Application Note
0380-0194-10 Rev 3.1

Embedded Performance, Inc.
606 Valley Way, Milpitas, CA 95035
Telephone: (408) 957-0350
FAX: (408) 957-0307
e-mail: support@epitools.com
web: www.epitools.com

Using the EPI Flash Programming Utility

0380-0194-10 Rev 3.1

August 1, 2003

Introduction

The EPI Development Tools (EDT) package includes a utility for programming flash memory on a target system. The EPI Flash programming utility (EPIFlash) is an application which runs on the target system and allows the user to program and erase the target flash memory device(s) through the JTAG port.

EPIFlash is menu driven, and comes with support for parts using either the Intel or AMD programming algorithms. It is open-source and can be recompiled with the ARM or GNU tools. Information on EPIFlash can be found in Appendix E of this application note. The pre-built flash utility must be run from either EDBICE or MONICE (for ARM and XScale customers AXD may also be used), but the flash utility can NOT be run from an RDI, MDI or eXDI Monitor window.

The information in this manual is intended to provide a step by step guide to running EPIFlash. It documents how to configure the MAJIC and how to run the utility, along with special notes on good flash programming procedures.

How to Use This Manual

Section 1, Startup Files

Provides an overview of the files required to run MAJIC for standard reference boards and proprietary targets.

Section 2, MAJIC Setup Wizard

Provides instructions for running the MAJIC Setup Wizard in order to configure the MAJIC to run with EPIFlash.

Section 3, Configuration under Linux and Solaris

Describes how to configure MAJIC to run EPIFlash under Linux or Solaris.

Section 4, Running EPIFlash

Covers where EPIFlash is, and how to run it.

Appendix A, Trouble Shooting Guide

Provides solutions to run-time issues with EPIFlash.

Appendix B, EPIFlash Release Notes

Outlines the revision history, and feature additions, for EPIFlash.

Appendix C, Standard Reference Boards

Lists the current standard reference boards for which samples files have been created by EPI, as of July 2003.

Appendix D, Menu Descriptions

Provides a complete description of each menu in EPIFlash. Provides detailed information on all parameters and functionality.

Appendix E, Adding a New Flash Device Type

Describes how to modify the open source code for EPIFlash, in order to add a new flash part that uses either the AMD or Intel algorithms.

Additional Documentation

Additional documentation for MAJIC can be found in the manuals directory of the EDT installation. Specifically, it is recommended that users read the *MAJIC Quick Start Guide*, *MAJIC User's Manual*, *EDB User's Manual*, and the *Using MAJIC with the Intel XScale Micro-Architecture* application note.

Getting Support

Please do not hesitate to contact our technical support group if you have any questions or need assistance in configuring the MAJIC for your system. We recognize that these issues are complex, and are committed to making sure our tools work well for you. If you find that the flash part that you are using is not currently supported by EPIFlash please contact EPI technical support to check for updates. Flash parts are added daily, and any Intel or AMD algorithm-based flash part can be added upon request.

Startup Files

MAJIC depends upon two main sources of information in order to connect to a target -- the *startice.cmd* file, and the board initialization file.

The *startice.cmd* file contains the required options for MAJIC to connect to a specific target. This file can be created by the MAJIC Setup Wizard, or by hand, and should not include any board specific code.

The board initialization file is target specific, and contains the commands required to initialize the target memory controller. In the case where boot code has not been written for the target, the initialization file can be used to for board bring-up and hardware initialization. In the case where boot code has already been written and programmed into flash, the initialization file is simply a back-up in case the flash is erased or becomes dysfunctional. If the flash is accidentally erased during programming a memory initialization script may be the only way to bring the board back up (if the flash part is not socketed). This is because EPIFlash uses the target system RAM to run, and as a temporary storage space for the image to be programmed to flash, therefore, if RAM is inaccessible the EPIFlash cannot be run.

In the case of standard reference boards (such as the Intel Lubbock), EPI partners with the board manufacturer to create the target startup files. For proprietary targets an initialization script will need to be created before flash programming can be completed. More information about creating startup files for proprietary targets can be found in the *Creating Startup Files for MAJIC* application note.

Requirements for XScale

EPIFlash can be used with all processors supported by the MAJIC, but, for XScale targets there are special configuration considerations. XScale targets require different configuration files for programming flash and for debugging an OS kernel or boot code. The configuration files for debugging an OS kernel will not work for flash programming, and visa versa – if the wrong sample files are chosen than flash programming will **NOT** work. Therefore, when selecting sample files do **NOT** choose a directory that ends in `_mv`, `_wince`, `_vx`, and `_qnx`!

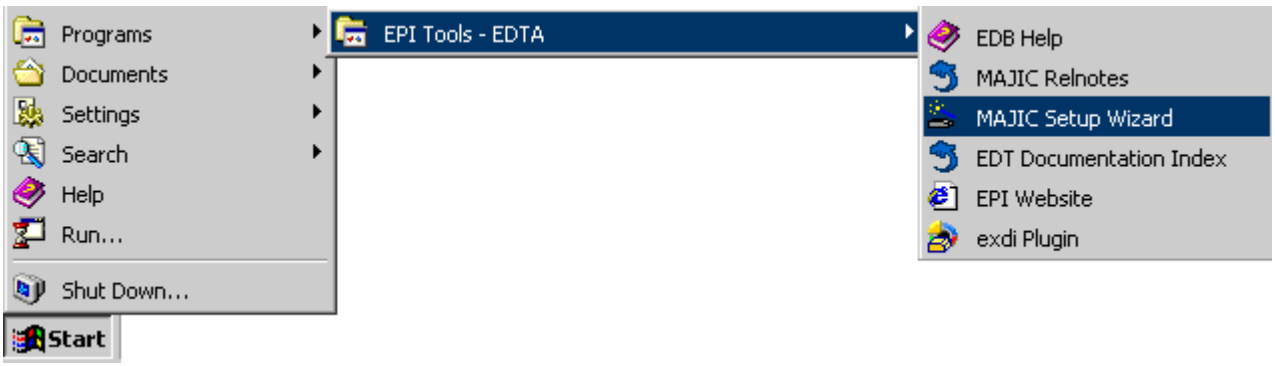
If you are just getting started with an XScale target it is recommended that you review the *Using MAJIC with the Intel XScale Micro-Architecture* application note.

MAJIC Setup Wizard

To run EPIFlash under Windows it is necessary to create a shortcut to the debugger, which references the appropriate startup files. To do this, run the MAJIC Setup Wizard found in the Start menu. For customer that have already created a shortcut for flash programming as described in the *Creating Startup Files for MAJIC* application note, this step can be skipped.

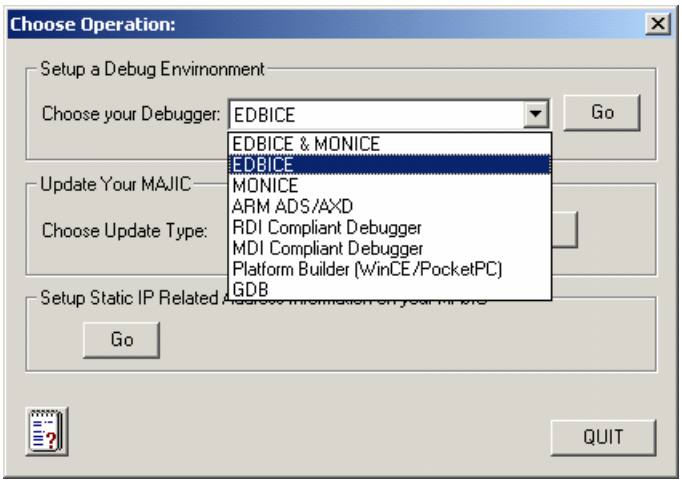
1. Run the MAJIC Setup Wizard found in the `Start` menu under

`Start-> Programs-> EPI Tools - EDTx-> MAJIC Setup Wizard`.



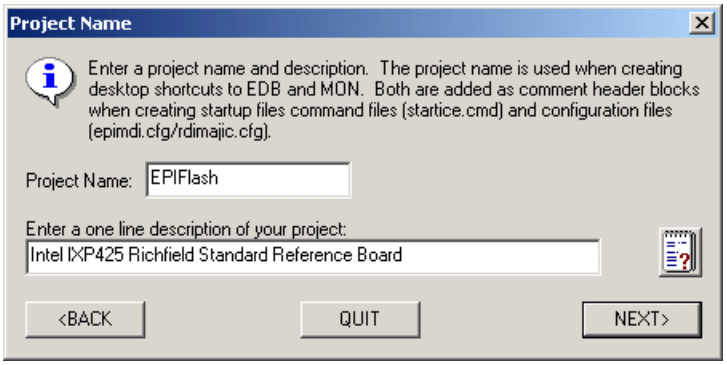
2. At the first screen read the description of the MAJIC Setup Wizard and then select `Next`.

3. Under `Choose your Debugger` select `EDBICE, MONICE` or `ARM ADS/AXD` and click `Go`. `RDI Compliant Debuggers, MDI Compliant Debuggers, Platform Builder, and GDB` cannot be used for flash programming.

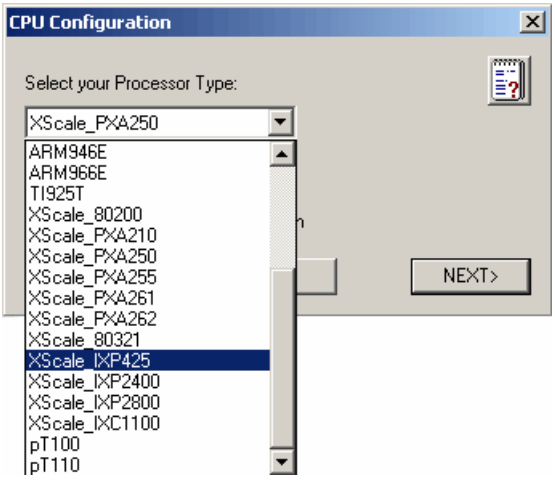


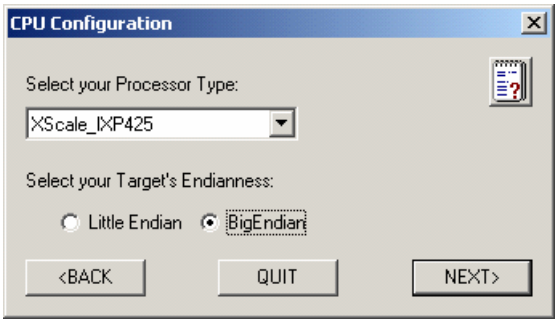
Be sure to select the `Go` button that is in the `Setup a Debug Environment` box. There are three `Go` buttons on the screen and each is associated with their own function.

