware data breakpoint registers.
6. Data hardware breakpoints have a minimum range of 8 bytes.
7. Data hardware breakpoints and execute hardware breakpoints share the same registers.
8. If you set a data hardware breakpoint with an address range, the target may not halt for accesses to the range, and may halt for accesses to addresses outside the range. For more information, see MPC860 chip errata CPU11.
9. Read-only data hardware breakpoints are not supported on this architecture.
10. Breakpoints with a range less than or equal to 4 are treated as if no range is specified. If you require breakpoints with this range, use a mask instead.
11. Cortex-A9 hardware breakpoint support varies by target; some do not support ranges or masks.

Execute Hardware Breakpoint Support

The following table describes execute hardware breakpoint support for a variety of targets:

Target	#	Ranges	Masks	Notes
ARM7, ARM9	1	not supported	arbitrary	1, 6
ARM11, Cortex-A8, Cortex-A9, Cortex-R4, PJ4	6	not supported	consecutive	
Cortex-A5	3	not supported	consecutive	
Cortex-A15	4	not supported	not supported	
Cortex-M0, Cortex-M3, Cortex-M4	10	not supported	consecutive	3
Cortex-R5	8	not supported	consecutive	
ColdFire 5206, 5272, 5249, 5282, 5235, 5271, 5275, 5251, 5253	1	aligned	arbitrary	1
ColdFire 5307	1	aligned	arbitrary	
ColdFire 5208, 5213, 52235, 52211, 52259, 52223, 52277, 5329, 5373, 5407, 5408, 5475, 5485, 54455	4	aligned	arbitrary	2
MIPS	4	aligned	arbitrary	
PPC 4xx	4	not supported	not supported	
PPC 55xx, 560x, 563x, and 566x	4	arbitrary	arbitrary	4, 5, 9
PPC 564x, 567x, 57xx	8	arbitrary	arbitrary	4, 5, 8
PPC 7xx, 74xx, 86xx	1	not supported	not supported	4, 7
PPC 603, 8240, 8245, 825x, 826x	1	not supported	not supported	4
PPC 5200, 8247, 8248, 827x, 828x	2	not supported	not supported	4

Target	#	Ranges	Masks	Notes
PPC 83xx	2	arbitrary	not supported	4
PPC BDM	3	not supported	not supported	
PPC e500, e500mc, e5500	2	arbitrary	arbitrary	4
PPC e6500	8	arbitrary	arbitrary	4
x86	4	not supported	not supported	1
XScale	2	not supported	not supported	

Notes

1. Data hardware breakpoints and execute hardware breakpoints share the same registers.
2. Only one execute hardware breakpoint supports ranges and masks.
3. Four of the execute hardware breakpoints support masks. These breakpoints share registers with data hardware breakpoints. The other six breakpoints do not support masks, and must be placed in flash.
4. These cores have an execute hardware breakpoint resolution that matches the instruction width (usually 4 bytes).
5. PPC 55xx cores that run VLE code have an execute hardware breakpoint resolution of 2 bytes.
6. If software breakpoints are disabled, 2 hardware breakpoints are available. On certain ARM9E cores, 2 hardware breakpoints are also available if **use_bpkt_inst** is set to `on`.
7. If software breakpoints are enabled, hardware breakpoints are not available.
8. Ranges are not supported on hardware execute breakpoints 5-8 of 564x and 567x targets.
9. If you set an execute hardware breakpoint while the target is running, the target may stop several instructions after the instruction that triggers the breakpoint.

Trace Usage Notes

ARM CoreSight Trace Support

The following terms and table describe ARM CoreSight trace component support:

- **Automatic** — Only GUI configuration is required.
- **Manual** — Register access provided, but your setup script must set them up.
- **Unsupported** — Contact Green Hills support for updated availability.

Component	Automatic	Manual	Unsupported
ATP Replicator	X		
CSTF (trace funnel)		X	
CTI (CTM and ECT)		X	
ETB (CoreSight)	X		
ETB (Trace Memory Controller)	X		
ETF			X
ETMv3	X		
ETR			X
HTM			X
ITM			X
MTB			X
PTM	X		
STM			X
TMC			X
TPIU	X		

Trace of CoreSight ETMv3 targets via the ETB may be incorrect due to an ambiguity between the CoreSight flush sequence and the ETMv3 branch encoding near a trace enabling event. As a workaround, collect full ETBs before disabling and re-enabling trace on these targets.

Allocation Errors While Tracing a Target

When tracing a target, if you receive an error similar to:

```
Unable to allocate resource
```

where *resource* is a type of trace resource, MULTI is indicating that it was unable to enable a trigger or filter that you have specified. For example, it may mean a trigger configuration is too complex for the target's available trigger programming model. The following list outlines architecture-specific reasons for this error:

ARM Cortex-M	DWT comparators are shared between trace and some hardware breakpoints. You can configure allocation with the **num_dwt** setting, or simplify your triggers.
ARM (except Cortex-M)	Trace comparators are used only by the trace system. Reduce the complexity of your triggers or filters.
ColdFire	The resources used for triggers are also used for hardware breakpoints. Use fewer hardware breakpoints or simplify your triggers.
PowerPC Nexus	The resources used for triggers are also used for hardware breakpoints. Use fewer hardware breakpoints or simplify your triggers or filters.

MULTI Usage Notes

MULTI Displays Incorrect Processor

The Green Hills Probe software provides **Project Wizard** entries for many development boards that correspond to processors supported by the probe. Though all of these processors are supported by the probe, some of them may not be explicitly supported by MULTI. If you select an entry that corresponds to one of these processors, MULTI sets your project's target to a compatible processor instead of the one you may expect from the name of the **Project Wizard** entry. This mapping is done so that you can debug new processor types with older versions of MULTI. It does not affect your probe's configuration, and should not cause problems during a debugging session. The mapping is more common with MULTI 4 than with MULTI 5, and explicit support for the processor may be added in a future MULTI release.

For example, if you create a project in MULTI 4 for a Freescale 8568E MDS, MULTI reports that your processor is a PowerPC 8548, because that is the closest compatible processor. You should still configure your probe using the appropriate target string (`ppc8568`).

Setting Flash Programmer Parameters

Flash programming parameters are stored in the **flash.cfg** file in the user configuration directory. These parameters, such as RAM location and size, may be valid for one target but not another. The flash programmer may load the saved parameters for a previous target instead of using the default parameters for the currently connected target. In this case, update the programming parameters in the flash dialog to match the current target. To return to the default settings for all targets, delete the **flash.cfg** file.

Trace is Not Supported With Multiple Debug Connections

Trace is not supported when multiple debug connections are made to a probe. To prevent multiple debug connections, use the **single_mpserv_only** configuration option (see “Disabling Multiple Binary Connections” on page 85).

Memory-Mapped Registers May Be Incorrectly Displayed

MULTI displays memory-mapped register information based on headers and documentation provided by silicon vendors. When probe software is released, there may only be preliminary headers and documentation, which may differ significantly from shipping hardware. For authoritative specifications, consult the documentation provided by the silicon vendor.

Multi-core Trace

Multi-core Trace Not Supported With Older Versions of MULTI

Tracing more than one core at the same time is supported in MULTI 6.1.4 or newer only.

Modifying Trace Options

When tracing more than one core simultaneously on MULTI 6.1.4, trace options should only be modified while trace is disabled.

Initialize Trace on All Cores Before Tracing

While tracing multiple cores on MULTI 6.1.4, initialize trace on all cores you want to trace before trace is enabled.

For each core you want to trace, select that core and type `trace` on the MULTI command line, or open a trace window to initialize the core for tracing.

If you want to initialize trace on other cores and trace is already enabled, first disable trace, then initialize the core by selecting the core and typing `trace` on the MULTI command line, or open a trace window and re-enable trace.

ARC Usage Notes

General ARC Usage Notes

Required Settings for ARCangel Evaluation Boards

For ARCangel evaluation boards, use the following DIP switch settings:

- 1 — `OFF`
- 2 — `OFF`
- 5 — `ON`

Also, the parallel port cable must be disconnected for FPGA blasting to work.

Error When Running Program After Blasting the Target

On some **.xbf** files, if you run a program immediately after blasting the target, an instruction error exception is taken right away. Usually, subsequent downloads work correctly.

Probe Cannot Reset ARC Hardware

The probe cannot reset ARC hardware, due to limitations in the ARC debug interface. When debugging ARC hardware, the **Reset** button in the MULTI Debugger appears dimmed.

The vb Command May Not Work

The **vb** test command may fail on ARC targets that do not implement the JTAG bypass register properly. Other debugging operations will usually work on these targets, so this failure can be ignored.

ARM Usage Notes

General ARM Usage Notes

Hardware Trace Limitations with the AM335x

Data address trace and data triggers of floating-point operations do not work on the AM335x due to hardware limitations.

Programming EE Flash on Fujitsu Cortex-R4 Chips

The internal flash of FCR4 series CPUs consists of two modules: TC flash and EE flash.EE flash is mirrored at two different addresses. `0xb0440000` is the ECC mirror and `0xb04c0000` is the non-ECC mirror. One of these addresses must be added to the bank list in the flash programmer to enable programming EE flash. If the EE flash is programmed in non-ECC mode, or if a programming error has occurred, reads from the ECC mirror will fail. Read failures will prevent the flash programmer from reprogramming the EE flash. The reported error message will either be the address of the read failure or a hardware exception. To resolve this condition, perform a blind erase of affected sectors by clearing both the **Program** and **Verify** options in the flash programmer. After the blind erase is done, re-enable Program and Verify to reprogram the flash module.

Programming the EE flash is not currently supported on the MP9EF126 and MB9DF126 ES1.

INTEGRITY ARMv7 Trace Support

When running INTEGRITY with an ARMv7 ASP (used by all Cortex-A9 BSPs), the trace decompressor can determine which AddressSpace was running at a given time. If you also need to know which Task was running, link the following function into the KernelSpace program:

```
#include <asp_export.h>
void BSP_ContextSwitchHook(void *oldtcb, void *newtcb, Value oldid, Value newid)
{
  // Write the Task control block address of the incoming task to CONTEXTIDR.PROCID
  MCR(CP15_CONTEXTIDR, MRC(CP15_CONTEXTIDR) & 0xff | (uint32_t)newtcb & ~0xff);
}
```

This hook adds about 100 clock cycles of overhead to each context switch.

Enabling Triggers on iMX6 Targets

The secondary cores on iMX6 targets are held in reset until they are released by the boot core. While they are held in reset, it is not possible to configure the Cross Trigger Interface (CTI) on those cores to enable triggers. While the Green Hills stand-alone setup script for the iMX6 releases the secondary cores from reset and configures the CTIs to enable triggers on all cores, startup code for multi-core aware applications, such as INTEGRITY, might require that the secondary cores are left in reset by the setup script. If this is the case, to enable triggers, you must configure the CTIs after your application has released the secondary cores from reset.

ARM1136 Cache Data is Incorrect or Not Viewable

The data cache on the ARM1136 is not viewable, but the cache tags are. The instruction cache is viewable, but due to a hardware error, the data shown for the odd-numbered words on each line may be incorrect.

Cannot Debug or Detect an ARMv7-A or ARMv7-R

If you cannot debug your ARMv7-A or ARMv7-R target, be sure you have correctly set the AP Index and Address of the core's registers. In most cases, the **detect** command can determine these values for you. If your target lacks a ROM table, or some parts of the ROM table are unreadable, you will have to set these values manually. For more information, see “Specifying CoreSight Targets” on page 78.

Fujitsu MB9 and Texas Instruments RM4x and TMS570 Flash Programming Configuration

When programming the internal flash modules of the Fujitsu MB9 or Texas Instruments RM4x processors, both the probe and the target must be configured for little-endian mode. When programming the Texas Instruments TMS570 series processors, both the probe and target must be configured for big-endian (BE32) mode.

Cannot Connect To An XScale Target While it is Running a Program

You cannot use the Green Hills Probe to connect to and debug an XScale target while it is running a program. Intel has published an application note related to this subject, but the Probe does not implement Intel's recommended solution. To gain control of an XScale target, reset it through the debug port using the Green Hills Probe.

Updating Exception Vectors on XScale Targets

In the XScale debug architecture, the debug solution (for example, your probe) must cache the processor's exception vectors, and the processor uses the cached vectors instead of the vectors in your target's memory. The probe guarantees its cached exception vectors are identical to those in memory after reset and each time the target is resumed or single-stepped. However, if your program updates exception vectors on-the-fly, the probe's cached vectors will not match the ones in memory unless you manually resume the target.

To work around this problem, use one of the following encodings of the `BKPT` instruction:

- ARM mode — `BKPT 0x1134` (opcode `0xe1211374`)
- Thumb mode — `BKPT 0x14` (opcode `0xbe14`)

These instructions tell the probe to stop the target, reload the vectors, and resume the target.

Alternatively, if your target does not change the exception vectors after the first exception, set the **auto_vector_reload** setting to `once`, and make sure that all vectors are set before the first exception of any kind is taken.

Cacheview Displays Incorrect Cache on XScale Manzano Cores

If you have an XScale target with a Manzano core (such as the PXA320 or IXP2350), you are using **Cacheview** to view a valid cache line in the L2 cache, and there is a valid L1-D line at the same address, **Cacheview** displays data from the L1 cache instead of the L2 cache. This is a hardware limitation.

Detect Sets Intel 80321 Targets to Intel 80219

The JTAG `IDCODE` registers on the Intel 80321 and 80219 processors are identical. As a result, the 80219 is detected as an 80321. To configure this target manually, use the following command:

```
set target i80219
```

Detect Sets Endianness Incorrectly on Some ARM Targets

detect may detect incorrect endianness on some ARM targets. Most ARM targets boot in little-endian mode, and can then be changed by boot code or a setup script to run in big-endian mode. However, the **detect** command often resets the target during the detection process, so the endianness will typically be detected as little-endian. To detect the proper endianness, run the **de** (detect endianness) command manually after the system has been configured to its final endianness, or set the endianness manually.

Detect Sets use_rtck Incorrectly on Some ARM Targets

detect may incorrectly set **use_rtck** to `normal` on some ARM targets, causing intermittent test, run control, and download failures. To fix this problem, either:

- Run the command `set use_rtck off`, or
- Load the probe configuration file for your board, located in ***install_dir*\target\*architecture***.

Trace Triggers on Single Addresses in Thumb Code Fail on iMX31 Targets

Due to a hardware limitation, trace triggers on single addresses in Thumb code do not work reliably on iMX31 targets. To work around this issue, use a ranged address trigger.

Data Value Read Triggers Unreliable on ARM11 Targets

Data value read triggers are unreliable on the ARM11. The target may indicate a trigger even when the data read is not equal to the comparison value.

How to Mass-Erase Flash on Kinetis Parts

Some Kinetis parts have a feature that locks flash and prevents debugger access until flash is erased. To mass-erase flash on a Kinetis part:

1. Make sure that the target type is set to `kinetis`.
2. Run the following commands:

```
> jp nsrst 0
> jr
> rr mdm-ap-status
> rw mdm-ap-control 1
> sleep 120
> rr mdm-ap-status
> tr
```

If the first command does not work, use `jp nreset 0` instead.

Cannot Trace More Than 128 Unique AddressSpaces On ARM Targets Without a ContextID Register

ARM processors that do not have a `ContextID` register limit the number of unique, traceable AddressSpaces to 128. Tracing a target on which the number of AddressSpaces exceeds 128 is not supported.

Base Address for Renesas RZ/A1 Flash Chip

Green Hills Probes support programming Spansion SPI flash memory chips from the Renesas RZ/A1. For example, the RTK772100FC evaluation board contains three SPI chips. Two are connected to channel 0, and one is connected to channel 1. These flash chips are identified in the flash programmer by the base address in the SPI multi I/O bus space. To program them, the SPI driver must be activated by adding a driver parameter to the base address in the flash dialog. For example, to program the first chip, set the base address to `0x18000000&driver=rzspi`.

