MULTI: Configuring Connections for ARM64 Targets

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

LEGAL NOTICES AND DISCLAIMERS

PubID: connect_arm64-744926
Branch: http://toolsvc/branches/release-branch-2023-5-comp
Date: September 1, 2023

Contents

Preface

Contents

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

About This Book

The MULTI Debugger can be used to debug applications that reside on native, embedded, or simulated *targets*. To debug these applications, MULTI must first connect to the target through a *debug server* that runs on the host. Each distribution of the MULTI Integrated Development Environment (IDE) includes multiple debug servers, each of which supports connections to a specific set of probes, in-circuit emulators, monitors, and/or target simulators. This book describes how to configure connections to your target using the appropriate debug server. Specifically:

- The chapters in this book provide connecting instructions, features, commands, and options specific to each debug server. This information supplements the general connection instructions described in the documentation about connecting to your target 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.

The MULTI Document Set

The primary documentation for using MULTI is provided in the following books:

- *MULTI: Getting Started* — Provides an introduction to the MULTI Integrated Development Environment and leads you through a simple tutorial.
- *MULTI: Licensing* — Describes how to obtain, install, and administer MULTI licenses.
- *MULTI: Managing Projects and Configuring the IDE* — Describes how to create and manage projects and how to configure the MULTI IDE.

- *MULTI: Building Applications* — Describes how to use the compiler driver and the tools that compile, assemble, and link your code. Also describes the Green Hills implementation of supported high-level languages.
- *MULTI: Configuring Connections* — Describes how to configure connections to your target.
- *MULTI: Debugging* — Describes how to set up your target debugging interface for use with MULTI and how to use the MULTI Debugger and associated tools.
- *MULTI: Debugging Command Reference* — Describes how to use Debugger commands and provides a comprehensive reference of Debugger commands.
- *MULTI: Scripting* — Describes how to create MULTI scripts. Also contains information about the MULTI-Python integration.
- *MULTI: Toolchain Error Messages and Other Diagnostics* — Provides a sequential listing of MULTI diagnostic numbers and messages for the compiler, assembler, and linker. Where possible, includes information about associated resolutions and workarounds.

For a comprehensive list of the books provided with your MULTI installation, see the **Help → Manuals** menu accessible from most MULTI windows.

All books are available in one or more of the following formats:

- Print
- Online help, accessible from most MULTI windows via the **Help → Manuals** menu
- PDF, available in the **manuals** subdirectory of your MULTI or compiler installation

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).

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	The **setup** command
	Option	The **-G** option
	Window title	The **Breakpoints** window
	Menu name or menu choice	The **File** menu
	Field name	The **Name** field
	Button name	The **Browse** button
italic type	Replaceable text	where *version* is replaced by the version number
	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 `unload`
	A word or words used in a command or example	`Path` must be a valid C string
	Source code	`printf("Hello world!\n");`
	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** *func* | *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 indicates that:

- **gxyz** is a command.
- 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.

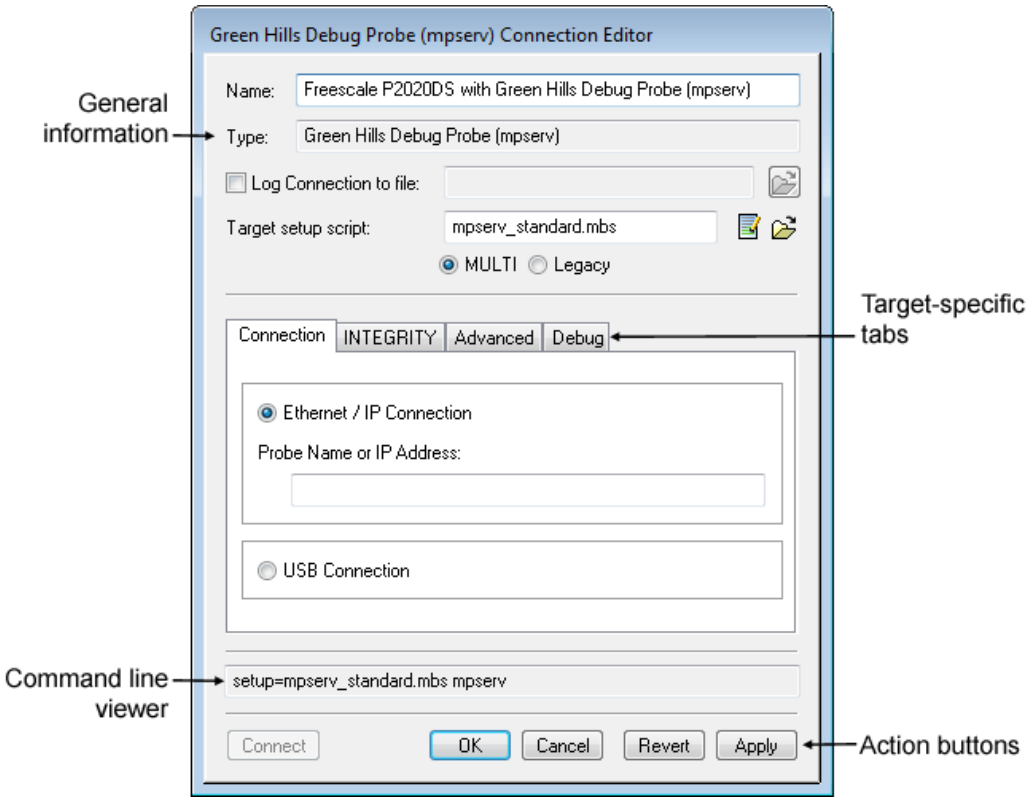
The square brackets and the ellipsis should not appear in the actual command you enter.

