

Adding Flash Devices to the EPPC Flash Programmer

Freescale Semiconductor

Document Number: CWFLASHPGRMWP
Rev. 0
11/2005

CONTENTS

1. Overview	3	9. What Device Is Most Similar	6
2. Device Name	4	10. Creating the New Flash Device Definition	6
3. Manufacturer ID Code and Device ID Codes	4	11. How to Generate a Flash Entry for the Master Configuration File Using a Perl Script	7
4. Number of Sectors	4	12. Summary	9
5. Starting and Ending Address for Each Sector	5		
6. Can the Device Be Chip-Erased	5		
7. Options for Data Bits per Device	5		
8. Number of Flash Devices on Your Target	5		

1. Overview

The CodeWarrior™ EPPC Flash programmer supports a number of Flash devices. The purpose of this document is to describe: 1) how to add a new Flash device to the CodeWarrior Flash programmer and 2) how to generate a Flash entry for the master configuration file (FPDeviceConfig.xml) using a Perl script (flash_entrygen.pl).

All Flash devices are defined in the master configuration file: FPDeviceConfig.xml This master file is located in one of these paths:

- > Metrowerks\CodeWarrior\bin\Plugins\Support\Flash_Programmer\EPPC\ (Windows)
- > Metrowerks/CodeWarrior_x.y/CodeWarrior/CodeWarrior_Plugins/Support/Flash_Programmer/EPPC (Solaris)

To add a new device to the CodeWarrior Flash programmer, you must add a new structured table of Flash device information to the master file. The generic structure of this table is shown in Figure 1.

```
</device>

<device>
<name>NameOfFlashDevice</name>
<manufacturerid>MfgID</manufacturerid>
<chiperase>TRUE or FALSE</chiperase>
<sectorcount>NumberOfSectorsInFlash</sectorcount>
<sectorname>0</sectorname>
<sectorname>1</sectorname>
:
:
<sectorname>LastSectorNumber</sectorname>
<sectorstart>StartAddrOfFirstSector</sectorstart>
<sectorend>EndAddrOfFirstSector</sectorend>
:
:
<sectorstart>StartAddrOfLastSector</sectorstart>
<sectorend>EndAddrOfLastSector</sectorend>
<organizationcount>NumberOfVariations</organizationcount>
<organization>Capacity-BusWidth-NumberOfDevices</organization>
:
:
<organization>Capacity-BusWidth-NumberOfDevices</organization>
<id>DeviceID_ForBusWidth</id>
:
:
<id>DeviceID_ForBusWidth</id>
<algorithm>FlashAlgorithmForVariant</algorithm>
:
:
<algorithm>FlashAlgorithmForVariant</algorithm>
```

FIGURE 1 GENERIC FLASH DEVICE TABLE ENTRY

To add Flash programming support for a new Flash device:

1. Locate the data sheet for the new device.
2. Examine the installed `FPDeviceConfig.xml` file for the most similar definition.
3. Copy and edit the definition to make it conform to the new device.

Before you can add a new entry to this file you need to know the following about your Flash device:

- > Device name
- > Manufacturer ID code
- > Device ID codes (8-bit, 16-bit)
- > Number of sectors
- > Starting and ending address for each sector
- > Can the device be chip-erased
- > Options for data bits per device (8-bits, 16-bits)
- > Number of Flash devices on your target
- > What device is most similar in the `FPDeviceConfig.xml` file

Most of this information can be found in your device's data sheet or your target's schematics, however, choosing a similar device involves some research.

2. Device Name

```
<name>NameOfFlashDevice</name>
```

This is a free-form text field that describes the Flash device, taken directly from the data sheet. Use only displayable ASCII characters with no spaces. Some examples, found in the `FPDeviceConfig.xml` file are: AM29LV128ML, AM29LV040B-HAWK and IN28F256K18.

3. Manufacturer ID Code and Device ID Codes

```
<manufacturerid>MfgID</manufacturerid>
```

```
<id>DeviceID_ForBusWidth</id>
```

These values are read from the Flash device after a specific sequence of writes to the Flash device. The data sheet lists both of these values, though only the Device ID varies among the Flash devices. The Manufacturer ID remains the same for all devices from a given vendor. If the Flash device supports more than one bus width (8-bit, 16-bit), then it might have different Device ID codes for each mode, as is the case for the MBM29DL640E.