The SPI flash controller, I/O pins, and clocks must be manually configured before programming. This should be done in a setup script run by the flash programmer.

The flash programmer will verify that the data was programmed correctly. After programming SPI flash, it will reset the target CPU, which will clear the SPI controller configuration. You will be able to read from flash in the debugger only after manually reconfiguring the SPI controller.

Setting the Trigger Position When Collecting Data from an ARM ETB

Trigger position works differently when collecting trace with an embedded trace buffer (ETB) than when collecting with a SuperTrace Probe. On an ETB, if the trigger occurs soon after trace collection is enabled, part of the ETB buffer may not be used because the ETB will stop collecting trace as soon as the trigger has reached the specified position in the buffer. For example, if trigger position is set to 50% and the trigger occurs after 5% of the buffer has filled:

- Collecting via STP, the buffer will fill completely, and the retrieved buffer will contain 5% data prior to the trigger and 95% data after the trigger.
- Collecting via ETB, the buffer will fill partially, and the retrieved buffer will be 45% unused (before the trigger), 5% data before the trigger, and 50% data after the trigger.

Note that this means using high trigger percentages with an ETB may result in very little data being collected.

If a trigger occurs within the first 1000 bytes or so of trace, the trigger packet might be before the first sync point. As a result, all of the trace data decompressed will be for code that executed after the trigger, and the trigger itself will not be in the trace list.

If you change the trigger position while tracing to the ARM ETB, the change will not take effect until you turn trace off and back on again.

Software Breakpoints in Memory Shared by Multiple Cores

Some SoCs, including the Freescale iMX6 and Renasas R-Car H2, prevent the probe from accessing the debug registers on the secondary CPUs until the boot core has done some initialization to power them up or bring them out of reset. On these cores, there is a brief period of time in between when the core begins executing code and when the probe can enable halt-mode debugging on the core. If a software breakpoint is hit during this time, the core will take an Undefined instruction exception rather than entering debug mode.

To mitigate this problem, early startup code may poll `DBGDSCR` until `DBGDSCR.HDBGen` is set. After `DBGDSCR.HDBGen` is set, software breakpoints will properly cause entry to debug mode.

CCN504 L3 Cache Home Node Mapping

For systems that include a CCN504 L3 Cache, there are effectively three orthogonal indices into the cache: `Set`, `Way`, and `Home Node`. To support this with the current `Set/Way` interface, the `Home Node` number is in bits `4-6` of the `Way`, while bits `0-3` contain the true `Way` number within the `Home Node`. For example, within a `Set`, the first `Home Node` will appear as `Ways 0-15`, the second `Home Node` as `Ways 16-31`, and so on.

Blackfin Usage Notes

General Blackfin Usage Notes

Power Detection on Targets with an ADI JTAG Port

Power detection is not supported for Blackfin targets using the ADI JTAG port, because it does not have a power pin. If you are not using the ADI target adapter, set the **power_detect** option to `off`. This option prevents the probe from trying to use a power pin with other adapters.

Hardware Breakpoint Support on bf533 and bf561 Cores

On bf533 and bf561 cores, hardware breakpoints are only supported in hardware revision 0.3 or greater.

Resetting bf561 Targets

On bf561 targets, the **tr d** command lets both cores run for a short while before putting them under debug control.

Single-stepping Over An Instruction May Return Incorrect Data

On Blackfin, if the **step_ints** option is set to `off` and you single-step over an instruction that directly reads or writes the memory-mapped `IMASK` register, a data read will return incorrect data, and a data write will not occur. To work around this problem, temporarily set **step_ints** to `on`, or run past the code instead.

User and Supervisor Stack Pointer Access with SP

When `SP` is accessed on Blackfin, the probe uses the user stack pointer or supervisor stack pointer depending on what mode the currently running program is in. `USP` always accesses the user stack pointer, and the virtual register `SSP` always accesses the supervisor stack pointer.

Debugging Programs Compiled with the ADI VDSP Compiler

If you are debugging a program compiled with the ADI VDSP compiler, pass the **-adi** option to **mpserv**.

Running a Mixture of GHS/VDSP-compiled Binaries Not Supported on Multi-Core Systems

Running a mixture of GHS/VDSP-compiled binaries on a multi-core system is not currently supported. If you need this functionality, contact Green Hills Software.

ColdFire Usage Notes

General ColdFire Usage Notes

ColdFire 5307 Data Value Triggers Unreliable

On ColdFire 5307, triggers that compare against data values do not work reliably. There are two failure modes:

- the trigger fires when the trigger condition has not occurred
- the trigger fails to fire when the trigger condition has occurred

This behavior is a hardware limitation.

Task-Aware Trace Not Available on ColdFire INTEGRITY

Due to hardware limitations, task-aware trace is not supported when tracing INTEGRITY.

MIPS Usage Notes

General MIPS Usage Notes

Detect Does Not Always Work For MIPS 74K Targets

In some cases, the detect command does not work on the MIPS 74K. This target cannot be halted unless the `DINT EJTAG` signal is pulled low. The probe does not drive `DINT` unless the target string is already set to `74K`. To work around this issue, set the target string manually or drive `DINT` low using the **jp** command.

Writing to Both the Even and Odd Ways of a TLB

To write to both ways of a TLB, use two successive **tlbw** commands. You do not need to specify the way; the probe determines this by looking at the page mask and virtual address. It then writes to the appropriate way and preserves the other one. For example:

```
tlbw 0 0x4110000 Pagemask=0x7fff 0x0 V,G,C=2,D        // This is an even entry
tlbw 0 0x4114000 Pagemask=0x7fff 0x14000 V,G,C=2,D    // This is an odd one
tlbr 0
entry 0, way 0: vaddr=0x04110000 vtag=0x0000:00006000 (ASID=0x0,PageMask=0x6000) ->
         paddr=0x00014000 ptag=0x17 (C=2,D,V,G)
entry 0, way 1: vaddr=0x04114000 vtag=0x0000:00006000 (ASID=0x0,PageMask=0x6000) ->
         paddr=0x00000000 ptag=0x01 (C=0,!D,!V,G)
```

TLB Initialization May Require Initializing All TLB Entries After Reset

TLB entries on MIPS 4K and variants include a hidden state bit which is cleared on reset and set when the entry is written. On reset, some entries may still be shown by the probe as valid when running the **tlbr** command. Because the hidden state bit is cleared, these entries will not be used by the processor for translation. This also affects the **vc** test. The **vc** test checks the TLB entries for a match against the starting address when it is outside the kseg regions. It may find a match for the starting address, but since the hidden state bit is cleared, the test will fail. This can be fixed by initializing all TLB entries after reset.

Target-Specific Permission Options for the ma Command

For MIPS32 and MIPS64 targets, the two target-specific permission options for **ma** (`0` and `1`) are used to indicate whether or not the probe should try to fill the TLB entry when a TLB miss occurs during an access to the specified range of memory.

- default (not set explicitly) — The probe reports the TLB miss, and the access fails.
- `0` — The probe fills TLB entries to handle misses when the processor is in kernel mode or supervisor mode.
- `1` — The probe fills TLB entries to handle misses when the processor is in user mode.

When the probe is instructed to fill the TLB entry, it calls the target code's TLB refill handler using a small agent (written into memory at the address specified by the **agent** option). In order for the probe to fill the TLB entry:

- you must have a working TLB refill handler

- you must not have breakpoints set in any code that might be called by the TLB refill handler

If either of these criteria is not met, your program may crash when the probe tries to fill the TLB entry.

For example, to tell the probe to fill TLB entries for misses that occur as a result of memory accesses between `0xc0000000` and `0xc01fffff` when the core is in kernel or supervisor mode, use the following command:

```
ma 0xc0000000 0xc01fffff rw0
```

To tell the probe to fill TLB entries for misses that occur as a result of memory accesses between `0x400000` and `0x4fffff` when the core is in user mode, use the following command:

```
ma 0x400000 0x4fffff rw1
```

PMC-Sierra Hardware Bug Workaround

The PMC-Sierra rm79xx and rm9xx families have a hardware bug that prevents debugging in user mode. Due to this bug:

- No software or hardware breakpoints are hit.
- The target cannot be halted while in user mode.

The Green Hills probe implements a workaround that simulates software breakpoints and allows you to single-step through code. This workaround does not simulate hardware breakpoints or allow you to halt the target. It involves inserting trap instructions where you want a software breakpoint and then inserting a real breakpoint in the exception vector to catch it while in kernel mode. The probe resumes when the exception vector is hit by a normal exception and halts as in a software breakpoint when the exception PC is one of those trap breakpoints.

Several probe options configure this workaround:

- **e9000_trap_brkpt** — dictates whether to use regular software breakpoints or the workaround.
- **e9000_trap_addr** — dictates the address where the probe sets the breakpoint in the exception vector.

- **e9000_trap_type** — dictates which type of breakpoint to use (hardware or software).
- **e9000_eret_addr** — indicates the location of any `eret` instruction in the code, enabling the probe to resume execution after hitting the trap breakpoint.

Set these options before debugging and do not change them. To have the most visibility, set **e9000_trap_addr** to the entry point of one of the following exception vectors:

- `0x80000180` — RAM BEV of Status `reg == 0`
- `0xbfc00380` — ROM BEV of Status `reg == 1`

This slows the program execution every time the system hits one of these exceptions; the probe checks to see if it is a trap breakpoint, and resumes if it is not. If you set the breakpoint further in the exception vector, there would be less slowdown, but registers `k0` and `k1` could be changed by the OS. A hardware breakpoint would have to be used when debugging the target in ROM.

Enforcing the Primary Core's ROCC Bit on Cavium Octeon cnMIPS64 Boards

Cavium Octeon chips using the cnMIPS64 core hold all but the primary core in reset until software releases them. Therefore, the probe firmware does not wait for the `ROCC` bit to go low on these cores. If you want the probe to enforce this bit for a primary core, set the **check_rocc_at_reset** option to `on`. During reset, if you see a `ROCC bit never went low` error, this option might be set to `on` for one of the cores that is being held in reset. To fix this error, set the option to `off`.

Configuring the Probe for Nuova Cores

When specifying a Nuova core with a 5-bit IR, use the target string `nuova5`. For a 6-bit IR, use `nuova6`.

Nuova Cores Do Not Support vb or detect

Nuova cores do not handle the JTAG bypass instruction properly. In a single-core setup, the probe does not normally scan a bypass instruction, however, you must not use the **vb** or **detect** commands, because they do use this instruction.

Data Hardware Breakpoints with Data Value Compares May Cause an Imprecise Exception

Data hardware breakpoints with data value compares may cause an imprecise exception. When this happens, the probe reports the correct hardware breakpoint, but at a later PC. See the MIPS EJTAG specification for more information.

PowerPC Usage Notes

General PowerPC Usage Notes

Triggering Data Hardware Breakpoints on Certain PowerPC Cores

In order for a data hardware breakpoint to trigger on PowerPC 7xx, 7400, 7410, 744x, or 86xx cores:

- The access address must match the breakpoint address
- `DABR[BT]` must match `MSR[DR]`

If your target is a 7xx, 7400, 7410, 744x or 745x, the probe attempts to compensate for this by setting the BT bit to match the current value of `MSR[DR]` when the breakpoint is first set or re-enabled. Even so, some data breakpoints might not be hit if the application code changes the value of `MSR[DR]`.

If your target is an 86xx, set the **databp_translate** option to configure this behavior.

Triggering Execute Hardware Breakpoints on Certain PowerPC Cores

In order for an execute hardware breakpoint to trigger on PowerPC 7xx, 7400, 7410, 744x, or 86xx cores:

- The current PC must match the upper 30 bits of `IABR`
- `MSR[IR]` must match `IABR[TE]`

If your target is a 744x or 745x, the probe tries to compensate for this by setting `TE` to match the current value of `MSR[IR]` when the breakpoint is first set or re-enabled. The probe does not set `TE` for hardware execute breakpoints with addresses less than `0x2000`, because this is the default exception vector space, and the MMU is disabled when processing exceptions.

If your target is a 7xx, 7400, or 7410, the probe does not set the `TE` bit when installing a breakpoint, so by default, execute hardware breakpoints do not trigger in cases where `MSR[IR]` is enabled. To work around this issue manually, set the least significant bit of the breakpoint address. For example, to set a hardware breakpoint to trigger when `MSR[IR]` is enabled and the instruction at `0x8000` is executed, use the following command:

```
bs x 0x8001
```

If your target is an 86xx, set the **execbp_translate** option to configure this behavior.

Tracing Nexus Targets With `Trace On Function and Callees Not Executing'

When tracing a Power Architecture Nexus target using the **Trace on: Function and Callees Not Executing** feature, no trace data is collected until the specified function exits for the first time.

Limitations Debugging Exception Handlers on Certain PowerPC Cores

On PowerPC 7xx, 74xx, 86xx, and BDM cores, do not set a software breakpoint in the following places:

- Inside an interrupt handler, before the context registers are backed up
- Inside an interrupt handler, after the context registers are restored, but before `rfi`

If you do, the `srr0` and `srr1` registers may get corrupted, putting the target in an unrecoverable state.

Long Pause Before Downloading Program from MULTI

If there seems to be a long pause before a download begins in MULTI, it is most likely due to the probe flushing the L1 data cache. If your application enables the data cache anyway, disable the data cache in the setup script (using an appropriate register write to `hid0`) to remove this pause.

Problems with Single-Stepping

The **step** configuration setting determines the method the probe uses to single-step through code. If you are having a problem with single-stepping, a different setting may single-step properly. Some PowerPC targets may have problems when using trace single-step with the I-Cache enabled.

No Support for Debugging eTPU on PowerPC 55xx Processors

MULTI does not support debugging the eTPU on PowerPC 55xx processors. When connecting with MULTI, only the e200 cores are shown. You can still use the probe console for debugging operations on the eTPU, such as viewing registers and memory, setting breakpoints, running, halting, and stepping.

Reading or Writing Memory Cleans Data Cache Lines on Certain PowerPC Cores

The probe issues a snoop request for most memory accesses on PowerPC e300, e500, e5500, and e6500 cores. This allows the probe to maintain cache coherency with the target, but also has the effect of flushing any D-Cache or L2 cache lines associated with the access address. This may have a small impact on target performance upon resume, as the core may need to refill affected cache lines.

Memory accesses to addresses marked instruction-only (`i` suffix) or raw (`r` suffix) do not cause snoop requests. For more information, see “Address Suffixes” on page 190.

PowerPC 4xx Usage Notes

Freeze-Mode Debugging with INTEGRITY

If you are using freeze-mode debugging to debug INTEGRITY on a PowerPC 4xx, set the **inval_entire_icache** option to `on`.

If you are using freeze-mode debugging in MULTI 4 to debug INTEGRITY virtual tasks on a PowerPC 4xx, add the following command to your setup script:

```
target allphysical on
```

Do not add this command if you are using MULTI 5.

No Valid Cache Lines May Exist In Cache Inhibited Memory Regions

On PowerPC 440 and 460 targets, if you use the **tlbw** command to set the cache inhibit bit in a TLB entry, you must also ensure that no valid cache lines exist for the memory region that is now cache inhibited. Use the following command sequence to invalidate the data cache:

```
clop invalidate i * *
clop invalidate d * *
```

310-PAC2M-02 Pin Adapter Hardware Flaw

Pin adapter 310-PAC2M-02 is a shrouded COP 16-pin male header placed opposite a 38-pin AMP MICTOR plug on a small PCB labeled PPC and 2A. This adapter has a minor error that affects PowerPC 4xx targets. Pin 7 of the MICTOR is connected to the active-low `HALT` signal on the target processor. The Green Hills Probe cannot assert this pin when you are using this adapter, because the pin is not wired correctly on the adapter. The `HALT` pin is only required to perform a precise system-type reset. It can also be asserted manually through the probe's **jp** pin control command (see “JTAG and SWD Commands” on page 183). No fix to this adapter is planned, although the adapter may be replaced in the future with a new model that does not have this limitation. Contact Green Hills support if you have further questions about this issue.