Chapter 1

Using the Connection Editor

Contents

The **Connection Editor** is a target specific dialog that you can use to configure and edit Connection Methods.



The **Connection Editor** window for Standard Connection Methods has four sections, which are called out in the above graphic and documented in the following sections.

General Information Settings

The first section of the **Connection Editor** contains fields that display and/or set basic properties of a Connection Method. These fields are standard and appear on nearly all **Connection Editor** windows. The following table describes each field.

Name Displays the name of the Standard Connection Method you define or edit. You can change the name of your method by editing this field.
Type Displays the type of debugging interface the Connection Method is for, which determines which debug server MULTI uses. The connection type is specified when you create a Standard Connection Method. You cannot edit this read-only field.
Log Connection to file Enables you to log transactions between MULTI and your debug server. To specify the file to write the log to, enter a pathname or click to browse for the desired file. On Linux, if you check the option box but do not specify a filename, MULTI writes the log to standard error output. Logging is disabled by default.
Target Setup script Specifies your target setup script file. This is an optional field because not all targets require setup scripts. If you require a setup script to configure your board, specify the filename (and full path, if necessary) of the appropriate script or click to browse for it. When this Connection Method is used, MULTI usually runs the specified script before each download. If you are editing a default Connection Method created by the Project Wizard for your processor-board combination, the necessary setup script file (if applicable) appears in this field automatically. To open the Editor on the target setup script, click . If no setup script is specified in this field, the Editor opens on a file with a filename derived from the debug connection's name. For more information about target setup scripts, see the documentation about configuring your target hardware in the *MULTI: Debugging* book.
Scripting language radio buttons Specify the scripting language so that MULTI knows how to interpret the specified target setup script, where: • **MULTI** — Indicates that the specified target setup script was written with the MULTI scripting language. This is the default setting. For more information about MULTI scripting in general, see the documentation about MULTI scripts in the *MULTI: Scripting* book. For more detailed information about editing and using setup scripts, see the documentation about configuring your target hardware in the *MULTI: Debugging* book.

Target-Specific Tabs

The second section of the **Connection Editor** contains one or more tabs that contain fields for setting options specific to your debugging interface, type of connection, and target architecture. The number of tabs and the fields that appear vary according to the type of Standard Connection Method you are editing.

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.

The following table describes in general terms the types of settings and options that appear on the most common tabs.

Tab	Description
Connection tab	The options on this tab usually require input for the Connection Method to function properly. The settings you choose for the connection options are probably different for each new target you connect to.
INTEGRITY tab	The option on this tab is useful if you are debugging an INTEGRITY system. For more information, see the documentation for **set_runmode_partner** in *MULTI: Debugging Command Reference*.
Download tab	The options on this tab allow you to customize the process of downloading programs to the target. For example, this tab allows you to control what program sections are downloaded to the target.
Advanced tab	The options on this tab allow a high degree of control over the details of your connection. These options can often be left in their default states. Use this tab sparingly because changing the advanced options from their default settings may cause problems with your connection.
Debug tab	The options on this tab are provided to help troubleshoot your connection. You should not change these settings unless instructed to do so by Green Hills Technical Support.

Most **Connection Editors** include at least a **Connection** tab and a **Debug** tab, but many contain additional tabs. For most connections, you need to set or edit some of the fields on these tabs for your Standard Connection Method to work properly. For more information about the **Connection Editor** settings available for your particular target, see the chapter that documents your debug server.

The Command Line Viewer

The third section of the **Connection Editor** displays the command line equivalents of your GUI selections. Default selections do not always have corresponding commands or options on the command line.

The command line viewer is a read-only field. You cannot make changes to the command line by typing in this field.

