

A P P N O T E SSM

Using a MAJIC® Probe with Embedded Linux Software

Revision: 1.1

Last Modified: October 25, 2006

©Copyright Mentor Graphics Corporation 2006. All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

Trademarks that appear in Mentor Graphics product publications that are not owned by Mentor Graphics are trademarks of their respective owners.

Introduction

A MAJIC® Intelligent Debug Probe performs debug services by using the target processor's JTAG interface to access registers within its Debug Support Unit (DSU). Using the MAJIC probe to debug your Embedded Linux kernel eliminates the need to rely on a debug agent running on the target. This allows you to debug the kernel at the very early stages of development and have full control of the processor. This application note explains the steps necessary to configure and use Embedded Linux software tools with a MAJIC probe. It will provide specific information on the following:

- Connecting a MAJIC probe using GDB to debug an Embedded Linux Kernel,
- Setting up the debug environment for simultaneous kernel-level debug-level, and application-level debug,
- Setting up the debug environment for loadable kernel module debug.

NOTE:

- This is not intended to be an Embedded Linux tutorial and in many places we simply describe what is needed to be done because the details are specific to your Embedded Linux environment. This is also not a tutorial on GDB itself; it explains how to connect GDB to the MAJIC probe, but assumes that you are familiar with the GDB debug environment. Please contact your Embedded Linux technical documentation and support sites for additional assistance.

- It is presumed that you already have the EDT and Embedded Linux software installed, have a Linux board support package installed for your target board, and have access to the documentation for the cross development tools and both the host and target configuration. The EDT package is normally distributed on a CD included with the MAJIC probe. Before installing the EDT package, it's a good idea to check the Mentor Graphics Web site to make sure you have the current EDT release.

Additional Documentation

Additional documentation for the EDT software package and MAJIC probes is installed as part of the EDT software package. On a Windows PC, use the EDT Launchpad in the Windows system tray to open the EDT Documentation Index. On a Linux PC, open `./manuals/edtX_doc_index.html` in your EDT software installation. The following documents will prove particularly useful through the course of this application note:

<i>MAJIC Probe User's Manual</i>	Complete information on configuration and operation of the MAJIC probe, and the MON command language.
<i>Using a MAJIC Probe With A Linux PC</i>	Application note that outlines installation of EDT software, and using the MAJIC probe on a Linux hosted environment.
<i>Using a MAJIC Probe In A Cygwin Environment</i>	Application note that outlines installation of EDT software, and using the MAJIC probe in the Cygwin environment.
<i>MDI for MAJIC User's Manual</i>	Details on the MDI interface to the MAJIC probe, and the MDI configuration file.
<i>gdb-readme.txt</i>	Additional technical details on setting up and using GDB and MDI-Server with a MAJIC probe.
<i>Your Embedded Linux Documentation</i>	Information on configuration and installation of your Embedded Linux.

Getting Support

For additional assistance configuring the MAJIC probe for use with an Embedded Linux target environment, please visit <http://www.mentor.com/supportnet/>.

The Debug Environment

Chapter 3 of the *MAJIC Probe User's Manual* explains the configuration process and the files that are involved in detail. The following section outlines the basic steps necessary to configure an Embedded Linux kernel for use with the MAJIC probe, and the GDB debug environment.

MAJIC Configuration Files

Every target system is different, so the MAJIC probe needs information about the target system design in order to operate correctly. The primary file, called *startice.cmd*, configures the JTAG interface and certain capabilities of the MAJIC probe. In some cases *startice.cmd* will call in a board initialization file to initialize the target hardware, and declare the memory map of the target board.

As part of your EDT software installation, we provide you with `./targets/*` directories that contain validated configuration files for many common reference platforms. These can be used as is, if you have one of these platforms, and they also serve as an example when creating custom initialization files for your own target hardware. The `./targets/_template` directory provides a template for creating a board initialization file set for a custom hardware design, plus instructions for filling in the template.

Configuring your Embedded Linux Kernel

To use the MAJIC probe for kernel debugging, you will need to configure your Embedded Linux kernel to disable *KGDB* and turn on debug information.