Cannot Trace More Than 255 Unique AddressSpaces On PowerPC 440 or 460

PowerPC 440 and 460 processors limit the number of unique, traceable AddressSpaces to 255. Tracing a target on which the number of AddressSpaces exceeds 255 is not supported.

PowerPC e200 Usage Notes

Reading and Writing While Core Is Running Bypasses the Cache

It is possible to read and write target memory while the processor is running. However, these memory accesses always bypass the cache and access memory directly. Only use this feature when the cache is disabled, or in memory corresponding to cache-inhibited TLB entries, or the results may be unpredictable or incorrect.

SPC56APxx Does Not Support Low-Power Mode Monitoring or Live Memory Access

Due to its design, the SPC56AP60 and SPC56AP54 do not support live memory access or monitoring of low-power mode.

Programming the UTest Block on MPC57xx and SPC57x

MPC57xx and SPC57x series CPUs contain internal flash modules which can be programmed from the Debugger. One of the internal flash blocks, UTest, requires special care when programming. The UTest block contains test, configuration, and security information. Changing the settings in this block may permanently modify CPU operation or lock out debugging access. The Debugger's flash programmer can program but not erase this block.

To program the UTest block:

1. Change the base address of internal flash from `0x400000` to `0x400000&modify_utest=1`. Some CPUs contain internal flash that starts at address `0`, in which case you must enter `0&modify_utest=1`.
2. Provide either a raw binary image of the modified `UTest` block or an ELF file that contains UTest data.
3. Disable the erase option at the top of the flash programmer dialog.

Because the flash programmer cannot erase the UTest block, only flash lines which have not yet been programmed can be modified. These flash lines each contain eight bytes. If a line is programmed twice with different values, the checksum for that line will be incorrect and this will cause permanent read failures. Please see the reference manual for your CPU for the UTest block map and DCF record format.

PowerPC 744x and 745x Usage Notes

Detect Sets PowerPC 7445 or 7447 Incorrectly

When using the **detect** command, the probe cannot determine if the target has an L3 cache present, so it may detect a 7455 when the actual target is a 7445, and similarly detect a 7457 when the target is a 7447.

Writes to the msscr0 Register Are Not Supported

The probe does not support writing to the `msscr0` register. Writes to this register should be accomplished through application code. Additionally, when reading this register via the probe, some bits may always appear as `0` on 7441, 7445, 7450, 7451, and 7455 targets.

Freeze-Mode Debugging Unreliable When L3 Cache Is Enabled

With the L3 cache enabled on 745x targets, freeze-mode debugging may be unreliable in certain configurations. It is recommended if possible to use run-mode debugging with **soft_stop** set to `off` in these situations.

Software Breakpoints Require a Valid Opcode at the Program Exception Vector

If you are using software breakpoints when debugging your target, there must be a valid opcode in memory at the program exception vector address (`0xfff00700` or `0x700`, depending on the value of `MSR[IP]`). Otherwise, the CPU will take a double exception when the software breakpoint is hit. For more information, see Errata 32 in the 7455 Chip Errata list.

744x or 745x Halts Unexpectedly

If a 744x or 745x target halts unexpectedly at the program interrupt vector (`+0x700`) after a run or step command, you may try setting the **icache_step** option to `off`. This is known to correct the behavior in certain situations.

PowerPC 750 Usage Notes

clst and clsa Modify the L2 Cache in Two Ways on 750GX Cores

Using **clst** or **clsa** to modify the L2 cache on a 750GX will modify two ways instead of just one. This is due to a chip bug.

L2 Cache Coherency Required on 750L Targets

The contents of the L2 cache cannot be read on the IBM PowerPC 750L using the **clr** command. The status and tag bits are still displayed. The agent must be enabled to maintain L2 cache coherency while the probe debugs this target. For more information about enabling the agent, see the **agent** option in "PowerPC" on page 124.

The D-Cache Tag Cannot Be Written On Some PowerPCTargets

The D-Cache tag cannot be written on the 750L or 750CX. `clsa d` does not work on these targets.

The Probe Cannot Access THRM4 on the 750GX

The probe cannot access the `THRM4` register when debugging a PowerPC 750GX.

MULTI Does Not Distinguish Among PowerPC 750 Variants

When connecting in MULTI to any PowerPC 750, MULTI lists the connected target as a PowerPC 750, even if it is a 750L, 750X, etc.

PowerPC 83xx Usage Notes

Core Cannot Be Halted or Debugged in Low-Power Mode

PowerPC 83xx cores cannot be halted or debugged if they enter low-power mode, which is indicated by the POW bit in the MSR. If you want to debug an RTOS or other application that enables low-power mode, you may need to disable this mode for debugging operations to work correctly. For example, if debugging Linux, you would comment out the line:

```
oris r7,r7,MSR_POW@h
```

in the file **idle_6xx.S**, because it enables low-power modes. If you enter debug mode and all of the SPRs have the same value, that usually indicates that the processor was in low-power mode when it halted.

PowerPC 85xx and QorIQ e500v2 Usage Notes

Valid Opcode Required at IVOR15

Some (typically older) processors have a requirement that for software and hardware breakpoints to work, `IVOR15` must point to an address that has a valid TLB mapping and contains a valid opcode. By default, this is not true at reset, so, for example, the processor may fail to stop at hardware breakpoints set in the boot page of memory if `IVOR15` (and `IVPR`) are not configured first. The failure mode will be that the processor will spin, and when halted, the processor will be at zero, and all registers will be zero.

Machine Check Exceptions When no_boot_rom is Enabled

When the **no_boot_rom** option is enabled on an 85xx, P1xxx, or P2xxx, and the target is reset, a machine check exception will occur if ever `HID0[EMCP]` is set. This is due to a debug interface limitation. To avoid this, only use the **no_boot_rom** option for restoring code at the reset vector and other board bring-up operations, and then disable it for normal debugging purposes. The **reset_fixup** option does not not result in a machine check, but it should only be used to perform the TLB or single-step workaround — it should not be depended upon to provide the same reset recovery behavior as **no_boot_rom**.

PowerPC QorIQ e500mc, e5500, and e6500 Usage Notes

INTEGRITY Trace Support

When running INTEGRITY with an e500mc, e5500, or e6500 processor, the trace decompressor needs additional Task information when the kernel switches to a virtual AddressSpace. Link the following function into the KernelSpace program so that the decompressor will have the additional task information and be able to trace virtual AddressSpaces.

```
/*
// Write the new TaskContext address to the NPIDR register so that trace can
// figure out what task we're switching to.
// NOTE: Trace will be confused if this task no longer exists by the time
// the trace data is processed.
*/
void BSP_ContextSwitchHook(void *oldtc, void *newtc, Value oldid, Value newid)
{
#if __ADDRESS_BITS == 32
   __MTSPR(SPR_NPIDR, (Address)newtc);
#else
 // The NPIDR register is still only 32 bits regardless of address size.
 // We know newtc will always be page aligned and we take advantage of
 // that by shifting it right by 11 and then setting the LSB (to indicate
 // to the trace decoder that we shifted it).
 // This way we'll be fine unless a TaskContext address is larger than 2^43.
   __MTSPR(SPR_NPIDR, (((Address)newtc) >> 11) | 1);
#endif
}
```

Trace Requirements for QorIQ over HSST

To trace QorIQ targets over HSST, the following requirements must be met:

- You must use a serial trace-equipped SuperTrace Probe.
- You must connect to your board using a HS70 or HS22 cable.
- Your board must enable one or more Aurora lanes. For an example, see the **.mbs** script provided by the New Project Wizard. This script includes support for temporarily overriding the target's reset configuration word (RCW). You may need to customize the RCW values in the script depending on your `SerDes` allocation requirements; see your device's reference manual for details on RCW bit descriptions.

- You must set the probe's **hsst_rx_lanes** option to the number of lanes that are enabled. You must do this before establishing a debug connection between your host and the probe (such as with **mpserv**).
- In MULTI, you must set the lane line rate in Mbps with `target tracereg aurora_linerate=`*`line_rate`*. The **.mbs** script provided by the New Project Wizard includes a utility function to dynamically detect the line rate based on RCW values. Supported values are `2500`, `3125`, and `5000`.

For more details, see ***comp_install*/ghprobe/hsst_debug.rc**.

Cannot Set Trace Triggers or Filters on a Running QorIQ Target

On QorIQ targets, triggers and trace filters cannot be configured while the target is running. The target must be halted to set a trigger or trace filter (or you can set it before downloading and running the program). If you attempt to modify an existing trigger while the target is running, the existing trigger remains set on the target, but the **Trace Triggers** dialog box will not indicate that the trigger is set.

Gap Between Trace Start Event and Actual Trace Data when using Filters on QorIQ Targets

When using trace filters with QorIQ targets, there is a small gap between when the start trace event occurs and when the trace actually begins.

Raw Memory Writes to SDRAM Locks Debug Interface on Some QorIQ Targets

When debugging some QorIQ targets, performing raw memory writes to SDRAM with Core Platform Caches enabled may cause the debug interface to become unresponsive.

36-Bit Addressing

When using addresses larger than 32 bits, you must set the `EN_MAS_7_UPDATE bit of HID0`, or the probe may not be able to translate addresses correctly.

PowerPC 86xx Usage Notes

Writes to the msscr0 Register Are Not Supported

The probe does not support writing to the `msscr0` register. Writes to this register should be accomplished through application code. Additionally, when reading this register via the probe, some bits may always appear as `0` on 86xx targets.

Target Halts Unexpectedly at the Program Interrupt Vector

If your target halts unexpectedly at the program interrupt vector (`+0x700`) after a run or step command, you may try setting the **icache_step** option to `off`. This is known to correct the behavior in certain situations.

PowerPC BDM Usage Notes

PowerPC BDM Targets Must Be Reset by the Probe Before Debugging

PowerPC BDM targets cannot be controlled from the probe unless the most recent target reset was performed by the probe. If an external reset causes the target to reset, the probe will not be able to halt the target until a subsequent reset command is sent by the probe.

Probe Does Not Provide Access to the DPDR Register on PowerPC BDM Targets

The probe cannot give the user access to the `DPDR` register because it must use this register for all debugging operations. The **rr** command cannot read this register.

SRR0 Register Always Refers to the Current PC on PowerPC BDM Targets

Because of the exception debug model these processors use, the value of the `SRR0` register, as viewed using the Green Hills Probe, will always refer to the current `PC`.

Thus the values the probe displays for the `PC`, `IAR`, and `SRR0` will always be the same.

Usage Notes for Other PowerPC Cores

Using Triggers with PowerPC 405, 440, and 460

PowerPC 405, 440, and 460 processors use breakpoint hardware to implement triggers. As a result, all hardware and software breakpoints must be disabled to enable the use of triggers (you can use either breakpoints or trace triggers). We recommend that you do not switch between using breakpoints and triggers during a debugging session. If you do, a breakpoint may appear in the trace list as a trigger, and a trigger will not appear.

To enable support for triggers on PowerPC 405, 440, and 460:

1. Disable all hardware and software breakpoints.
2. Enable support for triggers by passing the **-trace_triggers** option to **mpserv** or by issuing the command **target trace_triggers on**. Any breakpoints (such as those internally set for system calls) that are still enabled when you perform this step will produce a warning and then be automatically disabled. For more information, see “Options for Custom Connection Methods” on page 54 and “Other Commands” on page 209.

The deactivation of breakpoints to allow for trigger support means that any attempt to set new breakpoints will fail. Additionally, host-based system calls, which depend on breakpoints and which affect the functioning of host file I/O and the MULTI **I/O** pane, will not work. Consequently, when breakpoints are disabled, you should issue the command **target syscalls off** before loading a program. For more information, see “Other Commands” on page 209.

Note

When you halt your system with trigger support enabled and breakpoints disabled, you may get error messages about your program being stopped at unknown breakpoints. Please ignore the following error messages in this situation:

```
mpserv: error getting hardware breakpoint status.
```

```
        Stopped by unknown hardware break (tag=-1, cause=0)
```

To disable support for triggers and restore the use of breakpoints:

1. Issue the command **target trace_triggers off**.
2. Re-enable all hardware and software breakpoints.

Note

The Agilent E5904B trace probe for PowerPC 405 and 440 does not support tracing with breakpoints enabled.

603ev and 82xx Breakpoint and stw Instruction Interaction

On these targets, if the instruction cache is enabled and a software breakpoint is set on the instruction following an **stw** instruction, it is possible that the store will not have taken place yet. The write occurs after, if a single-step is performed. Also, if an **isync** instruction is inserted before the software breakpoint, the store takes place before the breakpoint occurs.

The clst Command Cannot Clear D-Cache Status Bits

The **clst** command cannot be used to clear status bits on the D-Cache for the PowerPC 5200, 8247, 8248, 8270, 8271, 8272, 8275, or 8280.

Data Corruption on a 7400 or 7410 with L2 Cache Enabled

Data corruption may occur when debugging a PowerPC 7400 or 7410 with the L2 cache enabled. This documented as Freescale's MPC7410 Chip Errata 20. We recommend either disabling the L2 cache, setting it to instruction mode only, or setting it to write through mode. Any of these should prevent the data corruption.

SH Usage Notes

General SH Usage Notes

Reset Retry Limit Error When Resetting Certain SH Targets

Some SH targets require a specific reset sequence before they respond to debugging. Because the H-UDI specification does not allow the probe to control the reset line, the probe is unable to perform this sequence itself, and if the probe is connected when the board is powered on, it may interfere with this process. When this problem occurs, resetting the board from the probe results in a `Reached reset retry limit` error. To solve this problem:

1. tristate the probe's outputs by pressing the front panel button or using the **jp off** command
2. cycle the power on the board.

Debugging should then work normally.

Required Setting for SH-2A Targets

To properly halt an SH-2A target at reset, set the **target_reset_pin** option to `freezes_tap`.

Cannot Detect Certain SH Targets

The **detect** command does not work on SH7261 and SH7780 cores that lack proper JTAG ID codes. Set these targets manually with the **set target** command.

Configuring Your Probe for SH7780 Targets

If your SH7780 target boots up with an 8-bit IR, your target string should be set to `sh7780_8`. Otherwise, use `sh7780`. Use the **detect** command to find the IR length for your target.

Collecting Trace Data from SH7206 Targets Requires Correct Setting for Pin Set

You cannot collect trace data from an SH7206 target until you set the **Pin Set** option in the **Target Specific Options** trace dialog from **Undecided** to the configuration of the target board. If you are using an M3A-HS60 evaluation board, the correct setting is `PA16, PE0, PE3, PE4, PE5, PE6`.

Special Considerations when Caches are Enabled on SH7263 Targets

When using an SH7263 processor with a trace probe, in some cases, real-time trace must be enabled if caches are also enabled (due to a hardware bug). If real-time trace is off, the hardware stalls when its trace output FIFO fills up, which may cause cache corruption.

Setting the Trace Clock Multiplier

In the SH trace **Target Specific Options** dialog, you can set the trace clock multiplier to `1/1`, `1/2`, `1/4`, or `1/8`. Some boards may output incorrect trace data if the trace clock frequency is too high. Unless you are having problems with DDR trace clocking on an SH7261 target, you should use DDR.