4. Number of Sectors

```
<sectorcount>NumberOfSectorsInFlash</sectorcount>
```

This information is listed in the data sheet. If the data sheet lists sector maps and tables for both 8-bit and 16-bit options, use the 8-bit data. CodeWarrior's Flash programming algorithms require byte-level addresses for each sector. This constraint greatly simplifies the design of the CodeWarrior's Flash programming interface for several data-bus configurations and sizes. For non-8-bit devices, create an 8-bit sector map, if one is not provided in the data sheet.

5. Starting and Ending Address for Each Sector

```
<sectorstart>StartAddrOfSector</sectorstart>
```

```
<sectorend>EndAddrOfSector</sectorend>
```

The data sheet lists the sector maps for the Flash device. If the data sheet lists sector maps for both 8-bit and 16-bit options, use the 8-bit map. If not, then multiply each sector address to obtain its 8-bit equivalent. Figure 2 shows an example of converting a 16-bit sector map to an 8-bit map.

16-bit Sector Map (64K word sectors)	8-bit Sector Map (128 KB sectors)
000000..00FFFF	000000..01FFFF
010000..01FFFF	020000..03FFFF
020000..02FFFF	040000..05FFFF
030000..03FFFF	060000..07FFFF

FIGURE 2 CONVERTING 16-BIT SECTOR MAP TO 8-BIT SECTOR MAP

Newer Flash devices typically have uniform sector sizes, but older parts may have sectors of different sizes within the same device. Thus, make sure each sector in the new `FPDeviceConfig.xml` table entry is the correct size.

6. Can the Device Be Chip-Erased

```
<chiperase>TRUE or FALSE</chiperase>
```

Some devices can be completely erased with one chip-erase command, and this is much faster than erasing sector-by-sector. If your Flash device supports this feature, set this value to TRUE.

7. Options for Data Bits per Device

Many Flash devices can be set to use either 8 data bits or 16 data bits depending on the status of a configuration pin on each device (typically named *BYTE#*.) This part of the Flash definition is used in the `<organization>` field described in the next section. Your target uses only one configuration so you need to support only that configuration, but it is good design practice to expand your new definition to include the other configurations for this device.

8. Number of Flash Devices on Your Target

```
<organization>Capacity-BusWidth-NumberOfDevices</organization>
```

Your target may use one, two or four devices at the same base address to support an 8-bit, 16-bit, 32-bit or 64-bit data bus. For example, two 8-bit Flash devices side-by-side support a 16-bit data bus, and four 16-bit devices support a 64-bit data bus. The `<organization>` field summarizes each possible combination of device-capacity, bus width and number of devices used. For example, 4Mx16x1 reads as "4MegHalfwords by 16 data bits per device by 1 Flash device", resulting in a total of 4M 16-bit halfwords. Similarly, 1Mx8x4 reads as "1MegaByte by 8 data bits per device by 4 Flash devices", resulting in 1M 32-bit words and a 32-bit data bus presented to the processor.

9. What Device Is Most Similar

To choose a device most similar to one in the master configuration file:

1. Start with the data sheet for the Flash devices on your target and determine the bus width for each device (8 data bits or 16 data bits).
2. Read through the `FPDeviceConfig.xml` file in your CodeWarrior Flash programmer installation and scan for other devices from the same manufacturer with similar part names. For example, an AM29LV640D is similar to the AM29LV641DU, and an IN28F128J3 is similar to the IN28F640J3. Manufacturers often base new designs on the architecture of previous designs to ensure that the new devices are virtually the same as the previous device. However, the new devices may have greater capacity or improved programming features, such as timing and operation. This pattern simplifies Flash programming because the Flash programming algorithms remain unchanged, while only the device names, sectors and Device IDs change.
3. Open the `FPDeviceConfig.xml` file in a text editor, and compare the entries of the IN28F640J3 and IN28F128J3. For example, see how the latter was built as an extension of the former. Note also, how the part number of your device may be just a revision letter different from a defined part. For example, the Flash programmer considers the AM29DL640B to be the same as the AM29DL640D and the AM29DL640G. Thus, if you use a part number like this, just tell the Flash programmer you are using the defined part and do not need to create a new device table entry.