4. Choose a `Project Name` and `Enter a one line description of your project`, choose `Next`.

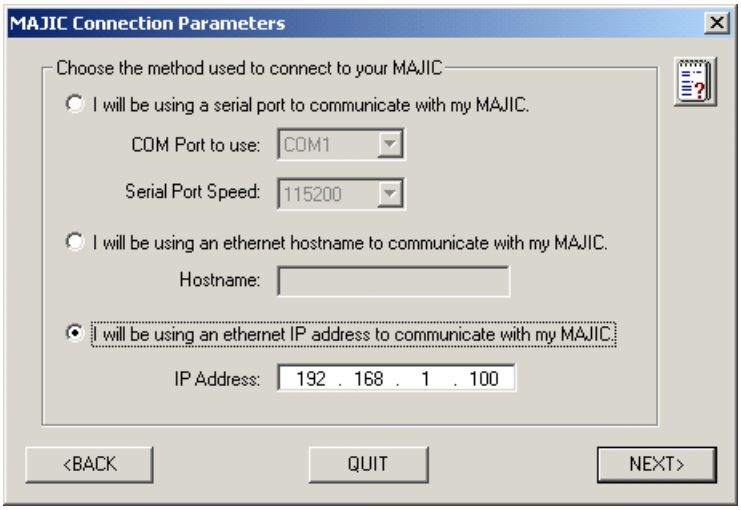


5. `Select your Processor Type` and `Select your Target's Endianness`, click `Next`. Choosing the incorrect endianess can result in EPIFlash not working!



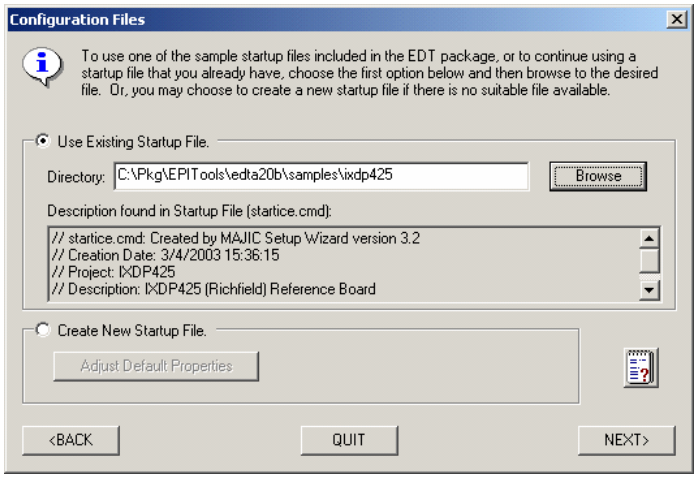


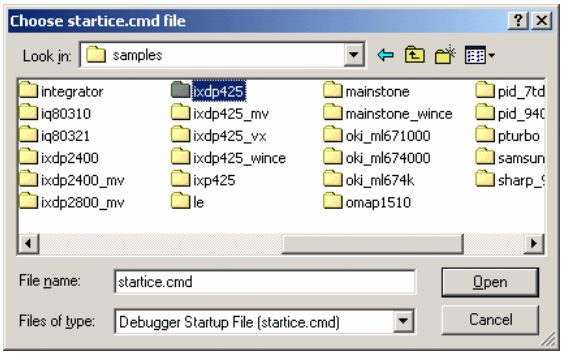
6. Choose your connection type. If it is the first time connecting to MAJIC the `Serial port` must be used until an IP address has been programmed. If Ethernet is being used with a cross-over cable, an IP address must be used rather than a hostname. For flash programming it is recommended that Ethernet be used -- as it will be significantly faster than a serial communication. For more information on setting up the IP address see the `MAJIC User's Manual` on page 9.



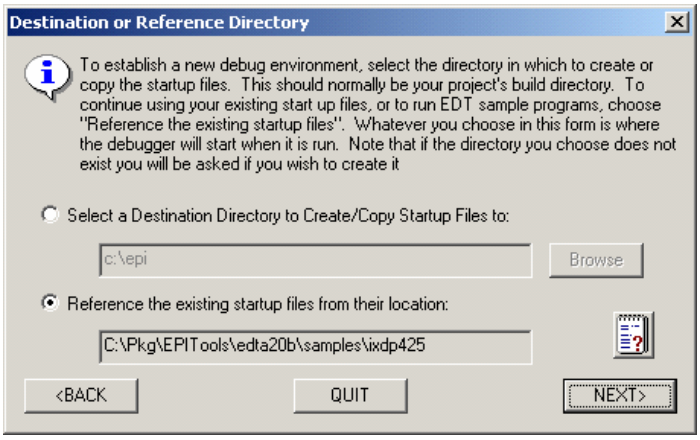
7. In the Configuration Files window choose Use Existing Startup Files. Browse to the samples directory and choose the sample folder for your reference platform. If there are no sample files for your target see the *Creating Startup Files for MAJIC* application note.

Note: For XScale customers, do **NOT** select a sample folder that has an OS extension, such as `_wince`, `_vx`, `_mv`, or `_qnx`! The settings for an embedded operating system will interfere with flash programming! See *Requirements for XScale* at the beginning of this document for more information.

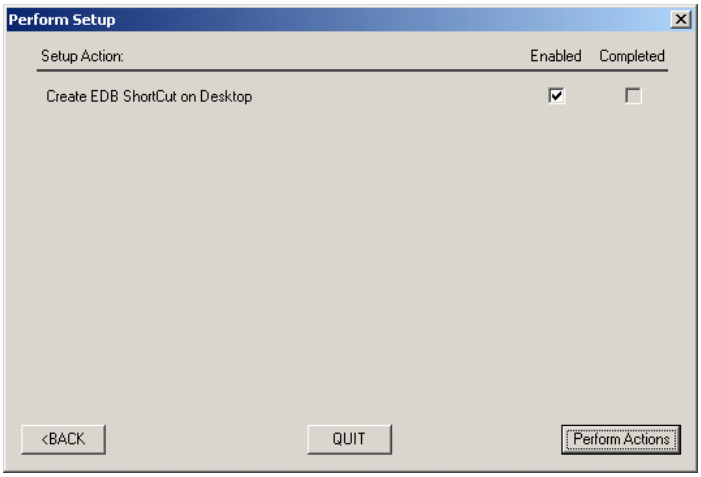




8. On the next screen choose Reference the existing startup files from their location.



9. Click `Perform Actions`, and then `Done.`



A shortcut for EDB or MONICE should pop up on your desktop. Power up your target and the MAJIC, then double-click on the desktop shortcut. Verify that you are able to connect to your target board, make sure that you see the message "JTAG connection established" in red letters. Use this connection in the future for flash programming.

Now that the MAJIC Setup Wizard is complete it is recommended that the *startice.cmd* file be reviewed for compatibility with EPIFlash. Samples files do not always default to settings compatible with the flash utility, so it is important to double check all options in the *startice.cmd* file. Open the *startice.cmd* file with a text editor and review the following options, changing those that do not match these requirements:

Common Target Requirements:

```
eo semi_hosting_enabled = ON           // Enable Semihosting support
```

ARM/XScale Target Requirements:

```
eo top_of_memory       = 0x????0000    // top of RAM on target
eo semi_hosting_vector = 0x8           // Use default vector for Semihosting
```

XScale Target Requirements:

```
eo reset_at_load       = OFF     // DON'T Reset CPU/target at load time
eo trgt_resets_jtag    = NO      // Target reset does not reset JTAG controller
```

ARM/XScale Recommendation:

```
eo vector_catch        = 0xFF    // Set Vector Catch, break on any exception
```

MIPS Target Requirements:

```
eo load_osboot         = OFF     // don't auto-load osboot.sys along with program
```

Configuration under Linux and Solaris

For Standard Reference Boards where the initialization files are already created, the next step is to place the files in the appropriate directory under Linux/Solaris. MONICE will look for the startup files in three places; in the current directory, in the `/opt/edtx/bin` directory, and in the `PATH`. Therefore, in order for MONICE to find the samples files they must either be placed in the `/opt/edtx/bin` directory; be in a directory that is on the path, such as the project directory; or MONICE must be called from the directory that contains the startup files. Once the files are in the appropriate location MONICE can be called with the appropriate command line options.

For proprietary targets see the *Creating Startup Files for MAJIC* application note.

Running EPIFlash

EPIFlash comes pre-built for MIPS and ARM/Xscale, little Endian and big Endian targets.

For MIPS targets, there are two pre-built Flash Utilities: flash0 (for kseg0, located at 0x80000000) and flash1 (for kseg1, located at 0xA0000000). For MIPS targets the pre-built Flash Utility can be found in:

For Little Endian Targets:

```
...\EPITools\edtm20\samples\le
```

For Big Endian Targets:

```
...\EPITools\edtm20\samples\be
```

For ARM/XScale targets, the pre-built Flash Utility is built for a number of different target memory (RAM) locations. If your target system has RAM located at the pre-built memory locations for the Flash Utility, you can use the pre-built utility. If EPIFlash is not compiled for your specific memory region, contact support@epitools.com for an update. For ARM/XScale targets, the pre-built Flash Utility is named *flash.axf*, and can be found in:

For Little Endian Targets:

```
...\EPITools\edta20\samples\le\ram_0x00008000
...\EPITools\edta20\samples\le\ram_0x0C008000
...\EPITools\edta20\samples\le\ram_0xA0008000
...\EPITools\edta20\samples\le\ram_0xC0008000
```

For Big Endian Targets:

```
...\EPITools\edta20\samples\be\ram_0x00008000
...\EPITools\edta20\samples\be\ram_0x0C008000
...\EPITools\edta20\samples\be\ram_0xA0008000
...\EPITools\edta20\samples\be\ram_0xC0008000
```

EPIFlash is open-source and can be found in:

Flash.c Source:

```
...\EPITools\edta20\samples
```

EPIFlash is loaded and executed from a debugger in the same manor as a user application would be. In this way EPIFlash is downloaded to the target and executed out of target RAM. EPIFlash then reads the flash image file (the image to be programmed) into a RAM buffer and program it into the flash device(s).