SH Trace Capacity

This note provides a rough estimate of SH trace capacity by examining how much execution can fit in the full 1 GB trace buffer running a simple test program. The test program consists of:

- 10% branches (75% taken)
- 21% loads
- 14% stores

All tests were run with processor stalling enabled, which is why the times are the same. The program counter was collected on each test:

Data Read	Data Write	Timestamps	Number of Instructions	Time
off	off	off	2790 million	n/a
off	on	off	689 million	n/a
on	off	off	466 million	n/a
on	on	off	305 million	n/a
off	off	on	1549 million	8.9 s
off	on	on	384 million	8.9 s
on	off	on	256 million	8.9 s
on	on	on	166 million	8.9 s

x86 Usage Notes

Cannot Single Step from a hlt Opcode on x86

The standard mechanism that the probe uses to single step the processor does not work if the processor is stopped at a `hlt` opcode. If the next opcode is `leave` or `nop`, the probe works around this issue by setting a hardware breakpoint at the address following the `leave` or `nop` opcode. Note that `BSP_Doze()` in the INTEGRITY BSP includes a `leave` opcode after `hlt` if you compile with **-G**.

When the probe cannot work around the issue, or no interrupt comes in, the probe is unable to resume the target. In this case, comment out the `hlt` opcodes in your code and rebuild.

Setting Logic High Level Is Unnecessary on x86 Targets

The TraceEverywhere cables for Intel x86 targets use GTL logic that automatically uses the proper logic level for JTAG signals. When a GTL connector is attached to the probe, the `logic_high` configuration option is not available, and the **dlh** command does not perform any logic detection.

Appendix E

Supported Devices and Adapter Types

Contents

Supported Devices

The following sections lists supported values for *device_name* for Green Hills Probe and SuperTrace Probe connections (see “Specifying Your Target” on page 76). The lists in the table may not be comprehensive. For a complete list of supported device names, use the following command:

```
set target ?
```

If your target is not listed, contact your Green Hills Software Sales Representative.

The columns on the right-hand side of each table indicate the following:

- **GHP** — Run control supported on the Green Hills Probe V3.
- **STPv1** — Run control supported on the SuperTrace Probe V1.
- **STPv3** — Run control supported on the SuperTrace Probe V3.
- **Trace** — External trace data collection is supported on SuperTrace probes; for targets with an ARM ETB or ARM CoreSight ETB, collection from ETB is supported with the STPv3 and GHPv3.

Supported ARM Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
ARM7	`arm7di`	X	X		X
	`arm7tdmi`	X	X	X	X
	`arm7tdmi-s`	X	X	X	X
ARM9	`arm920`	X	X	X	X
	`arm922`	X	X	X	X
	`arm926`	X	X	X	X
	`arm940`	X	X	X	X
	`arm946`	X	X	X	X
	`arm966`	X	X	X	X
	`arm968`	X	X	X	X
	`arm968srd`	X	X	X	X

Family	Device Name	GHP	STPv1	STPv3	Trace
ARM11	`arm1136`	X	X	X	X
	`arm1156t2fs`	X	X	X	X
	`arm1156t2s`	X	X	X	X
	`arm1176jzfs`	X	X	X	X
	`arm1176jzs`	X	X	X	X
	`arm11mp`	X		X	
	`omap2430`	X	X	X	X
	`pj4_v6`	X	X	X	X
Cortex	`a5` *	X		X	
	`a8` *	X	X	X	X
	`a9` *	X		X	X
	`a15` *	X		X	X
	`am335x`	X		X	
	`am389x`	X		X	
	`axx5500**`	X		X	X
	`fcr4`	X		X	
	`kinetis`	X	X	X	X
	`m0` *	X		X	
	`m3` *	X	X	X	X
	`m4` *	X	X	X	X
	`m4f` *	X	X	X	X
	`omap5`	X		X	X
	`pj4_v7`	X		X	X
	`r4` *	X	X	X	X
	`r4f` *	X	X	X	X
	`r5` *	X		X	
	`r5f` *	X		X	
	`vybrid`	X		X	
ICEPick	`icepick(`*`id`*`:`*`name`*`)`	X	X	X	

Family	Device Name	GHP	STPv1	STPv3	Trace
XScale	`i80200`	X	X		
	`i80219`	X	X		
	`i80321`	X	X		
	`i80331`	X	X		
	`ixp2350`	X	X		
	`ixp2400`	X	X		
	`ixp2800`	X	X		
	`ixp425`	X	X		
	`pxa210`	X	X		
	`pxa250`	X	X		
	`pxa255`	X	X		
	`pxa270`	X	X		
	`pxa320`	X	X		
	`xscale`	X	X		

* Indicates a target that must specified with `csdap()`, see “Specifying CoreSight Targets” on page 78 for more details.

** Different AXX5500 family members have different numbers of cores. Specify the number of cores as `axx5500.`*`n`*, where *`n`* is the number of cores. For example, for an AXM5512, which has 12 cores, use the target type `axx5500.12`. If not specified, the number of cores defaults to 16.

Supported ARC Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
ARC	`ARC600`	X			
	`ARC700`	X			
	`ARCtangent-A4`	X			
	`ARCtangent-A5`	X			

Supported Blackfin Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
Blackfin	`bf533`	X			
	`bf561a`	X			
	`bf561b`	X			

Note

The device names `bf561a` and `bf561b` are always used together, and only for the Blackfin 561 multi-core system. The correct target setting for this system is:

```
set target bf561a bf561b
```

Supported ColdFire Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
ColdFire	`cf5206e`	X	X	X	X
	`cf5208`	X	X	X	X
	`cf5213`	X	X	X	X
	`cf52211`	X	X	X	X
	`cf52223`	X	X	X	X
	`cf52235`	X	X	X	X
	`cf52259`	X	X	X	X
	`cf52277`	X	X	X	X
	`cf5235`	X	X	X	X
	`cf5249`	X	X	X	X
	`cf5251`	X	X	X	X
	`cf5253`	X	X	X	X
	`cf5271`	X	X	X	X
	`cf5272`	X	X	X	X
	`cf5275`	X	X	X	X
	`cf5282`	X	X	X	X
	`cf5301x`	X		X	X
	`cf5307`	X	X	X	X
	`cf5329`	X	X	X	X
	`cf5373`	X	X	X	X
	`cf5407`	X	X	X	X
	`cf54418`	X		X	X
	`cf54455`	X	X	X	X
	`cf5475`	X	X	X	X
	`cf5485`	X	X	X	X
	`cfv4e`	X	X	X	X

Supported Lexra Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
Lexra	`lx4189`	X			

Supported MIPS/EJTAG Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
MIPS32 MIPS64	`mips32_4kc`	X			
	`mips32_4kec`	X			
	`mips32_4kem`	X			
	`mips32_4kep`	X			
	`mips32_4km`	X			
	`mips32_4kp`	X			
	`mips32_4ksc`	X			
	`mips32_14kc`	X			
	`mips32_14kf`	X			
	`mips32_74kc`	X			
	`mips32_74kf`	X			
	`mips32_1004kc`	X			
	`mips32_1004kf`	X			
	`mips64_5kc`	X			
	`mips64_5kf`	X			
	`mips64_20kc`	X			
	`mips32_m4k`	X			
Broadcom	`bcm1250`	X			
	`bcm3345`	X			
	`bcm6352`	X			
	`bcm7115`	X			
	`bcm7320`	X			

Family	Device Name	GHP	STPv1	STPv3	Trace
IDT323xx	`idt32334`	X			
	`idt32355`	X			
	`idt32364`	X			
Intrinsity	`FastMATH`	X			
	`FastMIPS`	X			
Renesas	`vr4131`	X			
	`vr5500`	X			
Toshiba	`tx4937`	X	X		X
	`tx4938`	X	X		X
	`tx4955`	X	X		X

Supported PowerPC Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
4xx*	`apm82181`	X		X	X
	`ppc405`	X	X	X	X
	`ppc405ex`	X	X	X	
	`ppc405exr`	X	X	X	
	`ppc405ez`	X	X	X	
	`ppc405gp`	X	X	X	X
	`ppc405gpr`	X	X	X	X
	`ppc440ep`	X	X	X	X
	`ppc440epx`	X		X	X
	`ppc440gp`	X	X	X	X
	`ppc440gr`	X		X	X
	`ppc440grx`	X		X	X
	`ppc440gx`	X	X	X	X
	`ppc440spe`	X		X	X
	`ppc440x5`	X	X	X	X
	`ppc460ex`	X		X	X
	`ppc460gt`	X		X	X
	`ppc460gtx`	X		X	X
	`ppc460sx`	X		X	X
7xx	`ppc740`	X			
	`ppc745`	X			
	`ppc750`**	X			
	`ppc755`	X			

Family	Device Name	GHP	STPv1	STPv3	Trace
74xx	`ppc7400`	X			
	`ppc7410`	X			
	`ppc7441`	X			
	`ppc7445`	X			
	`ppc7447`	X			
	`ppc7447A`	X			
	`ppc7448`	X			
	`ppc7450`	X			
	`ppc7451`	X			
	`ppc7455`	X			
	`ppc7457`	X			

Family	Device Name	GHP	STPv1	STPv3	Trace
BDM	`ppc5xx`	X			
	`ppc533`	X			
	`ppc534`	X			
	`ppc535`	X			
	`ppc536`	X			
	`ppc555`	X			
	`ppc556`	X			
	`ppc560`	X			
	`ppc561`	X			
	`ppc562`	X			
	`ppc563`	X			
	`ppc564`	X			
	`ppc565`	X			
	`ppc566`	X			
	`ppc8xx`	X			
	`ppc821`	X			
	`ppc823`	X			
	`ppc823e`	X			
	`ppc850`	X			
	`ppc852t`	X			
	`ppc855`	X			
	`ppc855t`	X			
	`ppc857t`	X			
	`ppc857dsl`	X			
	`ppc859t`	X			
	`ppc859dsl`	X			

Family	Device Name	GHP	STPv1	STPv3	Trace
BDM	`ppc860`	X			
	`ppc860p`	X			
	`ppc860dp`	X			
	`ppc862`	X			
	`ppc862p`	X			
	`ppc862t`	X			
	`ppc866p`	X			
	`ppc866t`	X			
	`ppc870`	X			
	`ppc875`	X			
	`ppc880`	X			
	`ppc885`	X			
G2	`mgt5200`	X			
	`ppc603`***	X			
	`ppc8240`	X			
	`ppc8245`	X			
	`ppc8247`	X			
	`ppc8248`	X			
	`ppc8250`	X			
	`ppc8255`	X			
	`ppc8260`	X			
	`ppc8264`	X			
	`ppc8265`	X			
	`ppc8266`	X			
	`ppc8270`	X			
	`ppc8271`	X			
	`ppc8272`	X			
	`ppc8275`	X			
	`ppc8280`	X			

Family	Device Name	GHP	STPv1	STPv3	Trace
e200	`ppc5516`	X		X	X
	`ppc5517`	X		X	X
	`ppc5533`	X	X	X	X
	`ppc5534`	X	X	X	X
	`ppc5553`	X	X	X	X
	`ppc5554`	X	X	X	X
	`ppc5561`	X	X	X	X
	`ppc5565`	X	X	X	X
	`ppc5566`	X	X	X	X
	`ppc5567`	X	X	X	X
	`ppc560xB`	X	X	X	X
	`ppc560xC`	X	X	X	X
	`ppc560xE`	X		X	X
	`ppc560xP`	X	X	X	X
	`ppc560xS`	X	X	X	X
	`ppc563xM`	X	X	X	X
	`ppc564xA`	X		X	X
	`ppc564xB`	X		X	X
	`ppc564xC`	X		X	X
	`ppc564xL_LSM`	X		X	X
	`ppc564xL_DPM`	X		X	X
	`ppc564xS`	X		X	X
	`ppc5668`	X	X	X	X
	`ppc5674`	X	X	X	X
	`ppc567xK_LSM`	X		X	X
	`ppc567xK_DPM`	X		X	X
	`ppc567xR`	X		X	X
	`ppc5744P`	X		X	X
	`ppc5746M`	X		X	X
	`ppc5744K`	X		X	

Family	Device Name	GHP	STPv1	STPv3	Trace
	ppc5746R	X		X	X
	ppc5748G	X		X	
	ppc5775K	X		X	X
	ppc5777M	X		X	X
	spc56AP60	X		X	X
	spc570S50	X		X	
	spc572L64	X		X	
	spc574K72	X		X	
e300	ppc5121	X			
	ppc5125	X			
	ppc8308	X			
	ppc8313	X			
	ppc8314	X			
	ppc8315	X			
	ppc8321	X			
	ppc8323	X			
	ppc8343	X			
	ppc8347	X			
	ppc8349	X			
	ppc8358	X			
	ppc8377	X			
	ppc8378	X			
	ppc8379	X			
	ppc8360	X			

Family	Device Name	GHP	STPv1	STPv3	Trace
e500	ppc8533	X			
	ppc8536	X			
	ppc8540	X			
	ppc8541	X			
	ppc8543	X			
	ppc8544	X			
	ppc8545	X			
	ppc8547	X			
	ppc8548	X			
	ppc8555	X			
	ppc8560	X			
	ppc8568	X			
	ppc8569	X			
	ppc8572	X			
	ppcP1011	X			
	ppcP1020	X			
	ppcP2010	X			
	ppcP2020	X			
	ppcP2041	X		X	X
	ppcP3041	X		X	X
	ppcP4040	X		X	X
	ppcP4080	X		X	X
	ppcP5020	X		X	X
	ppcP5040	X		X	X
	ppcT4160	X		X	
	ppcT4240	X		X	
e600	ppc8610	X			
	ppc8641	X			
	ppc8641D	X			

* When specifying a PowerPC 4xx target by name, you can use several different suffixes to give the probe more information about your target. The following suffixes are allowed:

Suffix	Description
fpu	Indicates that this target has a floating point unit.
nofpu	Indicates that this target does not have a floating point unit.
raw	Indicates that this target does not have any core select bits.

Separate the suffix from the name with a period:

```
set target ppc440x5.raw
```

Separate multiple suffixes with commas:

```
set target ppc440x5.raw,fpu
```

Do not put spaces before or after a suffix.

** Use the `ppc750` setting when connecting to a `ppc750`, `ppc750L`, `ppc750CX(e)`, `ppc750GX`, or `ppc750FX`. Specify your exact target board with the `target_rev` option (see “PowerPC” on page 124). The cores have to be distinguished because each core has its caches in different locations.

*** The device name `ppc603` is a generic type that currently defaults to PowerPC 603ev, the only 603 variant currently supported.

Supported SH Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
SH	`sh7206`	X	X		X
	`sh7261`	X	X		X
	`sh7263`	X	X		X
	`sh7780`	X	X		X
	`sh7780_8`	X	X		X

Supported V800 Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
V800	`v850e-trace`				X

Note

Trace is supported with STPv1 probes only. Run control is not supported.

Supported x86 Targets

Family	Device Name	GHP	STPv1	STPv3	Trace
Intel	`atom`	X			
	`atom_pv`	X			
	`core2`	X			
	`nehalem`	X			
	`sandy_bridge`	X			

Supported Generic JTAG Devices

Family	Device Name	GHP	STPv1	STPv3	Trace
Other	`other`	X	X	X	

Note

For more information about specifying generic JTAG devices, see "Specifying Bypassed Devices" on page 77.

Green Hills Probe Target Adapter Types

The following table lists appropriate *adapter_type* values for targets currently supported by the Green Hills Probe. Use these values with the **set adapter** command (see "Setting the Target Adapter Type" on page 74).

adapter_type	Target	Pin-out
auto	Detects the adapter type. This is the default and the recommended setting. It is the only correct setting for TraceEverywhere (TE) cabling.	
adi	Blackfin targets that use a 14-pin debug connection.	"Analog Devices DSP" on page 239
arc_15	ARC targets that use a JTAG connection.	
arm-20	ARM targets, including XScale, that use a 20-pin debug connection.	"ARM Legacy 20-Pin" on page 243
arm-14	ARM targets that use a 14-pin debug connection.	"ARM Legacy 14-Pin" on page 241
cf-bdm	ColdFire targets that use a BDM connection.	"ColdFire BDM" on page 259
cop	PowerPC targets, such as PowerPC82xx series processors, that use a 16-pin COP connection.	"PowerPC COP" on page 266
ejtag20-12	MIPS or Lexra targets that use a 12-pin EJTAG v2.0 or v1.53 connection.	"MIPS EJTAGV2.0 and lower" on page 255
ejtag20-20	MIPS targets that use a 20-pin EJTAG v2.0 or v1.53 connection.	
ejtag25	MIPS targets that use a 14-pin EJTAG v2.5 or v2.6 connection.	"MIPS EJTAG V2.5 and higher" on page 257
eonce	Targets that use a 14-pin EOnCE connection.	"EOnCE" on page 273
ppc-bdm	PowerPC targets that use a 10-pin BDM connection.	"PowerPC BDM" on page 264
sh	SH targets that use a 14-pin H-UDI connection.	"Renesas H-UDI" on page 277
ti	Texas Instruments OMAP targets.	"Texas Instruments DSP" on page 275

For more information about each adapter type, see "CPU Families and Debug Connectors" on page 237.