1. Most Linux configurations provide a configuration tool which allows you to customize your Linux kernel. The details may depend upon your Linux provider, so consult your Linux documentation for details. We suggest you try running the following command in your Linux workspace directory:

```
$ make xconfig or $ make menuconfig
```

2. Select the **Kernel hacking** menu.
3. Enable the **Include debugging information in kernel binary** option. Then, enable the **Kernel debugging** option and any of its sub-options in which you are interested. Finally, make sure to disable the **KGDB support** option.
4. Build your kernel following the instructions outlined in your Linux documentation. Make sure to run the following command in the Linux shell before executing your build:

```
make clean
```

5. Please refer to your Linux documentation for further host and target board configuration.

Starting MDI-Server

GDB communicates to the MAJIC probe through a separate program named MDI-Server. GDB sends “remote protocol” debug requests to MDI-Server which then sends corresponding debug requests to the MAJIC probe. Therefore, the MDI-Server program must be running before GDB can connect. With this connection method you don't need to build a MAJIC-specific GDB, but you can use an off-the-shelf GDB (or any of the GUI front-ends available for GDB) and connect to MDI-Server with the "target remote" command.

MDI-Server, by way of the MDI shared library, is responsible for managing the details of the MAJIC probe configuration. MDI-Server reads in JTAG settings and target board configuration from the configuration files. CPU information and communication parameters are set via the *epimdi.cfg* configuration file instead of using command line switches. Please refer to the *Using a MAJIC Probe With A Linux PC* application note, or *Using a MAJIC Probe In A Cygwin Environment* application note for details on creating *epimdi.cfg*.

Once you have your MAJIC probe configuration files and *epimdi.cfg* file prepared, you are ready to start MDI-Server. You should always start MDI-Server from the directory containing the *epimdi.cfg* file. Open a Linux shell and execute the following minimum command for launching MDI-Server:

```
$ ./mdi-server -l ./mdi.so
```

When MDI-Server is running it simply waits for an instance of GDB to connect on port 2345. In some cases, additional start-up options beyond the minimum invocation line above may be desired or even required. Please refer to the *Using a MAJIC Probe With A Linux PC* application note, or *Using a MAJIC Probe In A Cygwin Environment* application note for more details.

Loading an Embedded Linux Kernel Image

For those unfamiliar with **minicom**, it is a serial communication application analogous to Windows' HyperTerminal. It provides a means to interact with the target board's boot loader and the Embedded Linux kernel. Open a second Linux shell, become super user, and execute the following command:

```
# minicom
```

The **minicom** application will need to be configured correctly to successfully connect to your target board. Please refer to the Embedded Linux documentation for setup instructions. You should also look for documentation on how to load the Embedded Linux kernel. However, before you load the kernel, you should start an instance of GDB.

Starting GDB

Before starting GDB, you must start the MDI-Server program as described earlier in this application note so that it will be ready to complete the interface between GDB and the MAJIC® probe. Once MDI-Server is ready, open a third Linux shell and start the GDB that is included in your target board's Embedded Linux Board Support Package (BSP). For example:

```
$ /<path to your cross tools gdb debugger>/xscale_be-gdb
```

From the (*gdb*) prompt enter the following commands to load your file, connect to the target, and begin to debug. Note that these commands can be placed in a GDB startup script so that they are run automatically at execution.

```
(gdb) set heur 0 /* MIPS only—see the following note */
(gdb) set remoteti 10 /* Remote timeout—see the following note */
(gdb) file /<path to linux image>/vmlinux_file /* Read in debug information */
(gdb) target remote localhost:2345 /* Open MDI connection to MAJIC */
```

NOTE:

- GDB has an internal variable named *heuristic-fence-post*. It controls how far back from the current PC GDB will scan memory looking for a function start. This variable should default to 0, but it may not in all GDB builds. If it is not 0, GDB may try to access invalid memory locations below the PC. This is primarily a concern for MIPS targets, where the initial PC value when connecting is usually 0xbfc00000 (the reset vector), and addresses in the 0xbfbxxxxx range are invalid.
- When GDB first connects, it can take a few seconds for MDI-Server to open the MDI connection to the MAJIC probe and process the *startice.cmd* start-up command file. Setting the internal GDB variable named *remotetimeout* prevents most GDB builds from timing out prematurely.
- The *vmlinux_file* is the binary that contains the kernel image along with symbolic debug information.

At this point, you should see a successful MDI connection in the MDI-Server Linux shell. The output should look similar to the following:

```
Accepted gdb MDI connection.
Capabilities:
    TraceOutput
    TraceCtrl
End of Capabilities List
Devices:
Index  Name
[0]    my_ixp425 via my_205.158.243.200
Selecting device [0] of 1
Notification from the target:
Target power detected on VREF
Auto JTAG detection process detected 1 TAP
JTAG connection established

JTAG ID Register: 19274013
Done.
```

Because the MAJIC® probe has reset and stopped the processor upon JTAG connection, you need to run the following command.

```
(gdb) continue
```

Now in the **minicom** Linux shell, you should see a boot loader prompt. You are now ready to download your Embedded Linux kernel onto your target board. Because the command(s) to do this differs from target to target, you should refer to the Embedded Linux documentation for your environment.

