

A P P N O T E S SM

Using the EPIFlash Utility

Revision: 4.2

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

The EPI Development Tools (EDT) software package includes a utility for programming the flash memory on your target system. The EPI Flash programming utility (EPIFlash) is an application which runs on the target system and allows the user to erase and program the target flash memory device(s) through the JTAG port. This application note explains the steps necessary to set up and use the EPIFlash programming utility.

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.

MAJIC Probe User's Manual

Complete information on configuration and operation of the MAJIC probe, and the MON command language.

EDB User's Manual

Details on using the EPI Source Level Debugger named EDB.

epiflash_parts.txt

List of the flash part types supported by EPIFlash.

Getting Support

For additional assistance configuring EPIFlash or the MAJIC® probe for your system, please visit <http://www.mentor.com/supportnet>.

EPIFlash Overview

The EPIFlash utility is a program that is downloaded to and run on the target board. It uses the Semi-Hosting feature to do the following:

- Display flash configuration settings such as flash device type and base address,
- Provide menus to control the flash settings and perform flash functions such as erase and program,
- Read the flash image (binary) file over JTAG when programming or verifying the image,
- Write the flash image to a file over JTAG when backing up the flash image to disk.

NOTE: Since EPIFlash runs from RAM on the target board, it must be possible to download it into RAM. Usually this means the memory controller must be initialized prior to running EPIFlash. If your board has good working boot code, then it will perform this initialization. However, if there is no boot code (or bad boot code) then you may need to use the MAJIC to initialize the memory controller. This is accomplished by running a board initialization script.

NOTE: Before erasing the boot code on your board, make sure you have a suitable memory controller initialization script or it may be very difficult to get back to the point where you can install new boot code.

Getting Started

Since the EPIFlash utility runs on the target board, it is downloaded and run via a debugger. This section explains how to set up the environment for running EPIFlash. There are several options for running EPIFlash:

- If you are already using EDB then you will probably prefer to use EDB for running EPIFlash as well. However, the requirements noted above may mean that you need different start-up files and, therefore, a separate EDB shortcut for running EPIFlash, depending on how you normally configure EDB and MAJIC for debugging.
- You can run EPIFlash with third party RDI or MDI compliant debuggers that support semi-hosting calls for screen, keyboard, and file I/O.
- Since you are not really debugging EPIFlash, you may prefer to simply run it under MONICE.

Windows

Use the MAJIC Setup wizard to set up an EDB or MONICE shortcut that uses suitable board initialization files, or if you are using an RDI or MDI compliant debugger, to set up appropriate configuration files.

Linux

Open a command shell and **cd** into the directory containing your board initialization files. Run MONICE with the appropriate **-d** and **-v** switches (for more information on running MONICE, please refer to the *.manuals/10002.pdf* application note).

Running EPIFlash

Some board initialization files define a command alias, **epiflash**, for running the EPIFlash utility. If your board initialization file does this, then you can run EPIFlash by simply typing **epiflash** at the EDB or MON command prompt, or by using the **epiflash** option in the new **Execute Alias** button in EDB. To see if your board initialization file defines an **epiflash** command alias, enter this command:

```
da epiflash
```

If no **epiflash** command alias is defined, or if you are using an RDI or MDI compliant debugger, then you can simply load and run the EPIFlash utility like any other program.

- In EDB, click **File > Program to Debug...** and browse to the correct build of EPIFlash for your board. If you are a new user you should then click the **Load** button to download EPIFlash, then, click **Exec > Verify** to verify that it was correctly downloaded. The load and verify steps can be skipped when you're confident of your setup (in which case it will automatically load when you run it).
- In MONICE, use the Load (**L**) command to load the right build of EPIFlash for your board. If you are a new user you should then use the Verify Load (**VL**) command to verify that it was correctly downloaded. The Verify Load step can be skipped when you're confident of your setup.

```
L "..\samples\__\_____\epiflash.____" // MONICE Load example
VL
```

- In any other debugger, use the debugger's toolbar and/or commands to load and run EPIFlash. Remember that some debuggers provide a separate window for the semi-hosting user interface.