Note

All x86 targets use the `auto` setting, because they require TE cabling. This list may not be comprehensive. If your target and adapter type are not listed, contact your Green Hills Software Sales Representative.

SuperTrace Probe Trace Pod Types

The following table lists appropriate *adapter_type* values for targets currently supported by the SuperTrace Probe. Use these values with the **set adapter** command (see "Setting the Target Adapter Type" on page 74).

adapter_type	Trace pod (adapter)	Pin-out
auto	Trace pod for any supported target. The probe automatically detects the adapter type. This is the default and the recommended setting. This is the correct setting for TraceEverywhere (TE) and legacy cabling.	
arm-coresight	Adapter for ARM CoreSight targets.	"ARM ETM/PTM/CoreSight MICTOR" on page 280
arm-etm	Adapter for ARM ETM targets	"ARM ETM/PTM/CoreSight MICTOR" on page 280
arm-etm3	Adapter for ARM ETMv3 targets.	"ARM ETM/PTM/CoreSight MICTOR" on page 280
coldfire-trace	Adapter for ColdFire targets	"ColdFire BDM" on page 259
ppc440-trace	Adapter for PowerPC 4xx targets	"PowerPC 405/440/460 MICTOR" on page 286
nexus-mictor	Adapter for PowerPC 55xx targets	"PowerPC 55xx/56xx/57xx Nexus" on page 289
swd-coresight	Adapter for ARM CoreSight targets with an SWD debug interface	"ARM CoreSight 20-Pin" on page 245
swd-etm3	Adapter for ARM ETMv3 targets with an SWD debug interface	"ARM ETM/PTM/CoreSight MICTOR" on page 280
v850e-trace	Adapter for Renesas V850E targets	
sh-trace	Adapter for SH targets	

Note

For the Renesas V850E, the SuperTrace Probe only collects trace data, and does not support run-control operations. To fully debug the Renesas

V850E, you must use a separate run-control probe and debug server. Currently, only the Midas IECUBE run-control probes are supported for use in tandem with the SuperTrace Probe.

Note

This list may not be comprehensive. If your target and adapter type are not listed, contact your Green Hills Software Sales Representative.

Trace Feature Support

The tables in this section list which trace features are supported on trace targets.

In addition to the listed features, all trace targets support timestamps on SuperTrace Probe v3, and data value trace implies TimeMachine register and memory reconstruction.

	ETMv3.4	ETMv3.x				ETMv1.x	PTM 1.x
	Cortex-M3, M4	PJ4	Cortex-A8	Cortex-A5[#], Cortex-R4	ARM9, ARM11	ARM7, ARM9	Cortex-A9, Cortex-A15
PC trace	X	X	X	X	X	X	X
Data addresses		X	X	X	X	X	
Data values				X	X	X	
Trigger, execution [+]	X	X	X	X	X	X	X[++]
Trigger, read/write [+]	X*	X*	X*	X*	X*	X	X*[++]
Trigger, value [+]	X*	X*	X*	X*	X*	X	X*[++]
Cycle accurate	X	X	X	X	X	X	X**
External triggers	X	X	X	X	X	X	X
INTEGRITY		X	X	X	X	X	X
Gap reconstruction						X	
Filters		X*	X*	X*	X*	X	X*
Port Widths	1, 2, 4	1, 2, 4, 8, 16	4, 8, 16	4, 8, 16	4, 8, 16	4, 8, 16	4, 8, 16

* Some targets do not support this feature; it is implementation dependent.

** Cumulative cycle counts are provided at each branch

[+] Triggers are not supported on targets for which the trace macrocell trigger output cannot be connected to the trace sink's trigger input, such as the TI OMAP3, TI OMAP4 and Freescale Vybrid.

[++] PTM triggers are not precise; they occur in the trace stream several instructions after the triggering event.

[#] Cortex-A5 trace is only supported to the ETB, not the TPIU.

	PPC		
	Qorivva Nexus	**QorIQ Nexus**	**PPC4xx**
PC trace	X	X	X
Data addresses	X*		
Data values	X*		
Trigger, execution	X	X	X
Trigger, read/write	X	X	X
Trigger, value			X
Cycle accurate			X
External triggers	X	X	X**
INTEGRITY	X	X	X
Gap reconstruction			
Filters	X	X	
Port Widths	2, 4, 8, 12, 16	N/A	N/A

* Data trace is not supported on e200z0 or e200z1 cores. This is a limitation of the chip.

** External trigger in is not supported on PPC405

	ColdFire
PC trace	X
Data addresses	
Data values	X*
Trigger, execution	X
Trigger, read/write	X
Trigger, value	X
Cycle accurate	X
External triggers	X
INTEGRITY	X**
Gap reconstruction	
Filters	
Port Widths	N/A

* When caches are enabled, data values that hit the cache are not output in trace. In addition, the trace output may drop data values due to trace port bandwidth constraints. Time Machine cannot reconstruct register and memory values.

** Trace can identify the AddressSpace, but not the specific Task.

Appendix F

Probe Error Codes

The following table lists probe error codes along with a brief description of what the code indicates.

Code	Indication
1	general error
2	target is not in debug mode
3	target is already in debug mode
4	could not allocate memory
5	ambiguous command
6	no match
7	invalid address
8	general FPGA error
9	EJTAG error
10	timeout
11	run
12	interrupt
13	test failed
14	invalid parameter
15	configuration verification failure
16	no target selected
17	the target is not running
18	the target is not stopped
19	this functionality has not yet been implemented
20	this functionality is not supported by the target
21	NULL or OTHER target; requested operation not available
22	NULL communications device
23	invalid memory index
24	invalid address range
25	no more breakpoint slots available
26	configuration integrity error
27	CRC check error
28	flash error

Code	Indication
29	ThreadX general error
30	Invalid/unexpected message type
31	unexepected heap write pending
32	unexepected read pending
33	unexepected heap read pending
34	unexepected instruction read pending
35	dbscript error
36	comm thread exited
37	permission denied by memory range
38	could not find a valid access size to use
39	port reset
40	this functionality has not yet been implemented
41	the target is not supported
42	failed to single step the target
43	bad value for this register
44	bad register ID
45	shadow not equal to register value
46	timeout in memory access
47	failed to halt the target
48	unhandled target-specific command
49	no template available for this target
50	failed to acquire the target mutex
51	failed to release the target mutex
52	failed to create the target mutex
53	failed to delete the target mutex
54	invalid target ID
55	memory read error
56	memory write error
57	memory alignment error
58	PPC COP error

Code	Indication
59	this register is not currently available for this operation
60	target power off
61	register is read only
62	breakpoint already set at this address
63	end of file encountered
64	option too long
65	no set command
66	could not open file
67	reboot requested
68	wrong protocol version
69	PC was unexpectedly reset
70	exception
71	probe output pins are disabled
72	register is write only
73	this feature is not currently enabled
75	Probe has shut down. Please contact Green Hills technical support for more information.
76	INTEGRITY error
77	error translating virtual address
78	cannot modify read-only option
81	no space left on device
82	the line is a comment
83	connection rejected
85	protocol consistency check failure
86	pod link down

Appendix G

ARM Register Names

Contents

General Information

The Cortex-A9 register that contains the base address for the PLE program new channel operation (`Rt`) is called `ple_prog_new_chan` by the probe.

Register Names For ARM1136

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	0	cp15_id
cp15	0	c0	c0	1	cp15_cachetype
cp15	0	c0	c0	2	cp15_tcm_status
cp15	0	c0	c0	3	cp15_tlb_type
cp15	0	c1	c0	0	control
cp15	0	c1	c0	1	cp15_aux_control
cp15	0	c1	c0	2	cp_access
cp15	0	c2	c0	0	cp15_ttbase0
cp15	0	c2	c0	1	cp15_ttbase1
cp15	0	c2	c0	2	cp15_ttbase_ctrl
cp15	0	c3	c0	0	cp15_dactl
cp15	0	c5	c0	0	cp15_dfsr
cp15	0	c5	c0	1	cp15_ifsr
cp15	0	c6	c0	0	cp15_far
cp15	0	c6	c0	1	cp15_wfar
cp15	0	c9	c0	0	cp15_dcache_lock
cp15	0	c9	c0	1	cp15_icache_lock
cp15	0	c9	c1	0	cp15_dtcm_region
cp15	0	c9	c1	1	cp15_itcm_region
cp15	0	c10	c0	0	cp15_tlb_lock
cp15	0	c11	c0	0	cp15_dma_st_present
cp15	0	c11	c0	1	cp15_dma_st_queued
cp15	0	c11	c0	2	cp15_dma_st_running
cp15	0	c11	c0	3	cp15_dma_st_interrupting

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c11	c1	0	cp15_dma_user_access
cp15	0	c11	c2	0	cp15_dma_ch_number
cp15	0	c11	c3	0	cp15_dma_enable_stop
cp15	0	c11	c3	1	cp15_dma_enable_start
cp15	0	c11	c3	2	cp15_dma_enable_clear
cp15	0	c11	c4	0	cp15_dma_control
cp15	0	c11	c5	0	cp15_dma_istart
cp15	0	c11	c6	0	cp15_dma_estart
cp15	0	c11	c7	0	cp15_dma_iend
cp15	0	c11	c8	0	cp15_dma_status
cp15	0	c11	c15	0	cp15_dma_contextid
cp15	0	c13	c0	0	cp15_pid
cp15	0	c13	c0	1	cp15_contextid
cp15	0	c15	c2	0	cp15_dm_remap
cp15	0	c15	c2	1	cp15_im_remap
cp15	0	c15	c2	2	cp15_dma_remap
cp15	0	c15	c2	4	cp15_periph_remap
cp15	0	c15	c12	0	cp15_perfmon_ctrl
cp15	0	c15	c12	1	cp15_cycle_count
cp15	0	c15	c12	2	cp15_count0
cp15	0	c15	c12	3	cp15_count1
cp15	3	c15	c0	0	cp15_dcache_debug
cp15	3	c15	c0	1	cp15_icache_debug
cp15	3	c15	c2	0	cp15_dtag_read_op
cp15	3	c15	c2	1	cp15_itag_read_op
cp15	3	c15	c4	1	cp15_icache_read_op
cp15	3	c15	c8	0	cp15_icache_mvr
cp15	3	c15	c10	0	cp15_iscache_mvr
cp15	3	c15	c12	0	cp15_dcache_mvr
cp15	3	c15	c14	0	cp15_dscache_mvr

CP	OP1	CRn	CRm	OP2	Name
cp15	5	c15	c4	0	cp15_dmicrotlb_entry_op
cp15	5	c15	c4	1	cp15_imicrotlb_entry_op
cp15	5	c15	c4	2	cp15_read_maintlb_entry
cp15	5	c15	c4	4	cp15_write_maintlb_entry
cp15	5	c15	c5	0	cp15_dmicrotlb_va
cp15	5	c15	c5	1	cp15_imicrotlb_va
cp15	5	c15	c5	2	cp15_maintlb_va
cp15	5	c15	c6	0	cp15_dmicrotlb_pa
cp15	5	c15	c6	1	cp15_imicrotlb_pa
cp15	5	c15	c6	2	cp15_maintlb_pa
cp15	5	c15	c7	0	cp15_dmicrotlb_attrib
cp15	5	c15	c7	1	cp15_imicrotlb_attrib
cp15	5	c15	c7	2	cp15_maintlb_attrib
cp15	5	c15	c14	0	cp15_maintlb_mvr
cp15	7	c15	c0	0	cp15_cache_debug_ctrl
cp15	7	c15	c1	0	cp15_tlb_debug_ctrl

Register Names For ARM1156

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	0	cp15_id
cp15	0	c0	c0	1	cp15_cachetype
cp15	0	c0	c0	2	cp15_tcm_status
cp15	0	c0	c0	4	cp15_mpu_type
cp15	0	c1	c0	0	control
cp15	0	c1	c0	1	cp15_aux_control
cp15	0	c1	c0	2	cp_access
cp15	0	c5	c0	0	cp15_dfsr
cp15	0	c5	c0	1	cp15_ifsr
cp15	0	c6	c0	0	cp15_far

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c6	c0	1	cp15_wfar
cp15	0	c6	c0	2	cp15_ifar
cp15	0	c6	c1	0	cp15_region_base
cp15	0	c6	c1	2	cp15_region_size
cp15	0	c6	c1	4	cp15_region_access
cp15	0	c6	c2	0	cp15_region_number
cp15	0	c9	c1	0	cp15_dtcm_region
cp15	0	c9	c1	1	cp15_itcm_region
cp15	0	c13	c0	0	cp15_pid
cp15	0	c15	c12	0	cp15_perfmon_ctrl
cp15	0	c15	c12	1	cp15_cycle_count
cp15	0	c15	c12	2	cp15_count0
cp15	0	c15	c12	3	cp15_count1
cp15	7	c15	c0	0	cp15_cache_debug_ctrl

Register Names For ARM1176

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	0	cp15_id
cp15	0	c0	c0	1	cp15_cache_type
cp15	0	c0	c0	2	cp15_tcm_status
cp15	0	c0	c0	3	cp15_tlb_type
cp15	0	c0	c1	0	cp15_proc_feat0
cp15	0	c0	c1	1	cp15_proc_feat1
cp15	0	c0	c1	2	cp15_debug_feat0
cp15	0	c0	c1	3	cp15_aux_feat0
cp15	0	c0	c1	4	cp15_mm_feat0
cp15	0	c0	c1	5	cp15_mm_feat1
cp15	0	c0	c1	6	cp15_mm_feat2
cp15	0	c0	c1	7	cp15_mm_feat3

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c2	0	cp15_inst_set_feat0
cp15	0	c0	c2	1	cp15_inst_set_feat1
cp15	0	c0	c2	2	cp15_inst_set_feat2
cp15	0	c0	c2	3	cp15_inst_set_feat3
cp15	0	c0	c2	4	cp15_inst_set_feat4
cp15	0	c0	c2	5	cp15_inst_set_feat5
cp15	0	c1	c0	0	control
cp15	0	c1	c0	1	cp15_aux_control
cp15	0	c1	c0	2	cp_access
cp15	0	c1	c1	0	cp15_secure_config
cp15	0	c1	c1	1	cp15_secure_debug_enable
cp15	0	c1	c1	2	cp15_non_secure_access_ctrl
cp15	0	c2	c0	0	cp15_ttbase0
cp15	0	c2	c0	1	cp15_ttbase1
cp15	0	c2	c0	2	cp15_ttbase_ctrl
cp15	0	c3	c0	0	cp15_dactl
cp15	0	c5	c0	0	cp15_dfsr
cp15	0	c5	c0	1	cp15_ifsr
cp15	0	c6	c0	0	cp15_far
cp15	0	c6	c0	2	cp15_ifar
cp15	0	c9	c0	0	cp15_dcache_lock
cp15	0	c9	c0	1	cp15_icache_lock
cp15	0	c9	c1	0	cp15_dtcm_region
cp15	0	c9	c1	1	cp15_itcm_region
cp15	0	c9	c1	2	cp15_dtcm_ns_ctrl_access
cp15	0	c9	c1	3	cp15_itcm_ns_ctrl_access
cp15	0	c9	c2	0	cp15_tcm_sel
cp15	0	c9	c8	0	cp15_cache_behavior_override
cp15	0	c10	c0	0	cp15_tlb_lock
cp15	0	c10	c2	0	cp15_primary_region_remap