NOTE: Depending on the reset management of the target design, sometimes an explicit reset is necessary before the GDB **continue** command can be issued. To reset the target board, execute the following **MON** command. Please refer to the [MON Commands](#) section for more information.

```
(gdb) mon rt /* MON command to reset target */
```

Once the kernel uncompresses, you can stop GDB (control-C) to set breakpoints and run, or single step through your kernel.

MON Commands

The GDB **monitor** command can be used to pass **MON** commands through to the MDILib, as shown in the following examples. This allows access to the MAJIC probe features that are not available in GDB, such as trace display, and access to all CPU registers (and user-defined registers) rather than the limited set that GDB knows about. Appendix C of the *MAJIC Probe User's Manual* provides a table showing which MON commands are supported by MDILib, and Chapter 5 documents the MON command language in detail.

```
(gdb) mon di                /* Pass di command to MDILib and display response */
(gdb) mon h                  /* Pass h command to MDILib to display help on MON commands */
(gdb) mon h h                /* Display help on using MON help */
(gdb) mon h d                /* Display help on MON Display commands */
(gdb) mon h dw               /* Display help on MON Display Word command */
(gdb) mon fr c cmdfile      /* Read and execute a MON command script file */
```

Application Level Debug

Using a MAJIC probe along with your Embedded Linux software provides more debug capability by giving you a way to have simultaneous kernel-level debug and application-level debug and not just application-level debug alone. The details of how to make your application available to load and run on your Embedded Linux target is outside of the scope of this document. But once you have that set up you can follow the instructions below to debug it using GDB or any debugger supporting the GDB remote protocol.

1. Boot the Linux kernel.
2. Launch a second instance of GDB from your Linux Package installation. Make sure that you launch GDB from your `<path to application>` directory.
3. Start gdbserver in the shell that's running **minicom**:

```
# gdbserver host:socket_number your_application_executable
```
4. In the second instance of GDB, type the following commands:

```
(gdb) target remote IP_of_target:socket_number
(gdb) symbol-file your_application_executable
```

Now, you are ready to debug your application in the second GDB session. When your application hits a breakpoint set in the second GDB session that process stops, but the rest of the system continues to run normally. If the execution of your application causes a kernel breakpoint to be hit (i.e. a breakpoint set in the first GDB session), then the CPU is stopped. This gives you the advantage of simultaneous kernel-level debug and application-level debug in the two instances of GDB.

Note that when a kernel breakpoint is hit in the GDB/MDI/MAJIC environment, the CPU is stopped, so the GDB/gdbserver environment is unusable until execution later resumes.

Loadable Kernel Module Debug

Another capability you have with the MAJIC® debug probe and Embedded Linux software, is the ability to debug loadable kernel modules (LKM). The following section describes the necessary debug configuration.

Three things are required to debug the kernel module. First, the loadable kernel module must be built with debug information. Second, the module needs to be saved in the mounted root file system used by the Embedded Linux target. And third, you will need to know where in the target system's memory the kernel will save the module's text area.

The following is an example command to launch/install a new LKM in a running Embedded Linux kernel. This must be run from a shell running on the target (e.g. your **minicom** Linux shell).

```
# insmod -m hello-1.o > map
```

The details of the address resolution are saved in the file map. You can then **grep** for the code location as shown in this example command:

```
# grep .text map
.text          00000050 c68c0060 2**2
c68c0060 T __insmod_hello-1_S.text_L80
c68c0060 t .text
```

NOTE: Be aware that you first have to release control of the processor by executing **continue** in the first GDB session (the one that is connected to the MAJIC probe).

In the first GDB session (that is connected to the MAJIC probe), read in the debug symbols of the kernel module as shown in the following example:

```
(gdb) add-symbol-file /<module_path>/hello.o 0xc68c0060
add symbol table from file /<module_path>/hello.o at .text_addr = 0xc68c0060
(y or n) y
Reading symbols from ../hello.o...done.
```

The GDB **add-symbol-file** command needs a memory address as a parameter, make sure to use the *.text* location from the map file. At this point, you can set breakpoints and run, or single-step through your loadable kernel module.