Changes you make in the GUI will not be reflected in the command line viewer until you click **Apply** or **OK**.

Action Buttons

The fourth section of the **Connection Editor** contains buttons that allow you to discard your edits, apply your edits, or connect using a Connection Method. The following table describes each button.

Button	Description
Connect	Records your changes, connects to your target using the Standard Connection Method you are editing, and closes the **Connection Editor**.
OK	Records your changes and closes the **Connection Editor**.
Cancel	Discards your changes and closes the **Connection Editor**.
Revert	Discards your changes and leaves the **Connection Editor** open.
Apply	Records your changes and leaves the **Connection Editor** open.

Note

Clicking **Connect**, **OK**, or **Apply** in the **Connection Editor** saves your Standard Connection Method. In the future, the Connection Method will appear in the list of available Connection Methods in the **Connection Chooser** and the **Connection Organizer**.

Chapter 2

Green Hills Debug Probe (mpserv) Connections

The Green Hills family of debug probes includes the Green Hills Probe and the SuperTrace Probe. These advanced hardware debugging instruments communicate with your target over a standard JTAG or SWD test port and allow you to use the MULTI Debugger to control, debug, and trace your target CPU. Green Hills Probe and SuperTrace Probe target connections are all supported through the **mpserv** debug server. For complete instructions on how to set up these probes and make target connections using **mpserv**, see the *Green Hills Debug Probes User's Guide* that came with your probe.

Chapter 3

Simulator for ARM64 (simarm64) Connections

Contents

MULTI includes a target simulator called **simarm64** that allows you to execute and debug ARM64 code on your host system. This chapter supplements the documentation about connecting to your target in the *MULTI: Debugging* book with information about how to connect to and use the **simarm64** Simulator.

Installing the Simulator for ARM64 (simarm64)

The **simarm64** Simulator is installed automatically when you install the MULTI Compiler. No other installation steps are necessary.

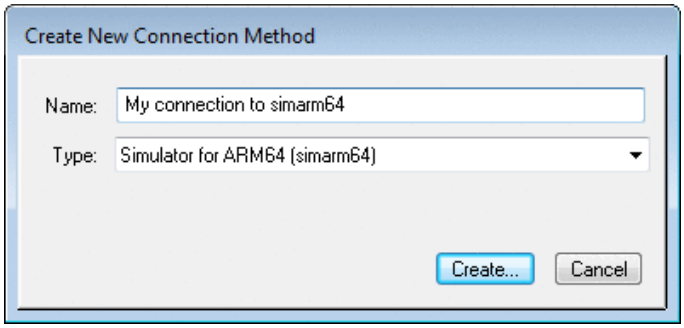
Connecting to the Simulator for ARM64 (simarm64)

To help you connect to targets (including simulators) quickly and easily, MULTI allows you to create and save Connection Methods that specify the parameters of your connection.

For general instructions that explain how to create and use Connection Methods, see the documentation about connecting to your target in the *MULTI: Debugging* book. The information in the following sections supplements the instructions provided there with information that is specific to the ARM64 Simulator.

Creating a Standard Connection Method for the ARM64 Simulator

When creating a new Connection Method for the ARM64 Simulator, select **Simulator for ARM64 (simarm64)** as the connection type in the **Create New Connection Method** dialog box.



For detailed information about creating new Standard Connection Methods, see the documentation about Standard Connection Methods in the *MULTI: Debugging* book.

The Simulator for ARM64 (simarm64) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods (see Chapter 1, "Using the Connection Editor" on page 1), the **Simulator for ARM64 (simarm64) Connection Editor** includes a **Debug** tab.

The field on the **Debug** tab of the **Simulator for ARM64 (simarm64) Connection Editor** is described in the following section.

Simulator for ARM64 (simarm64) 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 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 command for **simarm64** Custom Connection Methods.

To create a Custom Connection Method for the ARM64 Simulator, type the command given below into the **Start a Custom Connection** dialog box and click **Connect**.

simarm64 [*options*]...

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 you entered. You can change this name by using the **Custom Connection Editor**. See the documentation about Custom Connection Methods in the *MULTI: Debugging* book for more information.

Note

You can also enter the above connection command in the Debugger's command pane, where it must be preceded by the **connect** command.

Options for Custom simarm64 Simulator Connection Methods

The following options are available when creating a Custom **simarm64** simulator Connection Method.