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c10	c2	1	cp15_normal_memory_remap
cp15	0	c11	c0	0	cp15_dma_id_st_present
cp15	0	c11	c0	1	cp15_dma_id_st_queued
cp15	0	c11	c0	2	cp15_dma_id_st_running
cp15	0	c11	c0	3	cp15_dma_id_st_interrupting
cp15	0	c11	c1	0	cp15_dma_user_access
cp15	0	c11	c2	0	cp15_dma_ch_number
cp15	0	c11	c4	0	cp15_dma_control
cp15	0	c11	c5	0	cp15_dma_istart
cp15	0	c11	c6	0	cp15_dma_estart
cp15	0	c11	c7	0	cp15_dma_iend
cp15	0	c11	c8	0	cp15_dma_status
cp15	0	c11	c15	0	cp15_dma_contextid
cp15	0	c12	c0	0	cp15_vector_base_address
cp15	0	c12	c0	1	cp15_monitor_vector_base_address
cp15	0	c12	c1	0	cp15_interrupt_status
cp15	0	c13	c0	0	cp15_pid
cp15	0	c13	c0	1	cp15_contextid
cp15	0	c13	c0	2	cp15_user_rw_threadid
cp15	0	c13	c0	3	cp15_user_ro_threadid
cp15	0	c13	c0	4	cp15_priv_threadid
cp15	0	c15	c2	4	cp15_peripheral_mem_remap
cp15	0	c15	c9	0	cp15_access_val_ctrl
cp15	0	c15	c12	0	cp15_perfmon_ctrl
cp15	0	c15	c12	1	cp15_cycle_count
cp15	0	c15	c12	2	cp15_count0
cp15	0	c15	c12	3	cp15_count1
cp15	0	c15	c12	4	cp15_reset_count
cp15	0	c15	c12	5	cp15_irq_count
cp15	0	c15	c12	6	cp15_fiq_count

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c15	c12	7	cp15_debug_count
cp15	0	c15	c13	1	cp15_reset_count_start
cp15	0	c15	c13	2	cp15_irq_count_start
cp15	0	c15	c13	3	cp15_reset_irq_count_start
cp15	0	c15	c13	4	cp15_fiq_count_start
cp15	0	c15	c13	5	cp15_reset_fiq_count_start
cp15	0	c15	c13	6	cp15_irq_fiq_count_start
cp15	0	c15	c13	7	cp15_reset_irq_fiq_count_start
cp15	0	c15	c14	0	cp15_sys_val_cache_size_mask
cp15	1	c15	c13	8	cp15_debug_count_start
cp15	2	c15	c13	1	cp15_reset_count_stop
cp15	2	c15	c13	2	cp15_irq_count_stop
cp15	2	c15	c13	3	cp15_reset_irq_count_stop
cp15	2	c15	c13	4	cp15_fiq_count_stop
cp15	2	c15	c13	5	cp15_reset_fiq_count_stop
cp15	2	c15	c13	6	cp15_irq_fiq_count_stop
cp15	2	c15	c13	7	cp15_reset_irq_fiq_count_stop
cp15	3	c15	c8	0	cp15_icache_mvr
cp15	3	c15	c12	0	cp15_dcache_mvr
cp15	3	c15	c13	8	cp15_debug_count_stop
cp15	5	c15	c4	2	cp15_tlb_lock_index
cp15	5	c15	c5	2	cp15_tlb_lock_va
cp15	5	c15	c6	2	cp15_tlb_lock_pa
cp15	5	c15	c7	2	cp15_tlb_lock_attrib

Register Names For ARM11MP

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	0	cp15_id
cp15	0	c0	c0	1	cp15_cache_type

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	3	cp15_tlb_type
cp15	0	c0	c0	5	cp15_cpu_id
cp15	0	c0	c1	0	cp15_id_pfr0
cp15	0	c0	c1	1	cp15_id_pfr1
cp15	0	c0	c1	2	cp15_id_dfr0
cp15	0	c0	c1	4	cp15_id_mmfr0
cp15	0	c0	c1	5	cp15_id_mmfr1
cp15	0	c0	c1	6	cp15_id_mmfr2
cp15	0	c0	c1	7	cp15_id_mmfr3
cp15	0	c0	c2	0	cp15_id_isar0
cp15	0	c0	c2	1	cp15_id_isar1
cp15	0	c0	c2	2	cp15_id_isar2
cp15	0	c0	c2	3	cp15_id_isar3
cp15	0	c0	c2	4	cp15_id_isar4
cp15	0	c1	c0	0	control
cp15	0	c1	c0	1	cp15_aux_control
cp15	0	c1	c0	2	cp_access
cp15	0	c2	c0	0	cp15_ttbr0
cp15	0	c2	c0	1	cp15_ttbr1
cp15	0	c2	c0	2	cp15_ttbcr
cp15	0	c3	c0	0	cp15_dactl
cp15	0	c5	c0	0	cp15_dfsr
cp15	0	c5	c0	1	cp15_ifsr
cp15	0	c6	c0	0	cp15_far
cp15	0	c6	c0	1	cp15_wfar
cp15	0	c7	c5	0	cp15_icache_inval_all
cp15	0	c7	c5	1	cp15_icache_inval_mva
cp15	0	c7	c5	2	cp15_icache_inval_index
cp15	0	c7	c6	0	cp15_dcache_inval_all
cp15	0	c7	c6	1	cp15_dcache_inval_mva

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c7	c6	2	cp15_dcache_inval_index
cp15	0	c7	c7	0	cp15_l1cache_inval_all
cp15	0	c7	c10	0	cp15_dcache_clean_all
cp15	0	c7	c10	1	cp15_dcache_clean_mva
cp15	0	c7	c10	2	cp15_dcache_clean_index
cp15	0	c7	c14	0	cp15_dcache_clean_inval_all
cp15	0	c7	c14	1	cp15_dcache_clean_inval_mva
cp15	0	c7	c14	2	cp15_dcache_clean_inval_index
cp15	0	c9	c0	0	cp15_dcache_lock
cp15	0	c10	c0	0	cp15_tlb_lock
cp15	0	c10	c2	0	cp15_primary_region_remap
cp15	0	c10	c2	1	cp15_normal_region_remap
cp15	0	c13	c0	0	cp15_pid
cp15	0	c13	c0	1	cp15_contextid
cp15	0	c13	c0	2	cp15_user_rw_threadid
cp15	0	c13	c0	3	cp15_user_ro_threadid
cp15	0	c13	c0	4	cp15_priv_threadid
cp15	0	c15	c12	0	cp15_perfmon_ctrl
cp15	0	c15	c12	1	cp15_cycle_count
cp15	0	c15	c12	2	cp15_count0
cp15	0	c15	c12	3	cp15_count1
cp15	5	c15	c4	2	cp15_read_maintlb_entry
cp15	5	c15	c4	4	cp15_write_maintlb_entry
cp15	5	c15	c5	2	cp15_maintlb_va
cp15	5	c15	c6	2	cp15_maintlb_pa
cp15	5	c15	c7	2	cp15_maintlb_attrib
cp15	7	c15	c1	0	cp15_tlb_debug_ctrl

Register Names For PJ4v6

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	0	cp15_id
cp15	0	c0	c0	1	cp15_cachetype
cp15	0	c0	c0	3	cp15_tlb_type
cp15	0	c0	c1	0	cp15_proc_feat0
cp15	0	c0	c1	1	cp15_proc_feat1
cp15	0	c0	c1	2	cp15_debug_feat0
cp15	0	c0	c1	3	cp15_aux_feat0
cp15	0	c0	c1	4	cp15_mm_feat0
cp15	0	c0	c1	5	cp15_mm_feat1
cp15	0	c0	c1	6	cp15_mm_feat2
cp15	0	c0	c1	7	cp15_mm_feat3
cp15	0	c0	c2	0	cp15_inst_set_feat0
cp15	0	c0	c2	1	cp15_inst_set_feat1
cp15	0	c0	c2	2	cp15_inst_set_feat2
cp15	0	c0	c2	3	cp15_inst_set_feat3
cp15	0	c0	c2	4	cp15_inst_set_feat4
cp15	0	c0	c2	5	cp15_inst_set_feat5
cp15	0	c1	c0	0	control
cp15	0	c1	c0	1	cp15_aux_control
cp15	0	c1	c0	2	cp_access
cp15	0	c1	c1	0	cp15_secure_config
cp15	0	c1	c1	1	cp15_secure_debug_enable
cp15	0	c1	c1	2	cp15_nonsecure_access_ctl
cp15	0	c2	c0	0	cp15_ttbase0
cp15	0	c2	c0	1	cp15_ttbase1
cp15	0	c2	c0	2	cp15_ttbase_ctrl
cp15	0	c3	c0	0	cp15_dactl
cp15	0	c5	c0	0	cp15_dfsr
cp15	0	c5	c0	1	cp15_ifsr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c6	c0	0	cp15_far
cp15	0	c6	c0	1	cp15_wfar
cp15	0	c6	c0	2	cp15_ifar
cp15	0	c7	c5	0	cp15_icache_inval_all
cp15	0	c7	c5	1	cp15_icache_inval_mva
cp15	0	c7	c5	2	cp15_icache_inval_way
cp15	0	c7	c6	0	cp15_dcache_inval_all
cp15	0	c7	c6	1	cp15_dcache_inval_mva
cp15	0	c7	c6	2	cp15_dcache_inval_way
cp15	0	c7	c7	0	cp15_l1cache_inval_all
cp15	0	c7	c10	0	cp15_dcache_clean_all
cp15	0	c7	c10	1	cp15_dcache_clean_mva
cp15	0	c7	c10	2	cp15_dcache_clean_way
cp15	0	c7	c13	1	cp15_dcache_clean_inval_mva
cp15	0	c7	c13	2	cp15_dcache_clean_inval_index
cp15	0	c7	c14	0	cp15_dcache_clean_inval_all
cp15	1	c7	c7	0	cp15_l2_inval_all
cp15	1	c7	c7	1	cp15_l2_inval_mva
cp15	1	c7	c7	2	cp15_l2_inval_way
cp15	1	c7	c7	3	cp15_l2_inval_pa
cp15	1	c7	c11	0	cp15_l2_clean_all
cp15	1	c7	c11	1	cp15_l2_clean_mva
cp15	1	c7	c11	2	cp15_l2_clean_way
cp15	1	c7	c11	3	cp15_l2_clean_pa
cp15	1	c7	c15	1	cp15_l2_clean_inval_mva
cp15	1	c7	c15	2	cp15_l2_clean_inval_index
cp15	1	c7	c15	3	cp15_l2_clean_inval_pa
cp15	0	c9	c0	0	cp15_dcache_lock
cp15	0	c9	c0	1	cp15_icache_lock
cp15	0	c9	c12	0	cp15_perfmon_ctrl

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c9	c12	1	cp15_count_enable_set
cp15	0	c9	c12	2	cp15_count_enable_clear
cp15	0	c9	c12	3	cp15_overflow_flag
cp15	0	c9	c12	4	cp15_software_incr
cp15	0	c9	c12	5	cp15_perfctr_selection
cp15	0	c9	c13	0	cp15_cycle_count
cp15	0	c9	c13	1	cp15_event_selection
cp15	0	c9	c13	2	cp15_perfmon_counter
cp15	0	c9	c14	0	cp15_user_enable
cp15	0	c9	c14	1	cp15_int_enable_set
cp15	0	c9	c14	2	cp15_int_enable_clear
cp15	0	c10	c0	0	cp15_tlb_lock
cp15	0	c10	c2	0	cp15_primary_region_remap
cp15	0	c10	c2	1	cp15_normal_memory_remap
cp15	0	c12	c1	0	cp15_interrupt_status
cp15	0	c13	c0	0	cp15_pid
cp15	0	c13	c0	1	cp15_contextid
cp15	0	c13	c0	2	cp15_user_rw_threadid
cp15	0	c13	c0	3	cp15_user_ro_threadid
cp15	0	c13	c0	4	cp15_privileged_rw_threadid
cp15	1	c15	c1	0	cp15_control_config
cp15	1	c15	c1	1	cp15_aux_debug_modes
cp15	1	c15	c2	0	cp15_aux_func_modes
cp15	1	c15	c9	6	cp15_l2_err_counter
cp15	1	c15	c9	7	cp15_l2_err_threshold
cp15	1	c15	c10	7	cp15_l2_way_lockdown
cp15	1	c15	c11	7	cp15_l2_err_capture
cp15	1	c15	c12	0	cp15_cpu_id_code_ext
cp15	5	c15	c4	2	cp15_read_maintlb_entry
cp15	5	c15	c5	2	cp15_maintlb_va

CP	OP1	CRn	CRm	OP2	Name
cp15	5	c15	c6	2	cp15_maintlb_pa
cp15	5	c15	c7	2	cp15_maintlb_attrib

Register Names For PJ4v7

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	0	midr
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	3	tlbtr
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr1
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mmfr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	0	c0	c2	5	id_isar5
cp15	0	c0	c2	6	id_isar6
cp15	0	c0	c2	7	id_isar7
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr
cp15	1	c0	c0	7	aidr

CP	OP1	CRn	CRm	OP2	Name
cp15	2	c0	c0	0	csselr
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c1	c1	0	scr
cp15	0	c1	c1	1	sder
cp15	0	c1	c1	2	nsacr
cp15	0	c2	c0	0	ttbr0
cp15	0	c2	c0	1	ttbr1
cp15	0	c2	c0	2	ttbcr
cp15	0	c3	c0	0	dacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	0	c6	c0	0	dfar
cp15	0	c6	c0	2	ifar
cp15	0	c7	c4	0	par
cp15	0	c7	c8	0	v2pcwpr
cp15	0	c7	c8	1	v2pcwpw
cp15	0	c7	c8	2	v2pcwur
cp15	0	c7	c8	3	v2pcwuw
cp15	0	c7	c8	4	v2powpr
cp15	0	c7	c8	5	v2powpw
cp15	0	c7	c8	6	v2powur
cp15	0	c7	c8	7	v2powuw
cp15	0	c8	c5	0	itlbiall
cp15	0	c8	c5	1	itlbimva
cp15	0	c8	c5	2	itlbiasid
cp15	0	c8	c6	0	dtlbiall

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c8	c6	1	dtlbimva
cp15	0	c8	c6	2	dtlbiasid
cp15	0	c9	c0	0	dcache_lockdown
cp15	0	c9	c0	1	icache_lockdown
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr
cp15	0	c9	c13	0	pmccntr
cp15	0	c9	c13	1	pmxevtyper
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	1	c9	c0	0	l2_lockdown
cp15	1	c9	c0	2	l2_aux_control
cp15	0	c10	c0	0	tlb_lockdown
cp15	0	c10	c1	0	d_tlb_preload
cp15	0	c10	c1	1	i_tlb_preload
cp15	0	c10	c2	0	prrr
cp15	0	c10	c2	1	nmrr
cp15	0	c12	c1	0	isr
cp15	0	c13	c0	0	fcseidr
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw
cp15	5	c15	c4	2	tlb_lockdown_index
cp15	5	c15	c5	2	tlb_lockdown_va

CP	OP1	CRn	CRm	OP2	Name
cp15	5	c15	c6	2	tlb_lockdown_pa
cp15	5	c15	c7	2	tlb_lockdown_attr