NOTE: Several pre-built versions of EPIFlash that are linked for different RAM addresses are included in the EDT software package. They may be found in the following directories:

```
.\samples\be\ram_0XXXXXXXXX\epiflash.axf // Big Endian ARM/XScale
.\samples\le\ram_0XXXXXXXXX\epiflash.axf // Little Endian ARM/XScale
.\samples\be\ram_0XXXXXXXXX\epiflash.elf // Big Endian MIPS
.\samples\le\ram_0XXXXXXXXX\epiflash.elf // Little Endian MIPS
```

Now start EPIFlash. In EDB, use the **View** menu to open the Program I/O window, then, click **Go**. In MONICE, enter the **G** command. When EPIFlash starts it will display the flash settings and image settings, then the main menu.

Flash Settings

EPIFlash displays the following flash configuration settings. These flash configuration settings must be correct for EPIFlash to work properly. The following sections explain how to set the flash settings.

After you have set them, you can save them via the **Configuration** menu so they will be remembered next time you run EPIFlash.

```
EPI Flash Programmer v3.1.4.1 - LE

Flash Device   : AT49BV6416B  x16  8MEG  Boot Bottom
Devices       : 1-Series  1-Parallel  1-Total
Sector Groups  :      2
Sector Count   :      8    127
Sector Size    :      8K    64K (bytes)
Device Base    : 0x10000000
Device Offset  : 0x00000000
Device Buffer   :           0 (bytes, max = 0)
```

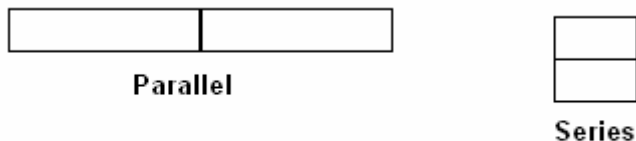
Flash Device

To set the **Flash Device**, select the **Flash Type** option from the main menu. First, it will ask for the flash part vendor, then, it will ask for the flash device type. For some flash parts it will also ask for the device width (8 vs. 16 bit) and sector organization (top vs. bottom boot mode). After you set the flash type, EPIFlash will display the new flash configuration settings so you can check. The **Sector Groups**, **Sector Count**, and **Sector Size** fields should reflect the details of the part type that you selected.

NOTE: If you do not find your flash device type in the list displayed by EPIFlash, then you should check the *.\manuals\epiflash_parts.txt* file to see if a compatible part is cross referenced there. If not, then go to SupportNet to see if an update is available — we're adding new flash parts all the time.

Multiple Devices

If your board has two flash devices of the same type connected in series or in parallel, EPIFlash can operate on them together. You can select two in a series or two in parallel via the **Flash Device** menu.



NOTE:

- They are connected in parallel if they occupy different byte lanes on the data bus and can be accessed simultaneously.
- They are connected in series if they are connected to the same byte lane(s), and the first address of the second part comes immediately after the last address of the first device.

Device Base

The **Device Base** should be set to the first address of the flash device, no matter whether you want to program all or just part of your flash device. This can be set from the main menu.

Device Offset

The **Device Offset** should normally be 0, unless your flash image will only occupy upper sectors of the flash part and you do not want to erase or program the lower sectors.

Device Buffer

Many newer flash devices provide a write buffer to faster programming. When you select the flash device EPIFlash sets the **Device Buffer** to the size of the write buffer for that part, or 0 if the part has no write buffer. During flash programming, EPIFlash uses buffered write mode if the device buffer is non-zero, or normal write mode if the device buffer is zero. If you experience trouble programming a part, but you are able to erase it, then you may need to change the device buffer size in the **Flash Device** menu.

Image Settings

After displaying the flash settings, EPIFlash displays the image settings.

Filename	:	test.bin	
File Offset	:	0	(0x00000000)
File Length	:	0	(0x00000000)
Image Size	:	0	(0x00000000)
Image Address	:	0x10000000	
Image Buffer	:	0x00800000	(bytes, Enabled)

Image Filename

The **Filename** is the file to program into the flash part(s). To specify the image filename, including the path to the file, select the **Image Filename** option in the main menu.

File Offset

The **File Offset** field allows you to start programming part way through the file, instead of starting with the first byte of the file. For example, you could skip over a header that was prepended to the file. Normally this option should be set to 0.