Starting the Debugger

For this example we will use the IXDP425 Richfield board. EPIFlash can be run with EDB under Windows (or the ARM tools), or MONICE under both Windows and Linux. If running under Windows, start by double clicking on the EDB or MONICE shortcut created on the desktop by the MAJIC Setup Wizard. If running under Linux, change directories to the folder that contains the initialization files for the target, in this case the `/opt/edta/samples/ixdp425` directory, then call MONICE:

```
[root@root /] cd /opt/edta/samples/ixdp425
[root@root ixdp425] monice -vixp425 -d demomajic5:e
```

Make sure that the target has started up correctly, and that MAJIC has connected without receiving any connection errors. A good connection will look like this:

```
Processing register file: /opt/edta/bin/arm/spaces.rd
Processing register file: /opt/edta/bin/arm/majic.rd
Reading command history from: /opt/edta/bin/arm/startedb.hst
Establishing communications with remote target via demomajic5...
Connection verified
Target System:  EPI Majic Probe, Version: 3.2.3, S/N 0210G010
Hardware Rev:   90:3:3:20
Target CPU:     IXP425
Ethernet:       at address 00:80:CF:00:14:E5
IP address:     205.158.243.204, Subnet mask: 255.255.255.0
Trace Buffer:   1 frames
Profiler:       Not Installed
Connected via:  Ethernet UDP/IP
Device name:    demomajic5
Target Endian:  big
Start Address:  00000000:
EPI-OS (HIF):   on
Reset Mode:     capture
Reading commands from /opt/edta/samples/ixdp425/startice.cmd
MON> +q  // Enter quiet mode
Reading startice.cmd file
Notification from the target:
Target power detected on VREF
Auto JTAG detection process detected 1 TAP
JTAG connection established
Reading ixdp425.cmd
Executing One Time Setup Commands
Executing Target Init Commands
Finished reading ixdp425.cmd
Finished reading startice.cmd
```

Note: For the Richfield target (and most IXP425 boards based upon the Richfield, such as the ADI Coyote target) it is necessary to run an unlock command script to disable the write-protect on the flash device. This unlock command has been placed in a script file called *flash.cmd*, which can be found in the `ixdp425` folder. To read this file in, and execute the unlock command, type the following on the command line:

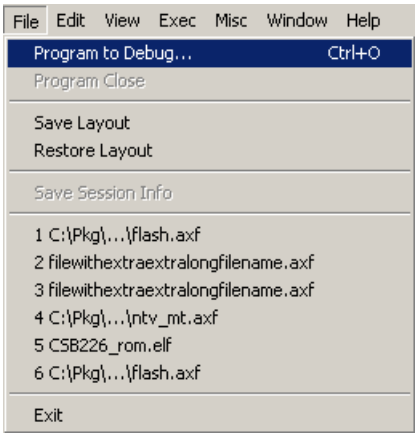
MON> **`fr c flash`**

A confirmation that the flash device has been unlocked should be printed to the screen:

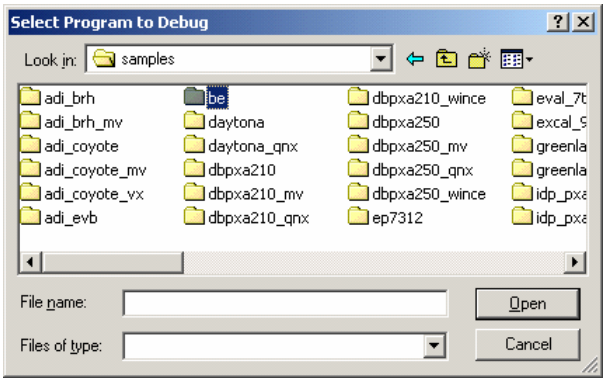
```
Enable Write to Flash Device on IXDP425
```

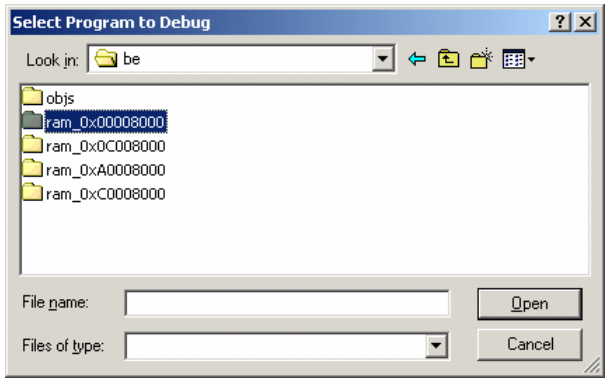
Next, pull EPIFlash into the debugger as if it was a program to debug.

Under EDB, do this from the `File` menu by selecting `Program to Debug`:

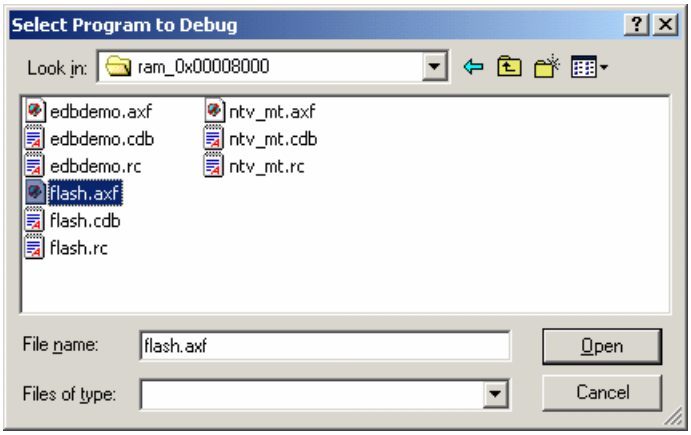


Then browse to the location of the pre-built flash utility. This will be under the EPI Development Tools installation, typically `C:\Program Files\EPITools\EDTx`. Next choose the correct folder for the target: `be` for big endian targets; `le` for little endian targets. For ARM and XScale there will be another set of folders; choose the folder that correlates to the base address of the target's SDRAM.



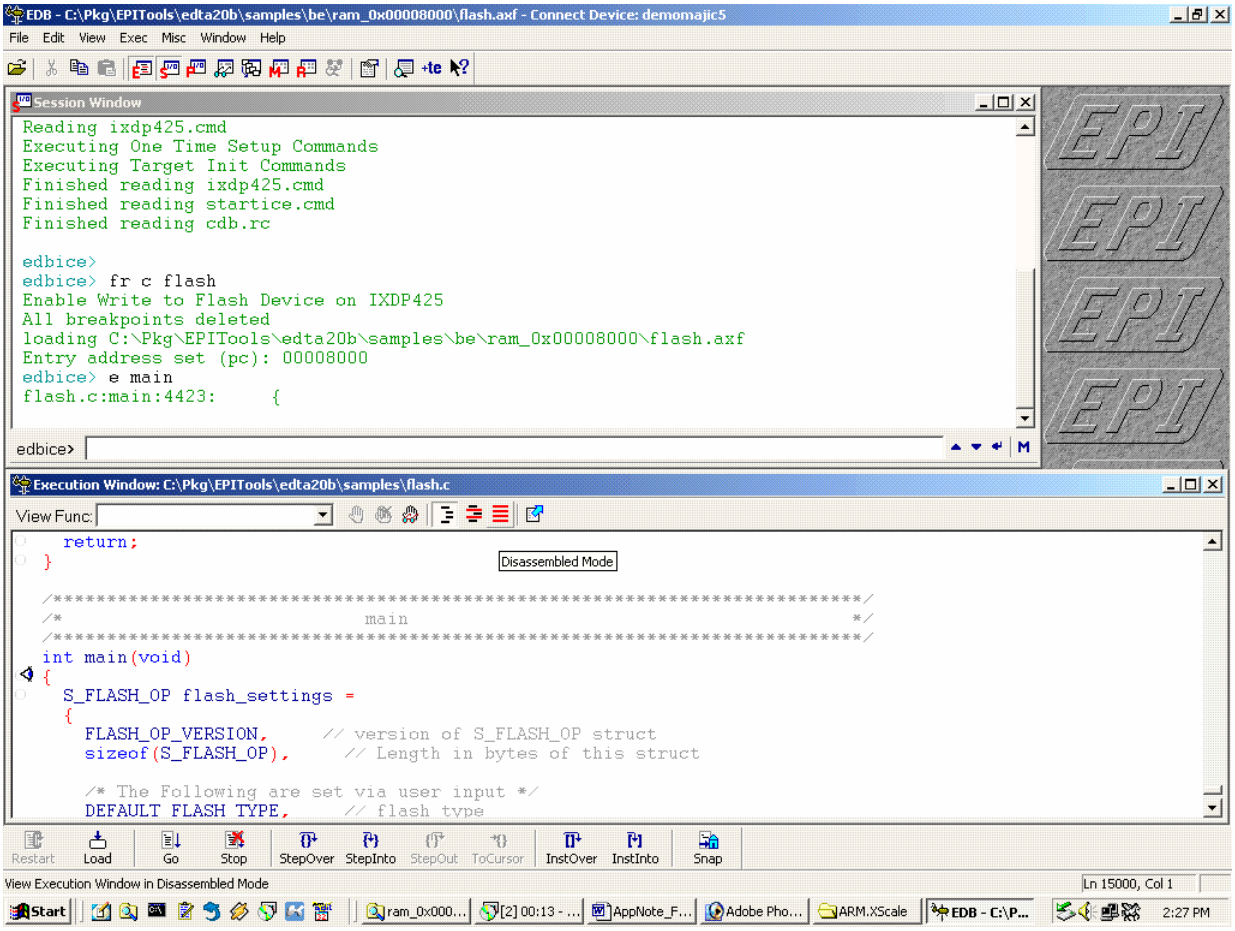


Select the flash utility, *flash.axf* for ARM and XScale targets; *flash0* or *flash1* for MIPS targets, and click Open.



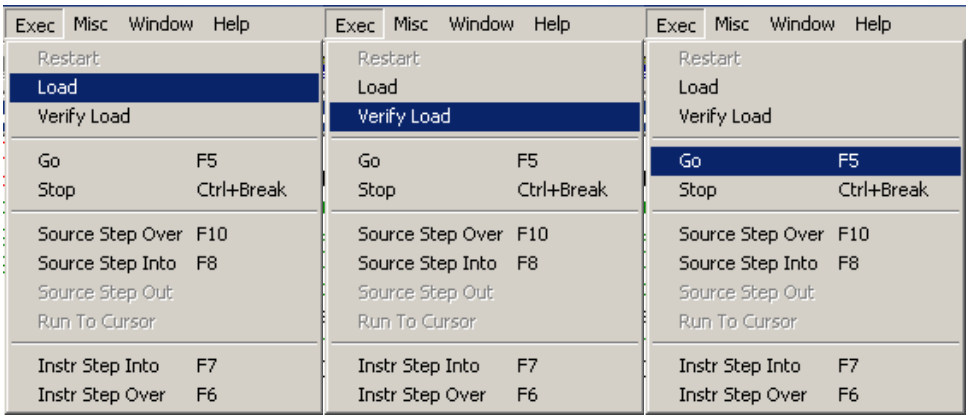
If asked the location of *flash.c*, browse back up to the `samples` directory.

The EDB window should look something like the following:



You can see the source code for EPIFlash displayed it the Execution window at the bottom of the screen.

The debug information for EPIFlash has now been read by the debugger, but the program has not been loaded to the target. To do that, go to the `Exec` menu at the top EDB and choose the `Load` button. Although you can jump ahead by just hitting the `Go` button, which will automatically load the file, it is recommended that you perform a `Load` and a `Verify Load` the first time you use EPIFlash. Performing a `Verify Load` provides verification that the memory controller was properly initialized by showing that programs can be loaded into memory (RAM). The `Load` command is available as an icon in the execution toolbar, or in the `Exec` menu. The `Verify Load` button is only available under the `Exec` menu. After you have performed a `Verify Load`, run the program by choosing `Go` through the `Exec` menu, by clicking the `Go` button, or by hitting the `F5 key`.