Register Names For Cortex-R4

CP	OP1	CRn	CRm	OP2	Name
cp14	6	c0	c0	0	teecr
cp14	6	c1	c0	0	teehbr
cp15	0	c0	c0	0	midr
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	4	mpuir
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c0	6	revidr
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr1
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mmfr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	0	c0	c2	5	id_isar5
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr

CP	OP1	CRn	CRm	OP2	Name
cp15	1	c0	c0	7	aidr
cp15	2	c0	c0	0	csselr
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	0	c6	c0	0	dfar
cp15	0	c6	c0	2	ifar
cp15	0	c6	c1	0	drbar
cp15	0	c6	c1	1	irbar
cp15	0	c6	c1	2	drsr
cp15	0	c6	c1	3	irsr
cp15	0	c6	c1	4	dracr
cp15	0	c6	c1	5	iracr
cp15	0	c6	c2	0	rgnr
cp15	0	c7	c5	0	iciallu
cp15	0	c7	c5	1	icimvau
cp15	0	c7	c5	4	cp15isb
cp15	0	c7	c5	6	bpiall
cp15	0	c7	c5	7	bpimva
cp15	0	c7	c6	1	dcimvac
cp15	0	c7	c6	2	dcisw
cp15	0	c7	c10	1	dccmvac
cp15	0	c7	c10	2	dccsw
cp15	0	c7	c10	4	cp15dsb
cp15	0	c7	c10	5	cp15dmb
cp15	0	c7	c11	1	dccmvau

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c7	c14	1	dccimvac
cp15	0	c7	c14	2	dccisw
cp15	0	c9	c1	0	dtcm_region
cp15	0	c9	c1	1	itcm_region
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	3	pmovsr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr
cp15	0	c9	c12	6	pmceid0
cp15	0	c9	c12	7	pmceid1
cp15	0	c9	c13	0	pmccntr
cp15	0	c9	c13	1	pmxevtyper
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw

Register Names For Cortex-R5

CP	OP1	CRn	CRm	OP2	Name
cp14	7	c0	c0	0	jidr
cp14	7	c1	c0	0	joscr
cp14	7	c2	c0	0	jmcr
cp15	0	c0	c0	0	midr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	4	mpuir
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c0	6	revidr
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr0
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mffr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr
cp15	1	c0	c0	7	aidr
cp15	2	c0	c0	0	csselr
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	0	c6	c0	0	dfar

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c6	c0	2	ifar
cp15	0	c6	c1	0	drbar
cp15	0	c6	c1	2	drsr
cp15	0	c6	c1	4	dracr
cp15	0	c6	c2	0	rgnr
cp15	0	c7	c1	0	icialluis
cp15	0	c7	c1	6	bpiallis
cp15	0	c7	c5	0	iciallu
cp15	0	c7	c5	1	icimvau
cp15	0	c7	c5	4	cp15isb
cp15	0	c7	c5	6	bpiall
cp15	0	c7	c5	7	bpimva
cp15	0	c7	c6	1	dcimvac
cp15	0	c7	c6	2	dcisw
cp15	0	c7	c10	1	dccmvac
cp15	0	c7	c10	2	dccsw
cp15	0	c7	c10	4	cp15dsb
cp15	0	c7	c10	5	cp15dmb
cp15	0	c7	c11	1	dccmvau
cp15	0	c7	c14	1	dccimvac
cp15	0	c7	c14	2	dccisw
cp15	0	c9	c1	0	btcmrr
cp15	0	c9	c1	1	atcmrr
cp15	0	c9	c2	0	tcmsr
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	3	pmovsr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c9	c13	0	pmccntr
cp15	0	c9	c13	1	pmxevtyper
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	0	c11	c0	0	slave_port_control
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw
cp15	0	c15	c0	0	secondary_auxiliary_control
cp15	0	c15	c0	1	normal_axi_peripheral_interface_region
cp15	0	c15	c0	2	virtual_axi_peripheral_interface_region
cp15	0	c15	c0	3	ahb_peripheral_interface_region
cp15	0	c15	c1	0	nval_irq_enable_set
cp15	0	c15	c1	1	nval_fiq_enable_set
cp15	0	c15	c1	2	nval_reset_enable_set
cp15	0	c15	c1	3	nval_debug_request_enable_set
cp15	0	c15	c1	4	nval_irq_enable_clear
cp15	0	c15	c1	5	nval_fiq_enable_clear
cp15	0	c15	c1	6	nval_reset_enable_clear
cp15	0	c15	c1	7	nval_debug_request_enable_clear
cp15	0	c15	c2	0	build_options_1
cp15	0	c15	c2	1	build_options_2
cp15	0	c15	c2	7	pin_options
cp15	0	c15	c3	0	correctable_fault_location
cp15	0	c15	c5	0	invalidate_all_data_cache
cp15	0	c15	c14	0	cache_size_override

Register Names For Cortex-A5

CP	OP1	CRn	CRm	OP2	Name
cp14	6	c0	c0	0	teecr
cp14	6	c1	c0	0	teehbr
cp14	7	c0	c0	0	jidr
cp14	7	c1	c0	0	joscr
cp14	7	c2	c0	0	jmcr
cp14	7	c3	c0	0	jpr
cp14	7	c4	c0	0	jcottr
cp15	0	c0	c0	0	midr
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	3	tlbtr
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c0	6	revidr
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr0
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mmfr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr
cp15	1	c0	c0	7	aidr

CP	OP1	CRn	CRm	OP2	Name
cp15	2	c0	c0	0	csselr
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c1	c1	0	scr
cp15	0	c1	c1	1	sder
cp15	0	c1	c1	2	nsacr
cp15	0	c1	c1	3	cp15_vcr
cp15	0	c2	c0	0	ttbr0
cp15	0	c2	c0	1	ttbr1
cp15	0	c2	c0	2	ttbcr
cp15	0	c3	c0	0	dacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	0	c6	c0	0	dfar
cp15	0	c6	c0	2	ifar
cp15	0	c7	c1	0	icialluis
cp15	0	c7	c1	6	bpiallis
cp15	0	c7	c4	0	par
cp15	0	c7	c5	0	iciallu
cp15	0	c7	c5	1	icimvau
cp15	0	c7	c5	4	cp15isb
cp15	0	c7	c5	6	bpiall
cp15	0	c7	c5	7	bpimva
cp15	0	c7	c6	1	dcimvac
cp15	0	c7	c6	2	dcisw
cp15	0	c7	c8	0	ats1cpr
cp15	0	c7	c8	1	ats1cpw

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c7	c8	2	ats1cur
cp15	0	c7	c8	3	ats1cuw
cp15	0	c7	c8	4	ats12nsopr
cp15	0	c7	c8	5	ats12nsopw
cp15	0	c7	c8	6	ats12nsour
cp15	0	c7	c8	7	ats12nsouw
cp15	0	c7	c10	1	dccmvac
cp15	0	c7	c10	2	dccsw
cp15	0	c7	c10	4	cp15dsb
cp15	0	c7	c10	5	cp15dmb
cp15	0	c7	c11	1	dccmvau
cp15	0	c7	c14	1	dccimvac
cp15	0	c7	c14	2	dccisw
cp15	0	c8	c3	0	tlbiallis
cp15	0	c8	c3	1	tlbimvais
cp15	0	c8	c3	2	tlbiasidis
cp15	0	c8	c3	3	tlbimvaais
cp15	0	c8	c7	0	tlbiall
cp15	0	c8	c7	1	tlbimva
cp15	0	c8	c7	2	tlbiasid
cp15	0	c8	c7	3	tlbimvaa
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	3	pmovsr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr
cp15	0	c9	c12	6	pmceid0
cp15	0	c9	c12	7	pmceid1
cp15	0	c9	c13	0	pmccntr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c9	c13	1	pmxevtyper
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	0	c10	c2	0	prrr
cp15	0	c10	c2	1	nmrr
cp15	0	c12	c0	0	vbar
cp15	0	c12	c0	1	mvbar
cp15	0	c12	c1	0	isr
cp15	0	c12	c1	1	vir
cp15	0	c13	c0	0	fcseidr
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw
cp15	3	c15	c0	0	data_register_0
cp15	3	c15	c0	1	data_register_1
cp15	3	c15	c2	0	data_cache_tag_read_operation_register
cp15	3	c15	c2	1	instruction_cache_tag_read_operation_register
cp15	3	c15	c4	0	data_cache_data_read_operation_register
cp15	3	c15	c4	1	instruction_cache_data_read_operation_register
cp15	3	c15	c4	2	tlb_data_read_operation_register
cp15	4	c15	c0	0	cbar
cp15	5	c15	c0	0	tlbhr

Register Names For Cortex-A8

CP	OP1	CRn	CRm	OP2	Name
cp14	6	c0	c0	0	teecr

CP	OP1	CRn	CRm	OP2	Name
cp14	6	c1	c0	0	teehbr
cp14	7	c0	c0	0	jidr
cp14	7	c1	c0	0	joscr
cp14	7	c2	c0	0	jmcr
cp15	0	c0	c0	0	midr
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	3	tlbtr
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c0	6	revidr
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr1
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mmfr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	0	c0	c2	5	id_isar5
cp15	0	c0	c2	6	id_isar6
cp15	0	c0	c2	7	id_isar7
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr
cp15	1	c0	c0	7	aidr
cp15	2	c0	c0	0	csselr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c1	c1	0	scr
cp15	0	c1	c1	1	sder
cp15	0	c1	c1	2	nsacr
cp15	0	c2	c0	0	ttbr0
cp15	0	c2	c0	1	ttbr1
cp15	0	c2	c0	2	ttbcr
cp15	4	c2	c1	2	vtcr
cp15	0	c3	c0	0	dacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	0	c6	c0	0	dfar
cp15	0	c6	c0	2	ifar
cp15	0	c7	c4	0	par
cp15	0	c7	c5	0	iciallu
cp15	0	c7	c5	1	icimvau
cp15	0	c7	c5	4	cp15isb
cp15	0	c7	c5	6	bpiall
cp15	0	c7	c5	7	bpimva
cp15	0	c7	c6	1	dcimvac
cp15	0	c7	c6	2	dcisw
cp15	0	c7	c8	0	v2pcwpr
cp15	0	c7	c8	1	v2pcwpw
cp15	0	c7	c8	2	v2pcwur
cp15	0	c7	c8	3	v2pcwuw
cp15	0	c7	c8	4	v2powpr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c7	c8	5	v2powpw
cp15	0	c7	c8	6	v2powur
cp15	0	c7	c8	7	v2powuw
cp15	0	c7	c10	1	dccmvac
cp15	0	c7	c10	2	dccsw
cp15	0	c7	c10	4	cp15dsb
cp15	0	c7	c10	5	cp15dmb
cp15	0	c7	c11	1	dccmvau
cp15	0	c7	c14	1	dccimvac
cp15	0	c7	c14	2	dccisw
cp15	0	c8	c5	0	itlbiall
cp15	0	c8	c5	1	itlbimva
cp15	0	c8	c5	2	itlbiasid
cp15	0	c8	c6	0	dtlbiall
cp15	0	c8	c6	1	dtlbimva
cp15	0	c8	c6	2	dtlbiasid
cp15	0	c8	c7	0	tlbiall
cp15	0	c8	c7	1	tlbimva
cp15	0	c8	c7	2	tlbiasid
cp15	0	c8	c7	3	tlbimvaa
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	3	pmovsr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr
cp15	0	c9	c12	6	pmceid0
cp15	0	c9	c12	7	pmceid1
cp15	0	c9	c13	0	pmccntr
cp15	0	c9	c13	1	pmxevtyper

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	0	c9	c14	3	pmovsset
cp15	1	c9	c0	0	l2_lockdown
cp15	1	c9	c0	2	l2_aux_control
cp15	0	c10	c0	0	tlb_lockdown
cp15	0	c10	c0	1	tlb_i_lockdown
cp15	0	c10	c1	0	d_tlb_preload
cp15	0	c10	c1	1	i_tlb_preload
cp15	0	c10	c2	0	prrr
cp15	0	c10	c2	1	nmrr
cp15	0	c11	c0	0	ple_st_present
cp15	0	c11	c0	2	ple_st_running
cp15	0	c11	c0	3	ple_st_interrupting
cp15	0	c11	c1	0	ple_user_access
cp15	0	c11	c2	0	ple_ch_number
cp15	0	c11	c3	0	ple_enable_stop
cp15	0	c11	c3	1	ple_enable_start
cp15	0	c11	c3	2	ple_enable_clear
cp15	0	c11	c4	0	ple_control
cp15	0	c11	c5	0	ple_istart
cp15	0	c11	c7	0	ple_iend
cp15	0	c11	c8	0	ple_status
cp15	0	c11	c15	0	ple_contextid
cp15	0	c12	c0	0	vbar
cp15	0	c12	c0	1	mvbar
cp15	0	c12	c1	0	isr
cp15	0	c13	c0	0	fcseidr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw
cp15	0	c15	c0	0	l1_data0
cp15	0	c15	c0	1	l1_data1
cp15	0	c15	c0	2	d_l1_cam_write
cp15	0	c15	c0	3	d_l1_tlb_attr_write
cp15	0	c15	c0	4	d_l1_tlb_pa_write
cp15	0	c15	c0	5	d_l1_hvab_write
cp15	0	c15	c0	6	d_l1_tag_write
cp15	0	c15	c0	7	d_l1_data_write
cp15	0	c15	c1	0	l1_inst0
cp15	0	c15	c1	1	l1_inst1
cp15	0	c15	c1	2	i_l1_cam_write
cp15	0	c15	c1	3	i_l1_tlb_attr_write
cp15	0	c15	c1	4	i_l1_tlb_pa_write
cp15	0	c15	c1	5	i_l1_hvab_write
cp15	0	c15	c1	6	i_l1_tag_write
cp15	0	c15	c1	7	i_l1_data_write
cp15	0	c15	c2	2	d_l1_cam_read
cp15	0	c15	c2	3	d_l1_tlb_attr_read
cp15	0	c15	c2	4	d_l1_tlb_pa_read
cp15	0	c15	c2	5	d_l1_hvab_read
cp15	0	c15	c2	6	d_l1_tag_read
cp15	0	c15	c2	7	d_l1_data_read
cp15	0	c15	c3	2	i_l1_cam_read
cp15	0	c15	c3	3	i_l1_tlb_attr_read
cp15	0	c15	c3	4	i_l1_tlb_pa_read
cp15	0	c15	c3	5	i_l1_hvab_read

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c15	c3	6	i_l1_tag_read
cp15	0	c15	c3	7	i_l1_data_read
cp15	0	c15	c5	2	i_l1_ghb_write
cp15	0	c15	c5	3	i_l1_btb_write
cp15	0	c15	c7	2	i_l1_ghb_read
cp15	0	c15	c7	3	i_l1_btb_read
cp15	0	c15	c8	0	l2_data0
cp15	0	c15	c8	1	l2_data1
cp15	0	c15	c8	2	l2_tag_write
cp15	0	c15	c8	4	l2_parity_ecc_write
cp15	0	c15	c8	5	l2_data2
cp15	0	c15	c9	2	l2_tag_read
cp15	0	c15	c9	4	l2_parity_ecc_read