File Length

The **File Length** field is automatically set when you choose the image filename, and you can not change it (except by choosing a different file).

Image Size

Normally the **Image Size** should be set to match the file length, but you can use that setting to constrain the amount of flash being programmed if upper sections need to be preserved, or to operate on the entire flash instead of just the part corresponding to the image file. To set the image size, select the **Image Size** option in the main menu.

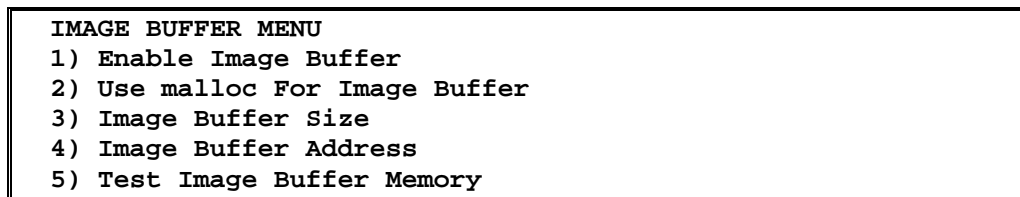
Image Address

The **Image Address** field shows the range of flash memory that is occupied by the selected image size. The start address is the device base address plus device offset, the end of the range is the start address plus the image size (the end address is not shown if the image size is 0).

Image Buffer

This is an advanced setting that is optional. It allows a copy of the image file to be saved locally when the device is programmed, so the program verify operation can complete faster. If your target has at least twice as much RAM as flash, and you are programming a large flash image, then you might want to enable the image buffer.

To set up an image buffer, click **Configuration > Image Buffer** to display the following menu.



First, set the **Image Buffer Size** to the flash device size (or total size, if you have multiple devices). For MIPS targets, set the **Image Buffer Address** to 1Meg above the start of your RAM area. For ARM and XScale targets, chose the option to **Use malloc For Image Buffer**. Then, enable the image buffer and optionally test it.

EPIFlash Operation

After you have the flash and image settings all correctly set, it is a good idea to save the settings so they will be remembered next time you run EPIFlash. You can save the settings using the **Configuration** menu.

Erasing

Use the **Erase** menu to erase part, or all, of the device, to verify that the area or device is erased, or to search for erased and programmed sections. The **Erase Device** option erases every sector of your flash part(s). The **Erase Image Sector(s)** option erases the sectors corresponding to the image address.

Programming

Use the **Program** menu to back up the flash part to a file, to program the image file into flash, or to search for programmed sections. This menu also provides options to erase the whole device or the sectors corresponding to the image address prior to programming the selected file.

Special Modes

Some target boards require special programming modes. For example, it may be necessary to XOR the address with 2 or 3 to ensure the flash command bytes are routed to the data bus byte lanes that are used by the flash device(s). This is a function of the processor's bus interface and endian mode of your memory system.

If your target requires special handling of address bits 1..0, set the XOR option accordingly via the **XOR Address** option in the **Program** menu. Please refer to your processor documentation for more information about this potential requirement.

It may also be desirable to byte-swap the image being programmed, backed up (saved from flash to a file), or verified. Byte swapping may be set using the Swap options in the Program menu.

Diagnostic Menu

Use the **Diagnostic** menu if you need to test the special modes listed above or if you need to check basic flash functionality.

Timeouts

Flash operations such as erase and program are performed by writing a command to the flash part, then polling the flash part for status. Most flash operations take much longer to complete than the single bus cycle that it takes to initiate them, so polling for status is how EPIFlash determines when it can advance to the next operation. If there is some problem with the hardware, the flash operation might never be able to complete, so EPIFlash employs a timeout mechanism to abort if it appears that an operation will not complete.

However, since EPIFlash is a generic program meant to run on any target that has sufficient RAM, it does not have a real time base to use for the timeout. Instead, it uses a large loop counter, which means the actual timeout period is a function of the speed of the processor. On particularly fast processors the timeout may occur too soon even though the flash operation would eventually complete. In this case, you can increase the timeout period for the various flash operations using the **Configuration** menu.