-allocmem Allocates memory pages at run-time when simulating access to addresses outside the known extent of ROM or RAM. When **-allocmem** is not enabled, accesses to addresses outside the known extent of memory result in a `Memory Access Violation` or `Invalid Memory Read` error. See the release notes in the MULTI installation directory for more information.
-collect_stats Enables collection of various run-time statistics (fast mode only).
-cpu=*cpu* Specifies which target processor to simulate.
-rom Enables ROM mode on the simulator and sets the simulator to start from the reset vector. For more information, see “ROM Mode” on page 14.
-rom_use_entry Enables ROM mode on the simulator and sets the simulator to start from the program's entry point. For more information, see “ROM Mode” on page 14.

Simulation Environments

The **simarm64** simulator has two simulation environments:

- *OS Simulation Mode* — Simulates an environment similar to a user process running in Linux. This is the default mode.
- *ROM Mode* — Simulates just the bare minimum hardware, such as the CPU and memory systems. To enable ROM mode, connect using a Custom Connection Method and include the **-rom** or **-rom_use_entry** option in the Connection Method command.

The **simarm64** Simulator is designed to be applicable for user level applications, but processor features such as caches, memory management unit, and any special peripherals and registers are generally not implemented.

OS Simulation Mode

The OS Simulator Mode (which is the default mode of the **simarm64** simulator) provides an environment similar to a user process running under Linux. The simulator allocates memory for the text and initialized and uninitialized data of the simulated program, loads it into memory, and starts executing at the entry point specified at link time. The stack pointer is initialized and any program arguments are passed to the main program via the normal `argc`/`argv` convention. A limited set of system calls is supported for I/O purposes. This level of simulation is suitable for debugging user-level programs, but if your programs need a more accurate simulation of the actual CPU behavior, you should run in ROM mode (see “ROM Mode” on page 14).

While running in the simulator, your program can make the system calls listed in the table below.

`access`	`alarm`	`brk`	`close`	`creat`	`exit`
`fstat`	`getpid`	`link`	`lseek`	`open`	`read`
`rename`	`stat`	`time`	`unlink`	`write`	

Be aware that while only these system calls are implemented, many library functions use a combination of these calls to achieve their goal.

In OS simulation mode, the only memory available is that allocated initially, plus any created by using the `brk` system call, such as using the `malloc` function in the libraries. References to other memory addresses will be trapped by the simulator and cause the program to abort. Similarly, attempts to use exception generating instructions also cause the program to abort. You should use ROM mode if the simulation must closely reflect actual hardware behavior.

ROM Mode

ROM Mode simulates only the bare minimum hardware, such as CPU and memory systems. In ROM mode, you can set the simulator to start at either the reset vector or at the program's entry point. To cause the simulator to always start execution at the address of the system reset vector , as if the CPU were given a hard reset, and to ignore any program startup address specified at link time, connect using a Custom Connection Method and include the **-rom** option in the Connection Method command. To start at the program's entry point instead, use a Custom Connection Method and include the **-rom_use_entry** option in the Connection Method command..

Other features of ROM mode include:

- Exceptions are handled as in the hardware, vectoring to the appropriate address. User code must be supplied to handle the exception appropriately.
- No arguments are passed to the running program.
- The stack pointer must be initialized by user startup code.
- A limited set of system calls for I/O is supported.

Since the simulator handles very few things for you in ROM mode, you can use this mode for writing and testing exception handlers, and even parts of a real operating system. You should write at least some start-up code in pure assembly language, since this mode simulates a bare bones processor.

Note

Memory pages are not allocated at run-time when simulating accesses to memory outside the known extent of RAM. If you encounter `Invalid memory read/write` errors, you can:

- Add the extra memory pages you need as a section in the link map (**.ld** file) for the program you are simulating and re-link your program.
- Pass the **-allocmem** option to the simulator (for example, `simarm64 -rom_use_entry -allocmem`).

Profiling with simarm64

You can use MULTI to profile with the **simarm64** simulator. The simulated clock rate is 20 MHz, but the clock cycles used by each instruction are approximations. For more information about profiling, see the documentation about recording and viewing profiling data in the *MULTI: Debugging* book.

Connecting to the Simulator for ARM64 as a Stand-alone Debug Server

To connect to the Simulator for ARM64 from the command line, independently of MULTI, enter:

`simarm64 [`*`options`*`] `*`image_file`*

where:

- *`options`* — Can be any non-conflicting combination of the options listed in "Options for Custom simarm64 Simulator Connection Methods" on page 12.
- *`image_file`* — Specifies an ELF-style executable file.

Connecting to the simulator in this way allows you to simulate your program, but does not offer the interactive debugging capabilities available through MULTI.

Chapter 4

Simulator for INTEGRITY on ARM64 (isimarm64)

The **isimarm64** simulator is a specialized version of the simulator for ARM64 intended for simulating a target running the INTEGRITY Real-Time Operating System. For more information, see the *INTEGRITY Development Guide*.

Index

T