Register Names For Cortex-A9

CP	OP1	CRn	CRm	OP2	Name
cp14	6	c0	c0	0	teecr
cp14	6	c1	c0	0	teehbr
cp14	7	c0	c0	0	jidr
cp14	7	c1	c0	0	joscr
cp14	7	c2	c0	0	jmcr
cp14	7	c3	c0	0	jpr
cp14	7	c4	c0	0	jcotr
cp15	0	c0	c0	0	midr
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	3	tlbtr
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c0	6	revidr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr1
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mmfr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr
cp15	1	c0	c0	7	aidr
cp15	2	c0	c0	0	csselr
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c1	c1	0	scr
cp15	0	c1	c1	1	sder
cp15	0	c1	c1	2	nsacr
cp15	0	c1	c1	3	cp15_vcr
cp15	0	c2	c0	0	ttbr0
cp15	0	c2	c0	1	ttbr1
cp15	0	c2	c0	2	ttbcr
cp15	0	c3	c0	0	dacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	0	c6	c0	0	dfar
cp15	0	c6	c0	2	ifar
cp15	0	c7	c0	4	nop
cp15	0	c7	c1	0	icialluis
cp15	0	c7	c1	6	bpiallis
cp15	0	c7	c4	0	par
cp15	0	c7	c5	0	iciallu
cp15	0	c7	c5	1	icimvau
cp15	0	c7	c5	4	cp15isb
cp15	0	c7	c5	6	bpiall
cp15	0	c7	c5	7	bpimva
cp15	0	c7	c6	1	dcimvac
cp15	0	c7	c6	2	dcisw
cp15	0	c7	c8	0	v2pcwpr
cp15	0	c7	c8	1	v2pcwpw
cp15	0	c7	c8	2	v2pcwur
cp15	0	c7	c8	3	v2pcwuw
cp15	0	c7	c8	4	v2powpr
cp15	0	c7	c8	5	v2powpw
cp15	0	c7	c8	6	v2powur
cp15	0	c7	c8	7	v2powuw
cp15	0	c7	c10	1	dccvac
cp15	0	c7	c10	2	dccsw
cp15	0	c7	c10	4	cp15dsb
cp15	0	c7	c10	5	cp15dmb
cp15	0	c7	c11	1	dccvau
cp15	0	c7	c14	1	dccimvac
cp15	0	c7	c14	2	dccisw

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c8	c3	0	tlbiallis
cp15	0	c8	c3	1	tlbimvais
cp15	0	c8	c3	2	tlbiasidis
cp15	0	c8	c3	3	tlbimvaais
cp15	0	c8	c5	0	tlbiall
cp15	0	c8	c5	1	tlbimva
cp15	0	c8	c5	2	tlbiasid
cp15	0	c8	c5	3	tlbimvaa
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	3	pmovsr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr
cp15	0	c9	c13	0	pmccntr
cp15	0	c9	c13	1	pmxevtyper
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	0	c10	c0	0	tlb_lockdown
cp15	0	c10	c2	0	prrr
cp15	0	c10	c2	1	nmrr
cp15	0	c11	c0	0	pleidr
cp15	0	c11	c0	2	pleasr
cp15	0	c11	c0	4	plefsr
cp15	0	c11	c1	0	pleuar
cp15	0	c11	c1	1	plepcr
cp15	0	c12	c0	0	vbar
cp15	0	c12	c0	1	mvbar

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c12	c1	0	isr
cp15	0	c12	c1	1	vir
cp15	0	c13	c0	0	fcseidr
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw
cp15	0	c15	c0	0	power_control
cp15	0	c15	c1	0	neon_busy
cp15	4	c15	c0	0	config_base_addr
cp15	5	c15	c4	2	read_main_tlb_entry
cp15	5	c15	c4	4	write_main_tlb_entry
cp15	5	c15	c5	2	main_tlb_va
cp15	5	c15	c6	2	main_tlb_pa
cp15	5	c15	c7	2	main_tlb_attr

Register Names For Cortex-A15

CP	OP1	CRn	CRm	OP2	Name
cp14	6	c0	c0	0	teecr
cp14	6	c1	c0	0	teehbr
cp14	7	c0	c0	0	jidr
cp14	7	c1	c0	0	joscr
cp14	7	c2	c0	0	jmcr
cp15	0	c0	c0	0	midr
cp15	0	c0	c0	1	ctr
cp15	0	c0	c0	2	tcmtr
cp15	0	c0	c0	3	tlbtr
cp15	0	c0	c0	5	mpidr
cp15	0	c0	c0	6	revidr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c0	c1	0	id_pfr0
cp15	0	c0	c1	1	id_pfr0
cp15	0	c0	c1	2	id_dfr0
cp15	0	c0	c1	3	id_afr0
cp15	0	c0	c1	4	id_mmfr0
cp15	0	c0	c1	5	id_mmfr1
cp15	0	c0	c1	6	id_mmfr2
cp15	0	c0	c1	7	id_mmfr3
cp15	0	c0	c2	0	id_isar0
cp15	0	c0	c2	1	id_isar1
cp15	0	c0	c2	2	id_isar2
cp15	0	c0	c2	3	id_isar3
cp15	0	c0	c2	4	id_isar4
cp15	1	c0	c0	0	ccsidr
cp15	1	c0	c0	1	clidr
cp15	1	c0	c0	7	aidr
cp15	2	c0	c0	0	csselr
cp15	4	c0	c0	0	vpidr
cp15	4	c0	c0	5	vmpidr
cp15	0	c1	c0	0	sctlr
cp15	0	c1	c0	1	actlr
cp15	0	c1	c0	2	cpacr
cp15	0	c1	c1	0	scr
cp15	0	c1	c1	1	sder
cp15	0	c1	c1	2	nsacr
cp15	4	c1	c0	0	hsctlr
cp15	4	c1	c0	1	hactlr
cp15	4	c1	c1	0	hcr
cp15	4	c1	c1	1	hdcr
cp15	4	c1	c1	2	hcptr

CP	OP1	CRn	CRm	OP2	Name
cp15	4	c1	c1	3	hstr
cp15	4	c1	c1	7	hacr
cp15	0	c2	c0	2	ttbcr
cp15	4	c2	c0	2	htcr
cp15	4	c2	c1	2	vtcr
cp15	0	c3	c0	0	dacr
cp15	0	c5	c0	0	dfsr
cp15	0	c5	c0	1	ifsr
cp15	0	c5	c1	0	adfsr
cp15	0	c5	c1	1	aifsr
cp15	4	c5	c1	0	hadfsr
cp15	4	c5	c1	1	haifsr
cp15	4	c5	c2	0	hsr
cp15	0	c6	c0	0	dfar
cp15	0	c6	c0	2	ifar
cp15	4	c6	c0	0	hdfar
cp15	4	c6	c0	2	hifar
cp15	4	c6	c0	4	hpfar
cp15	0	c7	c1	0	icialluis
cp15	0	c7	c1	6	bpiallis
cp15	0	c7	c5	0	iciallu
cp15	0	c7	c5	1	icimvau
cp15	0	c7	c5	4	cp15isb
cp15	0	c7	c5	6	bpiall
cp15	0	c7	c5	7	bpimva
cp15	0	c7	c6	1	dcimvac
cp15	0	c7	c6	2	dcisw
cp15	0	c7	c8	0	ats1cpr
cp15	0	c7	c8	1	ats1cpw
cp15	0	c7	c8	2	ats1cur

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c7	c8	3	ats1cuw
cp15	0	c7	c8	4	ats12nsopr
cp15	0	c7	c8	5	ats12nsopw
cp15	0	c7	c8	6	ats12nsour
cp15	0	c7	c8	7	ats12nsouw
cp15	0	c7	c10	1	dccmvac
cp15	0	c7	c10	2	dccsw
cp15	0	c7	c10	4	cp15dsb
cp15	0	c7	c10	5	cp15dmb
cp15	0	c7	c11	1	dccmvau
cp15	0	c7	c14	1	dccimvac
cp15	0	c7	c14	2	dccisw
cp15	4	c7	c8	0	ats1hr
cp15	4	c7	c8	1	ats1hw
cp15	0	c8	c3	0	tlbiallis
cp15	0	c8	c3	1	tlbimvais
cp15	0	c8	c3	2	tlbiasidis
cp15	0	c8	c3	3	tlbimvaais
cp15	0	c8	c3	4	tlbiallnsnhis
cp15	0	c8	c7	0	tlbiallh
cp15	0	c8	c7	1	tlbimvah
cp15	0	c8	c7	2	tlbiasid
cp15	0	c8	c7	3	tlbimvaa
cp15	0	c8	c7	4	tlbiallnsnh
cp15	0	c9	c12	0	pmcr
cp15	0	c9	c12	1	pmcntenset
cp15	0	c9	c12	2	pmcntenclr
cp15	0	c9	c12	3	pmovsr
cp15	0	c9	c12	4	pmswinc
cp15	0	c9	c12	5	pmselr

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c9	c13	0	pmccntr
cp15	0	c9	c13	1	pmxevtyper
cp15	0	c9	c13	2	pmxevcntr
cp15	0	c9	c14	0	pmuserenr
cp15	0	c9	c14	1	pmintenset
cp15	0	c9	c14	2	pmintenclr
cp15	0	c9	c14	3	pmovsset
cp15	1	c9	c0	2	l2ctlr
cp15	1	c9	c0	3	l2ectlr
cp15	0	c10	c2	0	prrr
cp15	0	c10	c2	1	nmrr
cp15	0	c10	c3	0	amair0
cp15	0	c10	c3	1	amair1
cp15	4	c10	c2	0	hmair0
cp15	4	c10	c2	1	hmair1
cp15	4	c10	c3	0	hamair0
cp15	4	c10	c3	1	hamair1
cp15	0	c12	c0	0	vbar
cp15	0	c12	c0	1	mvbar
cp15	0	c12	c1	0	isr
cp15	4	c12	c0	0	hvbar
cp15	0	c13	c0	0	fcseidr
cp15	0	c13	c0	1	contextidr
cp15	0	c13	c0	2	tpidrurw
cp15	0	c13	c0	3	tpidruro
cp15	0	c13	c0	4	tpidrprw
cp15	4	c13	c0	2	htpidr
cp15	0	c14	c0	0	cntfrq
cp15	0	c14	c1	0	cntkctl
cp15	0	c14	c2	0	cntp_tval

CP	OP1	CRn	CRm	OP2	Name
cp15	0	c14	c2	1	cntp_ctl
cp15	0	c14	c3	0	cntv_tval
cp15	0	c14	c3	1	cntv_ctl
cp15	4	c14	c1	0	cnthctl
cp15	4	c14	c2	0	cnthp_tval
cp15	4	c14	c2	1	cnthp_ctl
cp15	0	c15	c0	0	il1data0
cp15	0	c15	c0	1	il1data1
cp15	0	c15	c0	2	il1data2
cp15	0	c15	c1	0	dl1data0
cp15	0	c15	c1	1	dl1data1
cp15	0	c15	c1	2	dl1data2
cp15	0	c15	c1	3	dl1data3
cp15	0	c15	c4	0	ramindex
cp15	1	c15	c0	0	l2actlr
cp15	1	c15	c0	3	l2pfr
cp15	1	c15	c0	4	actlr2
cp15	4	c15	c0	0	cbar

Appendix H

Third-Party Licensing and Copyright Information

Contents

This appendix contains licensing and copyright information for third-party software used by the Green Hills Probe.

MD5 Message-Digest Algorithm License Agreement

Third-Party DES Copyright Information

Appendix I

Declaration of Conformity

Declaration of Conformity

Manufacturer:	Green Hills Software 30 West Sola Street Santa Barbara, CA 93101 USA

Declares that the following product:

Product Description:	Trace Port Analyzer (Serial)
Model:	SuperTrace Probe, 520-ST401 (serial numbers greater than 25500)

Is in conformity with the EMC and EMI requirements set forth in:

- European Standards EN 61326-1:2006
- FCC Title 47, Part 15, Subpart B, per CISPR 22: 1997 Class A Limits (Test procedure ANSI C63.4: 2003)

Additional Information:	This product was tested in a typical configuration.
Date:	December 7, 2012 - December 14, 2012

Declaration of Conformity

Manufacturer:	Green Hills Software
	30 West Sola Street
	Santa Barbara, CA 93101 USA

Declares that the following product:

Product Description:	Trace Port Analyzer (Parallel)
Model:	SuperTrace Probe, 520-ST400 (serial numbers greater than 25500)

Is in conformity with the EMC and EMI requirements set forth in:

- European Standards EN 61326-1:2006
- FCC Title 47, Part 15, Subpart B, per CISPR 22: 1997 Class A Limits (Test procedure ANSI C63.4: 2003)

Additional Information:	This product was tested in a typical configuration.
Date:	August 4, 2010 - August 17, 2010

Declaration of Conformity

Manufacturer:	Green Hills Software 30 West Sola Street Santa Barbara, CA 93101 USA

Declares that the following product:

Product Description:	Debug Emulator
Model:	Green Hills Probe (serial numbers greater than 10000)

Is in conformity with the EMC and EMI requirements set forth in:

- European Standards EN 61326: 2002, which references the following specifications:
  - CISPR 11: 1990
  - EN 55011: 1991
  - EN 61000-3-2+A14: 2000
  - EN61000-3-3: 1994
  - EN 61000-4-2: 1995+A1: 1998
  - EN 61000-4-3: 1996+A1: 1998
  - EN 61000-4-4: 1995
  - EN 61000-4-5: 1995
  - EN 61000-4-6: 1996
  - EN 61000-4-11: 1994
- FCC Title 47, Part 15, Subpart B, per CISPR 22: 1997 Class A Limits (Test procedure ANSI C63.4: 2003)

Additional Information:	This product was tested in a typical configuration.
Date:	March 16, 2006 and March 17, 2006

Declaration of Conformity

Manufacturer:	Green Hills Software 30 West Sola Street Santa Barbara, CA 93101 USA

Declares that the following product:

Product Description:	JTAG Probe
Model:	SuperTrace Probe

Is in conformity with the EMC and EMI requirements set forth in:

- European Standards EN 61326: 1997 +A1: 1998, which references the following specifications:
  - EN 55011: 1991
  - EN 61000-3-2: 1995
  - EN 61000-3-2 +A12: 1996
  - EN 61000-4-2: 1995
  - EN 61000-4-3: 1996
  - EN 61000-4-4: 1995
  - EN 61000-4-5: 1995
  - EN 61000-4-6: 1996
  - EN 61000-4-11: 1994
- European Standards EN 61000-3-2 +A14: 2000 (Class A)
- European Standards EN 61000-3-3 +A14: 2000 (Class A)

Additional Information:	This product was tested in a typical configuration.
Date:	September 29, 2003

Declaration of Conformity

Manufacturer:	Green Hills Software 30 West Sola Street Santa Barbara, CA 93101 USA

Declares that the following product:

Product Description:	Debug Emulator
Model:	Green Hills Probe (serial numbers greater than 1799)

Is in conformity with the Class B EMC and EMI requirements set forth in:

- European Standards EN 61326: 1997 +A1: 1998, which references the following specifications:
  - EN 61000-4-2: 1995
  - EN 61000-4-3: 1995
  - EN 61000-4-4: 1995
  - EN 61000-4-5: 1995
  - EN 61000-4-6: 1996
  - EN 61000-4-11: 1994
- European Standards EN 61000-3-3: 1995
- European Standards EN 61000-3-2: 2000
- FCC Title 47, Part 15, Subpart B, per CISPR 22: 1997 Limits (Test procedure ANSI C63.4: 1992)

Additional Information:	This product was tested in a typical configuration.
Date:	September 3, 2002 and September 13, 2002

Declaration of Conformity

Manufacturer:	Green Hills Software
	30 West Sola Street
	Santa Barbara, CA 93101 USA

Declares that the following product:

Product Description:	Debug Emulator
Model:	Green Hills Probe (serial numbers less than 1700)

Is in conformity with the Class B EMC and EMI requirements set forth in:

- European Standards EN 55022: 1998
- European Standards EN 61000-3-2: 1995
- European Standards EN 61000-3-3: 1995
- European Standards EN 55024: 1998, which references the following specifications:
  - EN 61000-4-2: 1995
  - EN 61000-4-3: 1995
  - EN 61000-4-4: 1995
  - EN 61000-4-5: 1995
  - EN 61000-4-6: 1996
  - EN 61000-4-8: 1993
  - EN 61000-4-11: 1994
- FCC Title 47, Part 15, Subpart B, per CISPR 22: 1997 Limits (Test procedure ANSI C63.4: 1992)

Additional Information:	This product was tested in a typical configuration.
Date:	May 14, 2001 and July 2, 2001

Index