Under MONICE load the application into the target by typing the following commands:

```
MON> l ../be/ram_0x00008000/flash.axf
loading /opt/edta/samples/ixdp425/../be/ram_0x00008000/flash.axf
 section be/ram_0 from 00008000 to 000142ff
 section be/ram_0 from 00014300 to 00014d87
 section be/ram_0 from 00014d88 to 0001520f
Entry address set (pc): 00008000
!!!:
00008000:     e59f0034    LDR    r0,0x803C
```

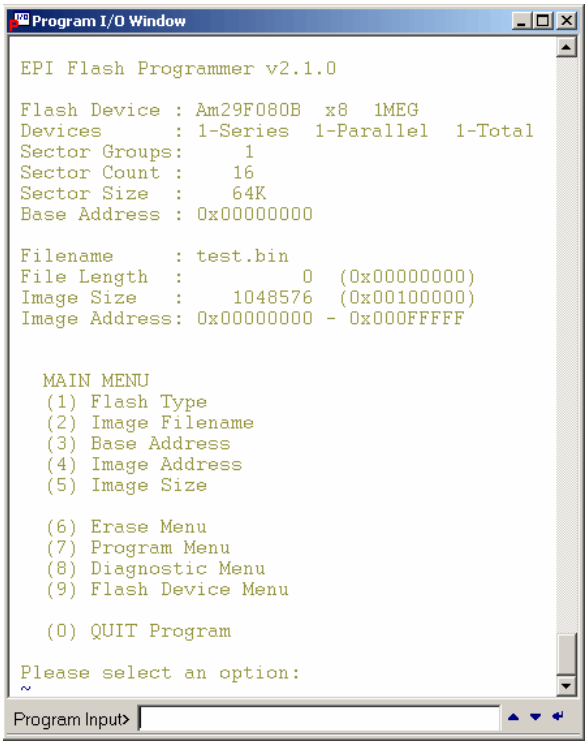
It is always recommended that you perform a verify load in order to ensure that the target RAM has been correctly initialized:

```
MON> vl
checking /opt/edta/samples/ixdp425/../be/ram_0x00008000/flash.axf
 section be/ram_0 from 00008000 to 000142ff
 section be/ram_0 from 00014300 to 00014d87
 section be/ram_0 from 00014d88 to 0001520f
Verify succeeded.
```

Then run the application:

```
MON> g
```

In EDB the Program I/O window should now pop up with the user interface; in MON the user interface will print to the command line.



Configuring Settings

The first time EPIFlash is run it will NOT default to correct settings for the target, each of the options must be configured. The options can be changed by selecting the menu options `Flash Type, Image Filename, Base Address, Image Address`, and `Image Size` by their numbers on the command line. Extra options are also available under the `Flash Device Menu`. As each option is changed the information at the top of the utility will be updated to reflect the current settings. If, for example, the wrong filename is entered, the previous setting will not change. Be sure to double check all settings before erasing or programming the flash!

Start by choosing the flash part for your target. You can find out what flash part is on the target by looking in the board documentation, or just reading the number off of the flash part itself. Selecting the `Flash Type` menu, a list of all available flash types will be printed. If the flash part needed is not in the list, contact support@epitools.com for an update. The abbreviated list of supported flash parts from the EDT2.0b release looks like this:

```
Please select an option: 1

                    FLASH TYPE MENU

 1: Am29LV400BB x8  512K        20: Am29DL163DT x16   2M
 2: Am29LV400BB x16 512K        21: 28F008B3B   x8    1M
 3: Am29LV800BB x8    1M        22: 28F008B3T   x8    1M
 4: Am29LV800BT x8    1M        23: 28F016B3T   x8    2M
 5: Am29LV800BB x16   1M        24: 28F320B3B   x16   4M
 6: Am29LV800BT x16   1M        25: 28F640B3B   x16   8M
 7: Am29F040B   x8  512K        26: 28F160C3B   x16   2M
 8: Am29F080B   x8    1M        27: 28F160S5    x16   2M
 9: Am29LV160BB x8    2M        28: 28F320J3    x8    4M
10: Am29LV160BT x8    2M        29: 28F320J3    x16   4M
11: Am29LV160BB x16   2M        30: 28F640J3    x16   8M
12: Am29LV160BT x16   2M        31: 28F128J3    x16  16M
13: Am29LV320DT x16   4M        32: 28F256K3    x16  32M
14: Am29LV641M  x16   8M        33: 28F640W18B  x16   8M
15: Am29LV128M  x16  16M        34: 28F640W18T  x16   8M
16: Am29LV256M  x16  32M        35: AT29LV1024  x16 128K
17: Am29DL163DB x8    2M        36: SST39LF200A x16 256K
18: Am29DL163DB x16   2M        37: SST39LF400A x16 512K
19: Am29DL163DT x8    2M

Please select your flash type (1..37): 31
```

Double check that the bus width and flash size are correct for your target. Choosing the wrong bus width can lead to programming failures farther down the line.

```
Flash Device : 28F128J3  x16  16MEG
```

After the `Flash Type, Base Address, and Image Size` have been selected all settings should be double checked in the summary at the top of the session.

Bootloaders

If a bootloader is being programmed the same procedure is used for other images, with one exception. Some bootloaders are shipped endian-swapped. This will generally be the case if the bootloader was compiled with a little endian compiler – but the target is currently configured for big endian (or visa-versa). In this case it is necessary to swap the endianness of the file before programming. This can be done under the `(7) Program Menu`, using the `(9) Swap Image Endianess` option.

```
PROGRAM MENU
  (1) Program Image
  (2) Erase Image,  Program Image, Verify
  (3) Erase Device, Program Image, Verify
  (4) Verify Image Erased
  (5) Verify Image Programmed
  (6) Find Programmed Sections
  (7) Find Verify Failed Sections
  (8) Backup Flash Device (to Disk)
  (9) Swap Image Endianness

  (0) Return to Main Menu

Please Select an Option: 9

Enter Swap Image Endian Operation:
(1)  Swap Endian: NONE
(2)  Swap Endian: 16-Bit
(3)  Swap Endian: 32-Bit
Enter 1, 2 or 3:2
```

Note: For programming MontaVista bootloaders into big endian targets choose to Swap Endianness by 16-Bits. For programming WinCE bootloaders into big endian targets choose to Swap Endianess by 32-Bits. The .bin file should be programmed in for bootloaders – NOT the .nb0 file.

Backing up the Flash Device

If you do not have an original copy of the contents of flash on your local computer it is recommended that you create a back-up before programming. To do this go to the `(7) Program Menu`, then choose option `(8) Backup Flash Device (To Disk)`. You will be prompted for a path and filename where the backup will be stored on your local computer.

```
PROGRAM MENU
(1) Program Image
(2) Erase Image,  Program Image, Verify
(3) Erase Device, Program Image, Verify
(4) Verify Image Erased
(5) Verify Image Programmed
(6) Find Programmed Sections
(7) Find Verify Failed Sections
(8) Backup Flash Device (to Disk)
(9) Swap Image Endianness

(0) Return to Main Menu

Please Select an Option: 8

Enter Filename (Including Path):
/home/epi/richfieldimage.bin

File Exists, Do you wish to Overwrite? (Y/N):y
You Selected Backup Flash to Disk, Are You SURE? (Y/N):y
```

After the image has been backed-up be sure to verify that it has been uploaded successfully. To do this return to the main menu and change the `Filename` to reference the back-up file on the host computer. The `Image Size` of the back-up file, which will be filled in automatically by EPIFlash, should be equal to the `File Length` selected in the last section. Next, return to the `Program Menu` and choose `(5) Verify Image Programmed`. If this passes successfully then it is safe to proceed to the next step.

```
EPI Flash Programmer v2.1.0

Flash Device : 28F128J3  x16  16MEG
Devices      : 1-Series  1-Parallel  1-Total
Sector Groups:     1
Sector Count :   128
Sector Size  :   128K
Base Address : 0x50000000

Filename     : /home/epi/richfieldimage.bin
File Length  :    4194305  (0x00400000)
Image Size   :    4194305  (0x00400000)
Image Address: 0x00000000 - 0x00400000


  PROGRAM MENU
  (1) Program Image
  (2) Erase Image,  Program Image, Verify
  (3) Erase Device, Program Image, Verify
  (4) Verify Image Erased
  (5) Verify Image Programmed
  (6) Find Programmed Sections
  (7) Find Verify Failed Sections
  (8) Backup Flash Device (to Disk)
  (9) Swap Image Endianness

  (0) Return to Main Menu

Please Select an Option: 5

Verifying Flash Image...

Flash Image  : PASSED
```

Testing Programming

The next step is to test erasing and programming the flash device – without affecting the current program in memory. This is recommended as a precaution to avoid accidental erasures of boot code, which can prevent board bring-up if there is no target initialization script. For this test, find a small area in the flash part that is unused, and attempt erasing and programming in that unused portion. Functions in the `(6) Erase` Menu will allow you to find unused areas of memory, such as `(5) Find Erased Sections`.

```
  ERASE MENU
  (1) Erase Device
  (2) Erase Image Sector(s)
  (3) Verify Device Erased
  (4) Verify Image Erased
  (5) Find Erased Sections
  (6) Find Programmed Sections

  (0) Return to Main Menu

Please Select an Option: 5

Enter Erase Section Size (1024 is the minimum
 size to qualify as a section): 1024
```

```
Find Erased Sections...

Bytes Erased : 0x0039AE5C (3780188)
First Address: 0x500251A4
Last  Address: 0x503BFFFC

Bytes Erased : 0x0003EFFC (258044)
First Address: 0x503C1004
Last  Address: 0x503FFFFC


Erased Sections Found:  2
```

Take note of the largest erased section found, then change the `Image Address` and `Image Size` to define the erased section. For example, filling in the `Image Address` and `Image Size` based upon the information above would look like this:

```
EPI Flash Programmer v2.1.0

Flash Device : 28F128J3  x16  16MEG
Devices      : 1-Series  1-Parallel  1-Total
Sector Groups:     1
Sector Count :   128
Sector Size  :   128K
Base Address : 0x50000000

Filename     : c:\epi\filename.bin
File Length  :    4194305  (0x00400000)
Image Size   :    3780188  (0x0039AE5C)
Image Address: 0x000251A4 - 0x003BFFFC
```

Next, use the Program Menu to `(2) Erase Image, Program Image,` and `Verify` code into the defined area of memory. After flash programming has been proven successful it is safe(r) to program the entire device.

Programming Flash

To program the new image into flash first double check all setting for the target. Ensure that the Flash Part, Filename, Base Address, Image Address, and Endianess are all correct. Once this is verified choose the option `(2) Erase Image, Program Image, Verify` under the `(7) Program Menu`.