10. Creating the New Flash Device Definition

If you find a likely candidate on which to base your new Flash configuration table:

1. Copy the whole table entry, everything between `<comment>`, through `<name>/<sectorname>/<algorithm>/` etc. ... to the following `<comment>`.
2. Paste this table entry either just past the original entry or at the end of the file.
3. Change the `<name>` field to the part name of your device, making sure to use no spaces in the name.
4. Edit the remaining fields as appropriate according to the data sheet for your device, starting with the `<id>` values.

If you do not find a similar device already defined in `FPDeviceConfig.xml` file:

1. Create an entirely new definition using Figure 1 as the template and existing definitions as guidelines.
2. Use the information discussed above to define all the fields for your new entry.
3. Define the `<algorithm>` field by choosing which of the available Flash programming algorithms best works with your device. To do this, it helps to understand the file-naming convention for these algorithms, as described below.

Algorithm file names are created by combining these fields: manufacturer, data-bits-per-device, number-of-flash-devices and optional.

For example, the Flash algorithm for a single Intel 28F640J3 device, used in its 8-bit configuration (BYTE# = 0), is `intel18x1j3.elf` (Intel + 8-bits-per-device + 1 device + J3 variant).

Similarly, the Flash algorithm for two AMD 29LV320MB devices, used in their 16-bit configurations (BYTE# = 1), is `amd16x2.elf`.

4. Define the `<option>` field, which may be the most confusing.

For example, Intel has J3 and C3 variants of their devices, and AMD has two methods of Flash programming, with the alt method as their variant. The J3 and C3 designators indicate the type of devices being programmed and reference the J or C in the device name's suffix.

The alt method indicates the use of AMD's alternative method of programming devices, used when devices that support BYTE and WORD modes are used in BYTE mode. The differences between these two methods are the Flash addresses to which the processor writes its Flash access commands. These differences are summarized in Figure 3, where ##### is the generic placeholder for the starting address of the Flash devices (Flash base address).

	Standard	Alternate
8-bit	0x###00555	0x###00aaa
	0x###002aa	0x###00555
	0x###00555	0x###00aaa
16-bit	0x###00aaa	0x###01554
	0x###00554	0x###00aaa
	0x###00aaa	0x###01554

FIGURE 3 DIFFERENCES BETWEEN AMD'S TWO METHODS OF FLASH PROGRAMMING

Another variant of the Intel algorithms is the collection of files with `wbe` in their filenames. This acronym stands for With Block Erase, meaning the algorithm erases each Flash sector before programming it. The `wbe30` algorithms, if supported, erase the whole device before programming.

11. How to Generate a Flash Entry for the Master Configuration File Using a Perl Script

Before you can generate a Flash entry for the master configuration file `FPDeviceConfig.xml` using the Perl script `flash_entrygen.pl`, you must have the following:

- > Flash device data sheet
- > `flash_entrygen.pl` script
- > A Perl installation

To generate a Flash entry for `FPDeviceConfig.xml` using `flash_entrygen.pl`:

1. Edit the script.

First fill in the Flash device parameters. These are Perl variables that you can find in the section at the end of the `flash_entrygen.pl` script. As an example, the script is already filled in, and adapting it to a new Flash is straightforward. The Flash parameters significance is as follows:

```
$_dev_name = "MBM29DL164TE";
```

This string represents the Flash device name, as it will appear in the Flash programmer's device list. Replace the name between quotes with the actual name inside the Flash data sheet.

```
$_manufacturer_id = 0x04; Fujitsu
```

The `0x04` value represents the manufacturer's code, and it is necessary for the Flash device identification. You can find this number inside the Flash data sheet. Some of the most common codes are:

```
0x01=AMD, 0x89=Intel, 0x04=Fujitsu, 0x20=ST, 0xBF=SST, 0x1F=Atmel
```

```
$_chip_erase = "TRUE";
```

This parameter tells if the Flash device supports the chip-erase command. If you can find this command in the supported commands table inside the Flash data sheet, set it to `TRUE`. Otherwise, set it to `FALSE`.

```
$_sect_count = 39;
```

This is the number of sectors contained by the Flash device. You can find this value inside the Flash data sheet along with a sector map.

```
@_sector_map = (
(0x10000, 31),
(0x2000, 8)
);
```

This list contains the sector organization of the Flash device. Each entry represents a sector group of equally sized sectors. The first value is the size of the sector. It is given as a hex number representing the number of bytes in the sector. The second value represents the number of sectors in this group. For instance, the above example shows a top-boot Flash device. It contains 31 sectors of 64 KB each at lower addresses and 8 sectors of 8 KB each at higher addresses.