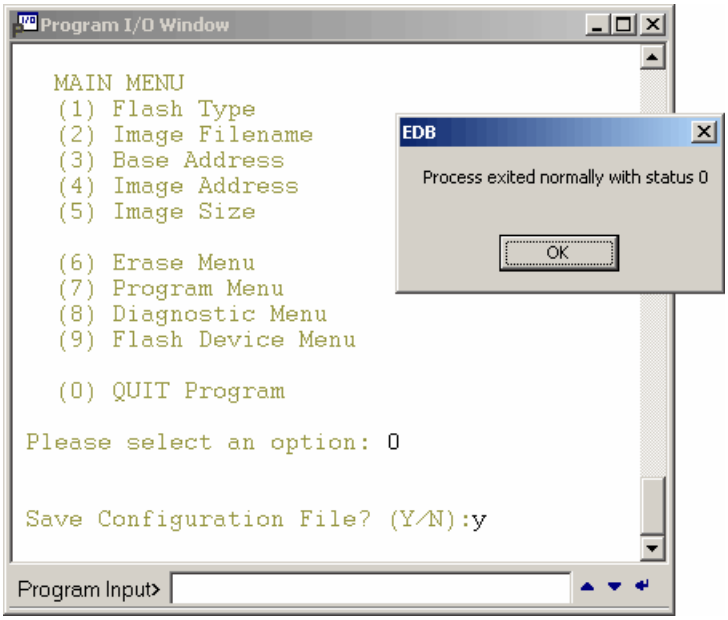
```
  PROGRAM MENU
  (1) Program Image
  (2) Erase Image,  Program Image, Verify
  (3) Erase Device, Program Image, Verify
  (4) Verify Image Erased
  (5) Verify Image Programmed
  (6) Find Programmed Sections
  (7) Find Verify Failed Sections
  (8) Backup Flash Device (to Disk)
  (9) Swap Image Endianness

  (0) Return to Main Menu

Please Select an Option: 2
You Selected ERASE Device, PROGRAM, Verify, Are You SURE? (Y/N):y

 Erasing Device...
Device Erase - Complete

Verifying Device Erased...
```

```
Device Erased: PASSED

Programming Device...
 Done!

Verifying Flash Image...

Flash Image  : PASSED
```

This will be the results displayed when all sections erase, program and verify correctly.

Closing EPIFlash

The last step is to exit EPIFlash – and save the settings. To do this select `(0) Return to Main Menu`, then select `(0) Quit Program`. By selecting yes to `Save Configuration File? (Y/N)`, all settings will be remember the next time EPIFlash is run.



Or under MONICE:

```
Please select an option: 0


Save Configuration File? (Y/N):y
Process exited normally with status 0
__sys_exit+0x10:
00014244:     eafffffe    B      0x14244     ; __sys_exit+0x10
MON>
```

When the `Process exited normally with status 0` message appears then the program exited cleanly with no errors.

Appendix A – Trouble Shooting Guide

EPIFlash will not be able to program your target flash device if the Target:

a) Flash Device is non-functional.

b) Memory (RAM) is non-functional.

c) Flash Device is mapped out by Memory Controller initialization logic.

d) Memory Controller is not initialized.

Some targets require boot code in flash to initialize the memory controller (a command script file will not work), if this boot code becomes corrupted or erased, you will not be able to load and run EPIFlash.

1. Is there a way to use the JTAG directly to program FLASH? Does EPI support any methods that do not use RAM?

EPI does not support any methods to program JTAG directly, without using RAM. We have looked into this type of support in the past, but the download speeds for this method are incredibly slow, on the order of 15minutes per Megabyte. We have no plans to add support for this in the future. Our recommendation is to use a socketed flash part for your development board in order to work around having non-working RAM.

2. My flash part has random data in it, and I cannot connect to the MAJIC. When I take my flash part out, I can connect fine. Why?

If there is garbage in the flash device and the processor tries to execute it as instructions, then it could accidentally be interpreted as instruction code. This can cause the target to lock up. If you erase your flash, take it out, or reprogram it, these problems will go away. In order to avoid this happening in the future you should map the flash as read only in the MC table.

3. I am having problems programming flash on the Richfield board (IXDP425), when I try to program the flash utility seems to hit a breakpoint, and I get an error message in the session window. Any ideas?

Make sure that the EDT2.0b release, and the startup files from the samples/ixdp425 directory, are being used. If so, then ensure that the script to unlock the flash device is being run. To do this type the following before running the flash utility:

```
EDB> fr c flash
```

4. The flash utility seems to be running on the target, but I don't get any feedback from it - i.e., the user interface/ Program I/O window does not appear... ?

If the Program I/O window is not popping up - or has no data, then check the following:

a) When you load EPIFlash, do a "verify load". If this comes out with failures then see (f) and (h).

b) Did you use a *startice.cmd* file from a folder in the samples directory that ends in `_wince`, `_vx`, `_qnx`, or `_mv`? If you did you need to create a second shortcut that does not use an embedded OS *startice.cmd* file.

c) Is `semi_hosting_enabled` on? Double check this setting in your *startice.cmd* file. It must be on!

d) Is the `semi_hosting_vector` set? This needs to be set equal to 0x8 in the *startice.cmd* file.

e) Is your `top_of_memory` setting in the *startice.cmd* file correct? This needs to be pointed at the top of RAM.

f) Is the memory controller being initialized, and is the Flash being set up?

g) Make sure that if you have defined `load_entry_pc` in your *startice.cmd* that you set it equal to `yes`.

h) If none of these suggestions help try writing to RAM and seeing if this works. You can do this by typing

```
MON> ew 0x0 = 1,2,3
```
To see if the memory changed type:

```
MON> dw 0x0 L 4
```
You can also run the native memory test provided with MAJIC; type the following for more information:

```
MON> h mt
```
i) After running through these basics contact support@epitools.com.

5. The Program I/O window pops-up, and I am able to enter my values. But, when I try to erase or program I get failures…

Common problems why EPIFlash will not be able to erase or program:

a) The flash device is write protected. This can by implemented in either software (like the ixdp425) or with hardware (such as with a jumper or switch).

b) An incorrect Flash part was selected. To double check look at the writing on the top of the flash device – the flash type is generally printed on the device.

c) An incorrect `Bus Width` was selected in EPIFlash.

d) The flash device `Base Address` is incorrect. Double check your memory map – this address is virtual, and with the memory controller initialized.

e) The flash utility Endian is Incorrect. The endianess of the program loaded into the debugger does not match the endianess of the target. Double check this in your startice.cmd file, or type

```
MON> do trgt_little_endian
```
To find out the endianess the target is set to.

f) Semihosting support is not enabled. Double check this in your *startice.cmd* file, or type

```
MON> do semi_hosting_enabled
```
g) The top_of_memory EO option is Incorrect (ARM and XScale only). Double check this in your *startice.cmd* file, or type

```
MON> do semi_hosting_enabled
```
h) Using sample start-up files from a folder that ends in _WinCE, _MV, _QNX, or _VX (XScale only).

Appendix B – EPIFlash Release Notes

NEW TO VERSION 2.1.0

* Find Verify Failed Sections.
* Enhanced Configuration file structure to work with future versions.
* Configuration file has changed, a 2.0.9 or earlier configuration file can not be restored with 2.1.0.
* Fixed problem with erasing image sectors beyond image size.

NEW TO VERSION 2.0.9

* Swap Endianness of Image File.
* Image Size can be set to Current Image Size.
* Fixed problem with saving configuration file.
* Configuration file has changed, a 2.0.8 configuration file can not be restored with 2.0.9.

NEW TO VERSION 2.0.8

* Configuration saved.
* Find Programmed Sections.
* Image File Length displayed.
* Back-up Flash Device to Host Computer.
* Image Size can be set to Image File Length.
* Flash Type Menu Reconfigured to Support more Flash Types.
* Added Support for more than One Flash Device on Bus (in Series).
* Added Support for more than One Flash Device on Bus (in Parallel.).

  ALL x16 Devices Supported, now have Support for Devices in Parallel.

NEW TO VERSION 2.0.5

* Find Erased Sections.

NEW TO VERSION 2.0.4

* Added Verify Device Image to Diagnostic Menu.
* User Input Now Be In Hex, K or Meg For Numeric Input (Address or Size).
* Added Support For Two Flash Devices in Parallel.

  Note: Not All Devices Supported Support Devices in Parallel.

* Entering values for Base/Image Address or Image Size:

For versions 2.0.0 thru 2.0.3 of the Flash Utility, values entered for Device Fill Value, Base Address or Image Address are always assumed to be HEX (user did not have to enter a number with a 0x prefix), values entered for Image Size were always assumed to be decimal (and could not be entered in hex).

With version 2.0.4 or later of the Flash Utility, you may now enter values for Fill Value, Address or Size, in decimal, hex, K or M (K = 1024, M = 1024K). When entering numbers in M or K format, you may only use decimal values, do not use the hex numbers A-F or fractions. After Entering a value, you must hit the Enter Key. If you hit the Enter Key without entering a value, no change will be made to the entry. NOTE: A-F, M, K, and X are not case sensitive.

Examples:

To enter the values for 512K, 1MEG, 2MEG, 4.5MEG and 10MEG:

```
   ---            1M            2M           ---           10M
  512K          1024K         2048K         4608K         10240K
 80000X        100000X       200000X       480000X        A00000X
0x80000       0x100000      0x200000      0x480000       0xA00000
 524288        1048576       2097152       4718592       10485760
```

Before any operation which will result in the flash device being erased or programmed, you will be prompted with the operation to be performed, to proceed press `y` or `Y`, to cancel press `n` or `N`, then press the Enter key.

Example:

```
You Selected ERASE Device, PROGRAM, Verify, Are You SURE? (Y/N):
```

NEW TO VERSION 2.0.8

* With version 2.0.8 of EPIFlash you can NOW backup your target flash memory to a file on your host computer.

Before erasing or programming you target's flash device, we recommend that you Backup your flash memory, and verify the contents of flash memory with that of the backup image file. This will allow you to become familiar with the flash utility and validate that you can verify target flash memory.

Appendix C – Standard Reference Boards

A complete list of standard reference boards which have been verified by Embedded Performance Inc., and for which samples files have been created.

For ARM and XScale Reference boards these are located in the `...\EPITools\edta20\samples` directory:

Verified XScale Targets:

Sample Folder Name	CPU	Target Board / Reference Platform
Accelent Systems Inc.		
idp_pxa250	PXA250	PXA250 IDP
ADI Engineering		
adi_brh	80200	BRH
adi_evb	80200	80200EVB
adi_coyote	IXP425	Coyote
Intel		
daytona	PXA250	Daytona
dbpxa210	PXA210	DBPXA210 (Lubbock)
dbpxa250	PXA250	DBPXA250 (Lubbock)
greenlake	PXA210	(GreenLake)
iq80310	80200	IQ80310
iq80321	80200	IQ80321
ixdp425	IXP425	IXDP425 (Richfield)
ixp425	IXP425	(Matecumbe-Bahia)
ixdp2400	IXP2400	IXDP2400
Hybus		
x-hyper250B	PXA250	X-Hyper250B

Verified ARM Targets:

Sample Folder Name	CPU	Target Board / Reference Platform
Altera		
excal_922t	ARM922T	Excalibur - EXPA
ARM		
integrator	ARM920T	ARM Integrator
integrator	ARM946E	ARM Integrator
integrator	ARM966E	ARM Integrator
eval_7tdmi	ARM7TDMI	Evaluator 7t
ARM-EPI		
pid_7tdmi	ARM7TDMI	PID with ARM7TDMI Core Module
pid_940t	ARM940T	PID with ARM940T Core Module
ATMEL		
thunder_920t	ARM920T	Thunder AT91RM9200-DK
at91eb40a_7tdmi	ARM7TDMI	AT91EB40A
Cogent		
ep7312	ARM7TDMI	CDK238/EP7312
OKI		
oki_ml671000	ARM7TDMI	ML671000
oki_ml674000	ARM7TDMI	ML674000
oki_ml674k	ARM7TDMI	ML674K/5K
Samsung		
samsung920	ARM920T	SMDK2400X01 (S3C2400X)
Sharp		
sharp_922t	ARM922T	LH7A400
TI		
omap1510	TI925T	OMAP1510 EVM

For MIPS targets these are located in the ...\EPITools\edtm20\samples directory:

Verified MIPS Targets:

Sample Folder Name	CPU	Target Board / Reference Platform
ATI		
ati_x220	Xilleon220	SetTopWonder Xilleon 220
ati_x225	Xilleon225	SetTopWonder Xilleon 225
Atlas		
mti_4kc	MTI4Kc	Atlas Board
Broadcom		
bcm3310a	BCM3310/A	BCM93310 QAMLink with 3310 Rev A
bcm91101	BCM91101	BCM91101 IP Phone
bcm94702cpci	BCM4702	BCM94702CPCI
bcm94710ap	BCM4710	BCM94710AP
bcm96345r	BCM6345	BCM96345R
broadcom	BCM6352	BCM96352SV
broadcom	BCM3352	BCM96352D
broadcom	BCM3360	BCM93360
broadcom	BCM3350	BCM93350CLC QAMLink Cable Modem
broadcom	BCM3345	BCM93345
broadcom	BCM3310	BCM93310 QAMLink with BCM3310 Rev B
broadcom	BCM1100	BCM91100 IP Phone
broadcom	BCM7100	
CARPET		
carpet	PR1900	CARPET Evaluation Board
Centaurus		
centaurus	PR1910	Centaurus Development Board
IDT		
rc323xx	RC32364	79S134 Evaluation Board
rc323xx	RC32332	79S332/334 Evaluation Board
rc323xx	RC32334	79S332/334 Evaluation Board
rc323xx	RC32355	79EB355 Evaluation Board
rc324xx	RC32438	79EB438 Reference Board
LSI		
lsi4102	LSI4102	LR4102
lsi4102	LSI4103	LR4103
lsi4102	LSI4102/3	BDMR4102
Malta		
malta	MTI5Kc	Malta 5Kc
ProMIPS		
promips	PR3940	ProMIPS PR3940 Prototyping System

Appendix D – Menu Descriptions

Flash Device and Image Information

At the top of each menu, Flash Device and Image Information is displayed.

```
Flash Device : 28F640B3B  x16  8MEG
Devices      : 1-Series  1-Parallel  1-Total
Sector Groups:     2
Sector Count :     8    127
Sector Size  :     8K    64K
Base Address : 0x00000000

Filename     : c:\flash\bin\boot.bin
File Length  :    2097152  (0x00200000)
Image Size   :    2097152  (0x00200000)
Image Address: 0x00000000 - 0x001FFFFF
```

In this screen shot, this flash device is a Boot Bottom, 8Meg part with a 16-bit data bus, and has 2 Sector Groups, the 1st Sector Group consists of 8 - 8K blocks, the 2nd Sector Group consists of 127 - 64K blocks. Devices shows that this device has been configured for a single Device on the bus.

```
Flash Device : 28F640B3B  x32  16MEG
Devices      : 1-Series  2-Parallel  2-Total
Sector Groups:     2
Sector Count :     8    127
Sector Size  :    16K   128K
Base Address : 0x00000000

Filename     : c:\flash\bin\boot.bin
File Length  :    2097152  (0x00200000)
Image Size   :    2097152  (0x00200000)
Image Address: 0x00000000 - 0x001FFFFF
```

In this screen shot, Devices shows that this flash device has been configured for a single Device on the bus in Series and 2 Devices on the bus in Parallel. This part is now a Boot Bottom, 16Meg part with a 32-bit data bus, and has 2 Sector Groups, the 1st Sector Group consists of 8 - 16K blocks, the 2nd Sector Group consists of 127 - 128K blocks.

Flash Device

This is the flash device part type to be erased/programmed/verified, the device bus width, and the size in bytes. Flash part types which end in a 'B' or 'T' indicate that the part is a Boot Sector/Block type device, as described in SECTOR GROUPS. The flash device architecture information is also displayed.

Devices

This is the number of flash devices which are on the same bus, and indicates number of devices in Series and in Parallel. The default value is 1, for both Series and Parallel. If the target has more than 1 device on the bus, then the user must enter these values from the Flash Device Menu.

Sector Groups

This is the number of Sector Groups in the flash device. A Sector Group is a contiguous section of sectors/blocks of the same Sector Size. Boot Sector/Block devices will have 2 or more sector groups of different Sector Sizes. Flash part types which end in 'B' are Boot Bottom devices, and those which end in 'T' are Boot Top devices.

Sector Count

This is the number of sectors in a Sector Group of Sector Size. The above screen shot has 2 Sector Groups, the 1st with a Sector Count of 8, the 2nd with a Sector Count of 127.

Sector Size

This is the size of a sector in bytes. For symmetrical Sector/Block devices all sectors/blocks in the flash device will be this size, and will have only one Sector Group. For non-symmetrical (Boot) Sector/Block devices each Sector Group will be of the indicated Sector Size. The above screen shot has 2 Sector Groups, the 1st with a Sector Size of 8K, the 2nd with a Sector Size of 64K.

Base Address

This is the Base Address of the flash device. This is a virtual address!

Filename

This is the current image filename and path.

File Length

This is the length of the current image file.

Image Size

This is the length of the image file to be programmed into flash.

Image Address

This is the Address where the image is to be programmed into flash. This is an offset from the Base Address!

Main Menu

From the MAIN MENU, you may select the Flash Device Type, Erase Menu, Program Menu, and Diagnostic Menu. Other Main menu entries allow you to enter an Image Filename, the Base Address of the Flash Device, the Image Address, and the Image size.

```
MAIN MENU
(1) Flash Type
(2) Image Filename
(3) Base Address
(4) Image Address
(5) Image Size
(6) Erase Menu
(7) Program Menu
(8) Diagnostic Menu
(9) Flash Device Menu

(0) QUIT Program
```

Flash Type

Selecting Flash Type will display a listing of all supported flash types. Next to each flash device listed, is the bus width of the device, and the device size in Kbytes or Mbytes. To select the flash device for your target, enter the number next to the device as shown in the screen shot below. If the flash device on your target is not listed, the source code for the flash utility is provided so that you may add support for your target's flash device, See: Appendix E. Boot Block/Sector device types will end with a 'T' or 'B', for Boot Top or Boot Bottom device.

```
                    FLASH TYPE MENU

 1: Am29LV400BB x8  512K        18: 28F008B3B   x8    1M
 2: Am29LV400BB x16 512K        19: 28F008B3T   x8    1M
 3: Am29LV800BB x8    1M        20: 28F016B3T   x8    2M
 4: Am29LV800BT x8    1M        21: 28F320B3B   x16   4M
 5: Am29LV800BB x16   1M        22: 28F640B3B   x16   8M
 6: Am29LV800BT x16   1M        23: 28F160C3B   x16   2M
 7: Am29F080B   x8    1M        24: 28F160S5    x16   2M
 8: Am29LV160BB x8    2M        25: 28F320J3    x8    4M
 9: Am29LV160BT x8    2M        26: 28F320J3    x16   4M
10: Am29LV160BB x16   2M        27: 28F640J3    x16   8M
11: Am29LV160BT x16   2M        28: 28F128J3    x16  16M
12: Am29LV320DT x16   4M        29: 28F256K3    x16  32M
13: Am29LV641M  x16   8M        30: 28F640W18B  x16   8M
14: Am29LV128M  x16  16M        31: 28F640W18T  x16   8M
15: Am29LV256M  x16  32M        32: AT29LV1024  x16 128K
16: Am29DL163DB x8    2M        33: SST39LF400A x16 512K
17: Am29DL163DB x16   2M
```

Image Filename

This is the file which is to be programmed into the flash and/or verified with the contents already in the device. When entering the filename it must include the complete path. The Flash Utility requires that the file type be binary, S-Record file types are not presently supported.

Base Address

This is the base address of the flash device on the target. The typical base address For ARM and XScale targets is 0x0, for MIPS it is 0xBFC00000. All data reads and writes to the flash device will be an offset of the base address.

NOTE: Some target boot code and/or memory controller initialization command files will remap flash to an address other than the power-on (or reset) address. For Example: The Intel Richfield board powers up with flash at address 0x0, but the EPI ixdp425.cmd command file will initialize the IXP425 memory controller and map RAM to 0x00000000, map Flash to 0x50000000, and leave the flash device write protected. Therefore, the Base Address setting in the flash utility should be set to where the flash is moved to.

For targets with more than one flash device: If the flash device configuration is in parallel, then one base address will address both devices.

If the flash device configuration is in series, then you must enter the number of devices on the target that are in series. For some devices, some operations will work on image size alone. To select the number of devices that are in series, go to the "Flash Device Menu" and select "Devices on Bus (In Series)". To program or erase a single device in a multi-device configuration, select 1 for the number of devices and set the base address to that device, the image size should also be set be no greater then the single device size.

Image Address

This is the address within the flash device where the image will be programmed and/or the address of the sector/block to be erased (when using Erase Image Sector(s)). This address must be relative to the base address. For example, if you have a base address of 0xBFC00000, and a image address of 0x1000, then the Flash Utility will write and/or erase the image starting at address 0xBFC01000.

Image Size

This is the length in bytes of the image to be programmed. Starting with version 2.0.8 of the flash utility, you will be prompted to use the length of the image file, or enter in a value for the Image Size.

```
(4)  Keep Image Size : 0x00100000
(5)  Use File Length : 0x00200000
(6)  Enter Image Size.

Enter 4, 5 or 6:
```

At this prompt, pressing 4 then enter will leave Image Size unchanged, pressing 5 then enter will assign the image file length to the Image Size, pressing 6, then enter will allow you enter a value for Image Size.