Each Flash data sheet contains a sector table, which will help you retrieve the above values and their order in the list.

```
$_org_count = 6;
```

This represents the number of Flash configurations listed below. We support organizations up to 64-bit data bus per bank (i.e., a maximum of 16x4 or 32x2). Although, we do not yet have the 32-bit algorithms.

```
@_flash_org = (

("2Mx8x1", "33", "amd8x1alt.elf"),
("2Mx8x2", "33", "amd8x2alt.elf"),
("2Mx8x4", "33", "amd8x4alt.elf"),
("1Mx16x1", "2233", "amd16x1.elf"),
("1Mx16x2", "2233", "amd16x2.elf"),
("1Mx16x4", "2233", "amd16x4.elf")

);
```

Each entry in this list corresponds to a Flash organization. The first value is the organization itself. For instance, the 1Mx16x2 string represents 2 x 2 Megabyte chips organized as 1 Megabyte x 16 bits data-bus. The second number is the device ID. You can find this number inside the data sheet. For 8-bit devices, only the lowest byte of this number counts. The last string in the entry represents the name of the algorithm used to program or erase the Flash device. Its intuitive name is given as a combination of the manufacturer's name and the Flash organization.

2. Run the script.

At the command line, type:

```
perl flash_entrygen.pl > "device_name.txt"
```

Perl should be in the system path. After the script finishes successfully, the `device_name.txt` file will contain a Flash entry to add to `FPDeviceConfig.xml`.

3. Add the generated Flash entry to `FPDeviceConfig.xml`.

Tips:

- > Check the XML syntax after adding the device to `FPDeviceConfig.xml` using Internet Explorer.
- > Check to see if the generated Flash organization is correct. One way to do this is by replacing the `FPDeviceConfig.xml` in the CodeWarrior software and then selecting the new Flash device in the Flash programmer's panel to check its bounds. For instance, an 8 MB Flash organization (whether it is 4Mx16x1, 2Mx8x4, etc.) should take up 8 MB memory space. If the Flash base address is 0xFF800000, the last sector in the Flash organization should end at 0xFFFFFFFF (because 0x800000 is 8 MB).

12. Summary

Freescall is continually adding new Flash device definitions to the CodeWarrior Flash programmer. However, with this on-going effort, the Flash devices on your target may not be listed in your CodeWarrior software. If you follow the steps outlined in this document: 1) you can expand the CodeWarrior Flash programmer to include any devices you need and 2) generate a Flash entry for the master configuration file using a Perl script.

How to Reach Us:

Home Page:

www.freescale.com

e-mail:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor

Technical Information Center, CH370

1300 N. Alma School Road

Chandler, Arizona 85224

1-800-521-6274

480-768-2130

support@freescale.com

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH

Technical Information Center

Schatzbogen 7

81829 Muenchen, Germany

+44 1296 380 456 (English)

+46 8 52200080 (English)

+49 89 92103 559 (German)

+33 1 69 35 48 48 (French)

support@freescale.com

Japan:

Freescale Semiconductor Japan Ltd.

Headquarters

ARCO Tower 15F

1-8-1, Shimo-Meguro, Meguro-ku,

Tokyo 153-0064, Japan

0120 191014

+81 3 5437 9125

support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.

Technical Information Center

2 Dai King Street

Tai Po Industrial Estate,

Tai Po, N.T., Hong Kong

+800 2666 8080

support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor

Literature Distribution Center

P.O. Box 5405

Denver, Colorado 80217

1-800-441-2447

303-675-2140

Fax: 303-675-2150

LDCForFreescaleSemiconductor

@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright license granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.