The value entered should be less than or equal to the length of the image file. If you enter a value greater than the image file size, you will not get program or verify errors, instead you will be notified with the message:

```
Unexpected End of File.
xxx bytes not Programmed.
```
or
```
xxx bytes not Verified.
```

Entering a value greater than the image file size allows the user to erase the flash device past the end of the image when using either:

```
(2) Erase Image, Program Image, Verify   (In PROGRAM MENU)
```
or
```
(2) Erase Image Sector(s)                (In ERASE MENU)
```

Program Menu

The PROGRAM MENU allows you to Erase, Program, and Verify the file image with the flash memory image.

```
PROGRAM MENU
(1) Program Image
(2) Erase Image,  Program Image, Verify
(3) Erase Device, Program Image, Verify
(4) Verify Image Erased
(5) Verify Image Programmed
(6) Find Programmed Sections
(7) Find Verify Failed Sections
(8) Backup Flash Device (to Disk)
(9) Swap Image Endianness

(0) Return to Main Menu
```

Program Image

This selection will program the flash device with the contents of the Image File. The flash device will be programmed starting at the Base Address + Image Address for a length of Image Size. After the programming operation is completed, the image will be verified against the device.

Erase Image, Program Image, Verify

This selection will do the following operations:

1) Erase those sector(s)/block(s) within the flash device which will be programmed with the image.

2) Verify that the erased image sector(s)/block(s) are erased.

3) Program the flash device with the contents of the Image File.

4) Verify that the image was correctly programmed into the device.

Erase Device, Program Image, Verify

This selection will do the following operations:

1) Erase ALL sectors/blocks within the flash device.

2) Verify that ALL sectors/blocks within the flash device are erased.

3) Program the flash device with the contents of the Image File.

4) Verify that the image was correctly programmed into the device.

Verify Image Erased

This selection will Verify only those memory locations which correspond with the image to be programmed within the flash device are erased.

Verify Image Programmed

This selection will verify the image file contents against the flash device. This operation is independent of the program operation, and may be used to determine if the image in flash is the same as that in the file. If there are failures, only the first and last failed address will be displayed along with the total number of bytes which failed. To get the address of each failed sections, select (7) Find Verify Failed Sections.

Find Programmed Sections

This selection will display those memory locations which are programmed. For more detailed information see: FIND PROGRAMMED SECTIONS in ERASE MENU.

Find Verify Failed Sections

This selection will verify the image file contents against the flash device and display the memory location for each section which failed and the number of bytes that failed in each section.

Backup Flash Device (To Disk)

This selection will write a image file of the flash device to the host computer. The image of the flash device will start at the Base Address + Image Address for a length of Image Size (settings used for programming and erasing the flash device are used). You will be prompted for the image filename with path, if the file exists you will be warned before overwriting.

Swap Image Endianess

This selection will swap the endianness of the image file read from the host. You may select one of three choices, Swap Endian: NONE, Swap Endian: 16-Bit, or Swap Endian: 32-Bit. Most image files will not need to be endian swapped, this feature is used to handle the case where a tool generates a binary image file which is not in the same endianness as that of the target, or in some cases is only endian swapped on 16-bit boundaries. If Swap Image Endian has been selected, the message: "NOTICE: Image Endianness will be xx-bit Swapped", will appear above all menus, where xx-bit will be 16-bit or 32-bit.

NOTE: Swapping image endianness does not convert an image built for a little endian target to an image for a big endian target or visa versa.

Erase Menu

The ERASE MENU allows you to erase the flash device and verify that the device is erased.

```
ERASE MENU
(1) Erase Device
(2) Erase Image Sector(s)
(3) Verify Device Erased
(4) Verify Image Erased
(5) Find Erased Sections
(6) Find Programmed Sections

(0) Return to Main Menu
```

Erase Device

This selection will do, in the following order:

1) Erase ALL sectors/blocks within the flash device.

2) Verify that ALL sectors/blocks within the flash device are erased.

Erase Image Sector(s)

This selection will do the following operations:

1) Erase those sectors (or blocks) within the flash device which will be programmed with the image.

2) Verify that the erased image sector(s)/block(s) are erased.

Verify Device Erased

This selection will verify that ALL sectors/blocks within the flash device are erased.

Verify Image Erased

This selection will Verify only those memory locations which correspond with the image to be programmed within the flash device are erased.

Find Erased Sections

This selection will display those memory locations which are erased, and the length in bytes of each erased section. A section is determined to be erased when a contiguous section of flash is found to have "Erase Section Size" or more bytes erased. The logic for finding Erased Sections uses a 32-bit value read from flash and compared with 0xFFFFFFFF, because of this, the minimum size for "Erase Section Size" is 4, and may be entered as decimal, hex, K or M. The last "Erase Section Size" entered will be displayed and used if no size is entered.

```
Enter Erase Section Size: 256
(Minimum Size to Qualify as a Section)

Find Erased Sections...

Bytes Erased : 0x00000FE4 (4068)
First Address: 0x001AF01C
Last  Address: 0x001AFFFC
Bytes Erased : 0x00001C54 (7252)
First Address: 0x001BE3AC
Last  Address: 0x001BFFFC

Bytes Erased : 0x0002D7B0 (186288)
First Address: 0x001D2850
Last  Address: 0x001FFFFC

Erased Sections Found:  3
```

Find Programmed Sections

This selection will display those memory locations which are programmed (NOT erased), and the length in bytes of each programmed section. A section is determined to be programmed when a contiguous section of flash is found to have "Erase Section Size" or less bytes erased. See FIND ERASED SECTIONS for more information on "Erase Section Size".

```
Enter Erase Section Size: 1024
(Minimum Size to Qualify as a Section) 256

Find Programmed Sections...

Bytes Found  : 0x001AF01C (1765404)
First Address: 0x00000000
Last  Address: 0x001AF018

Bytes Found  : 0x0000E3AC (58284)
First Address: 0x001B0000
Last  Address: 0x001BE3A8

Bytes Found  : 0x00040000 (262144)
First Address: 0x001C0000
Last  Address: 0x001FFFFC

Programmed Sections Found:  3
```

Diagnostic Menu

The DIAGNOSTIC MENU allows you to erase, program, verify, fill a flash device, and repeat the operation any number of times.

```
DIAGNOSTIC MENU
(1) Fill Device
```

```
(2) Fill Sectors
(3) Verify Fill Device
(4) Verify Fill Sectors
(5) Verify Image Programmed
(6) Erase Device, Fill Device, Verify
(7) Erase Device, Fill Sectors, Verify
(8) Erase Image,  Program Image, Verify
(9) Erase Device, Program Image, Verify
(10) Device Fill Value :  0xAA551248
(11) Repeat Count      :  1

(0) Return to Main Menu
```

Fill Device

This selection will do the following operations:

1) Program the entire flash device with the "Device Fill Value".

2) Verify that the device was correctly programmed with the fill value.

3) Repeat steps 1 and 2, "Repeat Count" times. The device will not be erased.

Fill Sectors

This selection will do the following operations:

1) Program all flash device sectors/blocks with the sector/block number. Each sector will be filled with a 2-byte sector number, which starts at 0 for the 1st sector/block, 1 for the 2nd sector/block, ...

2) Verify that each sector/block within the device was correctly programmed with the sector number.

3) Repeat steps 1 and 2, "Repeat Count" times. The device will not be erased.

Verify Device

This selection will do the following operations:

1) Verify that the device was correctly programmed with the fill value.

2) Repeat step 1, "Repeat Count" times. The device will not be erased.

Verify Sectors

This selection will do the following operations:

1) Verify that each sector/block within the device was correctly programmed with the sector number.

2) Repeat step 1, "Repeat Count" times. The device will not be erased.

Verify Image Programmed

This selection will do the following operations:

1) Verify that the image in the flash device is the same as the image file.

2) Repeat step 1, "Repeat Count" times. The device will not be erased.

Erase Device, Fill Device, Verify

This selection will do the following operations:

1) Erase ALL sectors/blocks within the flash device.

2) Verify that ALL sectors/blocks within the flash device are erased.

3) Program the entire flash device with the "Device Fill Value".

4) Verify that the device was correctly programmed with the fill value.

5) Repeat steps 1 through 4, "Repeat Count" times.

Erase Device, Fill Sectors, Verify

This selection will do the following operations:

1) Erase ALL sectors/blocks within the flash device.

2) Verify that ALL sectors/blocks within the flash device are erased.

3) Program all flash device sectors/blocks with the sector/block number. Each sector will be filled with a 2-byte sector number, which starts at 0 for the 1st sector/block, 1 for the 2nd sector/block, ...

4) Verify that each sector/block within the device was correctly programmed with the sector number.

5) Repeat steps 1 through 4, "Repeat Count" times.

Erase Image, Program Image, Verify

This selection will do the following operations:

1) Erase those sector(s)/block(s) within the flash device which will be programmed with the image.

2) Verify that the erased image sector(s)/block(s) are erased.

3) Program the flash device with the contents of the Image File.

4) Verify that the image was correctly programmed into the device.

5) Repeat steps 1 through 4, "Repeat Count" times.

Erase Device, Program Image, Verify

This selection will do the following operations:

1) Erase ALL sectors/blocks within the flash device.

2) Verify that ALL sectors/blocks within the flash device are erased.

3) Program the flash device with the contents of the Image File.

4) Verify that the image was correctly programmed into the device.

5) Repeat steps 1 through 4, "Repeat Count" times.

Device Fill Value

A 32-Bit value used for the Fill Device operation. The default value is: 0xAA551248. Users may enter any value from 0x0 - 0xFFFFFFFF. Any fill value entered which is less than 32-bits will be right justified. Example: A fill value of 0xFFF will be written to flash as 0x00000FFF. With version 2.0.4 or later of the Flash Utility, you may enter values for Fill Value, in decimal, hex, K or M (See: USER INTERFACE)

Repeat Count

The Number of times to repeat any of the Diagnostic Menu operations.

Flash Device Menu

The FLASH DEVICE MENU allows you to configure the flash utility for the number of devices on the target that are in series.

```
FLASH DEVICE MENU
(1) Flash Type
(2) Device Bus Width
(3) Device Boot Type
(4) Devices In Series
(5) Devices In Parallel

(0) Return to Main Menu
```

Flash Type

Selecting Flash Type will display a listing of all supported flash types. See MAIN MENU: FLASH TYPE for more information.

Device Bus Width

This Selection Reserved for Future Version.

Device Boot Type

This Selection Reserved for Future Version.

Devices In Series

If the flash device configuration for your target has multiple devices and they are in series, then you must enter the number of devices on the target which are in series. If your configuration consists of flash devices which are both in series and in parallel, then each parallel set should be counted as one serial device.

Devices In Parallel

If the flash device configuration for your target has multiple devices and they are in parallel, then you must enter the number of devices on the target which are in parallel.

Appendix E – Adding a New Flash Device Type

Adding a new flash device type to EPIFlash can be as easy as adding a single simple entry in the *flash.c* source file. Many flash devices will be able to reuse the existing Device Functions, if the part you are adding can not, then Device Functions for Program and Erase will have to be added. In most cases you will be able to find where an existing Device Function is very similar to the flash device being added, and will require only minor changes to add a new device.

Adding a new device requires that you add or modify a entry in:

```
FLASH_PART_STRUCT flash_parts[ ]
```

Adding a new device MAY require that you add a entry in:

```
FLASH_SECTOR_STRUCT sector_info[ ]
typedef enum SECTOR_INFO_NDX
```

The 3 sections are grouped together in the source code and are preceded by:

```
/* ADD_DEVICE_PART_HERE */
/* ADD_DEVICE_SECTOR_INFO_HERE */
/* ADD_DEVICE_SECTOR_INFO_NDX_HERE */
```

/* ADD_DEVICE_PART_HERE */

To add a new device, add or modify a entry in the `flash_parts[]` array:

```
FLASH_PART_STRUCT flash_parts[] =
{

    //  Flash       Bus  Size in   Sector       Unlock Block
    //  Device     Width  Bytes  Info Index       Function
    {"Am29LV400BB",  8,  _512K, Am29LV400BB,   NO_BLOCK_LOCK,

    //   Erase Sector               Program Device
    //     Function                    Function
        Am_erase_sec_cmd2_8,      Am_program_cmd2_8,  },

};
```

FLASH DEVICE - flash_parts[ ].name

This is the flash device name, we recommend that it be 11 characters or less in length, but may be any length.

BUS WIDTH - flash_parts[ ].width

This is the device bus width, current support is limited to 8 and 16 bits devices. Prior to version 2.0.8, if your target has flash devices in parallel, such as 2x16-bit devices, it would be listed here as 32 bits, with version 2.0.8, any supported 16-bit device can be configured from the Flash Device Menu to be 32-bits (2-x16 devices).

SIZE IN BYTES - flash_parts[ ].size

This is the size of the flash device in bytes. The following constants have been defined for this field: _128K, _256K, _512K, _1MEG, _2MEG, _4MEG, _8MEG, _16MEG, _32MEG, _64MEG, _128MEG.

SECTOR INFO INDEX - flash_parts[ ].sector_info_ndx

This is the index into the FLASH_SECTOR_STRUCT sector_info[ ] array. The enum SECTOR_INFO_NDX, has been defined for this field. Additions may need to be made to both enum SECTOR_INFO_NDX and FLASH_SECTOR_STRUCT sector_info[], any change to one will require a corresponding change to the other. Please See flash.c for more detailed information on this subject.

UNLOCK BLOCK FUNCTION - flash_parts[ ].unlock_blk_op()

This is a function pointer to a function which will unlock all sectors or blocks on the flash device. If the device does not support lock and unlock sector/block, then this pointer should set to NO_LOCK (NULL).

ERASE SECTOR FUNCTION - flash_parts[ ].erase_sec_op()

This is a function pointer to a function which will erase a sector/block on the flash device.

PROGRAM DEVICE FUNCTION - flash_parts[ ].program_op()

This is a function pointer to a function which will program the flash device with a byte write for 8-bit devices, or a word (short) write for 16-bit devices.

The format of the FLASH_PART_STRUCT structure is:

```
typedef struct
{
  char  *name;              // Name of flash part          (Flash Device)
  int   width;              // Device(s) bus width in bits (Bus Width)
  ULONG size;               // Flash size in bytes         (Size in Bytes)
  int   sector_info_ndx;    // Index into sector_info      (Sector Info Index)
  void  (*unlock_blk_op)(); // Function pointer to: (Unlock Block Function)
  void  (*erase_sec_op)();  // Function pointer to: (Erase Sector Function)
  void  (*program_op)();    // Function pointer to: (Program Device Function)
} FLASH_PART_STRUCT;
```

NOTE: To add a Device, NO additions are needed to FLASH_PART_STRUCT;

/* ADD_DEVICE_SECTOR_INFO_NDX_HERE */

To add a new device, add an entry in enum SECTOR_INFO_NDX.

These enums are used to index into sector_info[ ], any additions or deletions to FLASH_SECTOR_STRUCT sector_info[ ], must be reflected here. These enums are intended to be used in the sector_info_ndx field of FLASH_PART_STRUCT flash_parts[ ]. Any additions or deletions to enum SECTOR_INFO_NDX must also be reflected in FLASH_SECTOR_STRUCT sector_info[ ].

```
typedef enum
{
        RESERVED_SECTOR_NDX,    //Logic REQUIRES that this be the FIRST enum.

        //Intel - Symmetrical Block Devices
        i28F160S5,                      //(S3 Series same as S5)
        i28F320S5,
```

```
        i28F320J3,
        i28F640J3,
        i28F128J3,
        i28F256K3,                          //(K18 Series same as K3)

        //Intel - Boot Block Devices
        i28F800B3T,                         //(C3 and W18 Series same as B3)
        i28F800B3B,
        i28F160B3T,
        i28F160B3B,
        i28F320B3T,
        i28F320B3B,
        i28F640B3T,
        i28F640B3B,

        //AMD - Symmetrical Sector Devices
        Am29F080B,
        Am29LV641M,
        Am29LV128M,
        Am29LV256M,

        //AMD - Boot Sector Devices
        Am29LV320DT,
        Am29LV320DB,
        Am29DL163DT,
        Am29DL163DB,
        Am29LV400BT,
        Am29LV400BB,
        Am29LV800BT,
        Am29LV800BB,
        Am29LV160BT,
        Am29LV160BB

} SECTOR_INFO_NDX;
```

/* ADD_DEVICE_SECTOR_INFO_HERE */

The `FLASH_SECTOR_STRUCT sector_info[ ]` structure defines the sector/block count and size(s) for those flash devices that do not have symmetrical sectors/blocks. If the flash device you are adding does not have a sector count and size which matches one already defined in the `sector_info[ ]` structure, then a new entry for the flash device must be added.

sector_info[]:

This structure defines the number of sectors/blocks and the size of the sectors/blocks for each flash device type.

Format of `sector_info[ ]`:

```
[Sector]   [Sector  Sector]   [Sector  Sector]   [Sector  Sector]   ...
[Groups],  [Count   Size  ],  [Count   Size  ],  [Count   Size  ],  ...


FLASH_SECTOR_STRUCT sector_info[] =
{
     { 0,  0, 0 },                          //RESERVED_SECTOR_NDX

     //Intel - Symmetrical Block Devices
     { 1,  32, P2_64K },                    //28F160S5
```

```
     { 1,  64, P2_64K },                    //28F320S5

     { 1,  32, P2_128K },                   //28F320J3
     { 1,  64, P2_128K },                   //28F640J3
     { 1, 128, P2_128K },                   //28F128J3
     { 1, 256, P2_128K },                   //28F256K3

     //Intel - Boot Block Devices
     { 2,  15, P2_64K,  8, P2_8K },         //28F800B3T/008
     { 2,   8, P2_8K,  15, P2_64K },        //28F800B3B/008
     { 2,  31, P2_64K,  8, P2_8K },         //28F160B3T/016
     { 2,   8, P2_8K,  31, P2_64K },        //28F160B3B/016
     { 2,  63, P2_64K,  8, P2_8K },         //28F320B3T
     { 2,   8, P2_8K,  63, P2_64K },        //28F320B3B
     { 2, 127, P2_64K,  8, P2_8K },         //28F640B3T
     { 2,   8, P2_8K, 127, P2_64K },        //28F640B3B

     //AMD - Symmetrical Sector Devices
     { 1,  16, P2_64K },                    //Am29F080B
     { 1, 128, P2_64K },                    //Am29LV641M
     { 1, 256, P2_64K },                    //Am29LV128M
     { 1, 512, P2_64K },                    //Am29LV256M

     //AMD - Boot Sector Devices
     { 2, 63,P2_64K, 8, P2_8K },            //Am29LV320DT
     { 2, 8, P2_8K, 63, P2_64K },           //Am29LV320DB
     { 2, 31, P2_64K, 8, P2_8K },           //Am29DL163DT
     { 2, 8, P2_8K, 31, P2_64K },           //Am29DL163DB
     { 4, 7, P2_64K, 1, P2_32K, 2, P2_8K,  1, P2_16K },      //Am29LV400BT
     { 4, 1, P2_16K, 2, P2_8K,  1, P2_32K, 7, P2_64K },      //Am29LV400BB
     { 4, 15,P2_64K, 1, P2_32K, 2, P2_8K,  1, P2_16K },      //Am29LV800BT
     { 4, 1, P2_16K, 2, P2_8K,  1, P2_32K, 15,P2_64K },      //Am29LV800BB
     { 4, 31,P2_64K, 1, P2_32K, 2, P2_8K,  1, P2_16K },      //Am29LV160BT
     { 4, 1, P2_16K, 2, P2_8K,  1, P2_32K, 31,P2_64K },      //Am29LV160BB
};
```

The format of the FLASH_SECTOR_STRUCT structure is:

```
typedef struct
{
   int sector_count;      // number of contiguous sectors of sector_size
   int sector_size;       // sector size in powers of 2
} SECTOR_STRUCT;


typedef struct
{
  int  sector_groups;    // number SECTOR_STRUCTS ss[] needed to define device
  SECTOR_STRUCT ss[8];
} FLASH_SECTOR_STRUCT;
```

NOTE: To add a Device, NO additions are needed to FLASH_SECTOR_STRUCT or SECTOR_STRUCT.

SECTOR GROUPS - sector_info[ ].sector_groups

This is the number of Sector Groups in the flash device. A Sector Group is a contiguous section of sectors/blocks of the same Sector Size. Boot Sector/Block devices will have 2 or more sector groups of different Sector Sizes. For example: In the above listing of FLASH_SECTOR_STRUCT sector_info[], The Intel 28F008/800B3T device has 2 Sector Groups, the 1st Sector Group consists of 15, 64k blocks, the 2nd Sector Group consists of 8, 8k blocks.

SECTOR COUNT - sector_info[ ].ss[ ].sector_count

This is the number of sectors in a Sector Group of Sector Size. For each Sector Group there must be one Sector Count and one Sector Size.

SECTOR SIZE - sector_info[ ].ss[ ].sector_size

This is the sector size in units of power of 2. The following constants have been defined for this field: P2_8K, P2_16K, P2_32K, P2_64K, P2_128K, P2_256K, P2_512K and P2_1MEG. For symmetrical Sector/Block devices, all sectors/blocks in the flash device will be this size, and will have only one Sector Group. For non-symmetrical (Boot) Sector/Block devices each Sector Group will be of the indicated Sector Size.

Building EPIFlash

A sample *makefile* is provided for rebuilding EPIFlash (and all other sample programs) using the EPI compiler tools CCE-MIPS for MIPS, and the ARM SDT or ADS tools for ARM. You should be able to use any cross compiler package to rebuild EPIFlash. EPIFlash does not need to be built with debug information enabled (unless you need to debug your modifications).

Note: *flash.c* contains single line comments `//`, if your compiler does not support single line comments, please contact support.

Big Endian vs. Little Endian

EPIFlash may be built for either big or little endian. The sample *makefile* provided has provisions for both, and has comments on selecting the building endianess.

ARM vs. MIPS

EPIFlash may be built for either ARM or MIPS architecture (with XScale being a member of the ARM architecture). This will be automatic, and is based on defining ARM or MIPS on the compiler command line, the conditional compilation statements for ARM and MIPS are then compiled in *flash.c*.

NOTE: The EPI Flash Utility cannot be built with Microsoft Platform Builder.