

# Freescale Semiconductor

**Application Note** 

Document Number: AN3389

# **Creating an External Flash Algorithm**

by Ildar Saifutdinov and Sebastien Duchamp

# 1 Introduction

This document explains how to use a Flash Tool Kit to support additional flash devices on the ColdFire<sup>®</sup> CodeWarrior<sup>™</sup> Flash Programmer by creating new programming algorithms. This document:

- Helps you confirm whether a new flash algorithm is necessary
- Provides instructions
- Provides an example project

# 2 Preliminary Background

Before you program or erase any flash device, you must ensure the CPU can access it. For example, you might need a different debug setup that requires modifications to the debugger configuration file. Consider the following before you begin:

• Read the flash device ID to verify correct connection and programmability. *Application Note 2980 Troubleshooting the Flash Programmer* provides instructions.

# Contents

1	Introduction 1
2	Preliminary Background 1
3	Flash Tool Kit Overview 2
4	Flash Programmer API 4
5	Creating a New Flash Programming Algorithm
6	Flash Programming Examples 21
7	Chip Makers' Flash Programming Recommendations



© Freescale Semiconductor, Inc., 2006-2008. All rights reserved.



Flash Tool Kit Overview

- Note that many manufacturers use the same flash-device algorithms, so it is likely that flashes can be programmed using algorithms included with CodeWarrior software. In addition, many manufacturers produce devices compatible with Intel or AMD.
- Check whether a new flash device can be programmed with the same algorithms that Intel or AMD use, as described in Section 7.
- Refer to the *Application Note 3390 Adding Device(s) to the Flash Programmer* to determine if a flash device is programmable with an algorithm already included with the CodeWarrior software.
- Follow the steps in Section 5 if the flash device cannot be programmed with an existing algorithm.

# 3 Flash Tool Kit Overview

The Flash Took Kit described in this document helps you develop flash programming algorithms for the CodeWarrior Flash Programmer, as shown in Figure 1. This section and Section 4 provide important information needed before you begin creating a flash programming algorithm.

### Figure 1. Flash Tool Kit

	Flash_ToolKit.mcp		
	😢 Flash Algo Development 🔽 🏥	<b>∀ </b>	<b>\$</b>
	Files Link Order Targets		
	💉 File	Code	Data 😢 🕊 🚊
	📓 flash info.txt	n/a	n/a• 🔳 📥
	🖗 🖃 🥽 SDK Common Files - No Modification 🗋	60	0 • • 🖬
	flash_algorithm.lcf	n/a	n/a• 🔟
"Common" Bort	Ilash_commands.h	U 0	U • <u>N</u>
Common Fait	exit c	12	
	flash start.c	48	0 • • 🖬
	V III flash_main_c	0	0 • • 🖬
	🖉 🖃 User Files - Implement Algo here	48	0 • • 🖬
III I a sell Doort	ingo_impl.c	48	0 • • 🖬
"User" Part –	<ul> <li>Eigen User Lests - Implement Algo test</li> <li>Implement Algo test</li> </ul>	288	1028 • • •
	Idsn_uevice.n	288	1028 • • •
		200	
			<b>•</b>
	10 files	396	1028



### 3.1 Flash Tool Kit General Structure

The flash programmer Flash Tool Kit (FTK) application is divided into four different sets of files:

- 1. FTK Common Files (No Modification Needed) contain initialization and other files. This component is common for any flash device and you should not change it while developing the new flash programming algorithm. It consists of the following files:
  - flash\_algorithm.lcf file linker command file. This linker command file is set up according to the rules for flash programming applet allocation in physical memory.
  - flash\_commands.h header file with API to CodeWarrior Flash Programmer commands definition
  - generic.h-header file with the generic data structures and definitions used by the flash programming algorithms
  - exit.c-exit point for the flash programming applet
  - \_\_\_flash\_start.c flash programmer start-up initialization file
  - flash\_main.c main function and API to the CodeWarrior Flash Programmer
- 2. User Files (Implement Algo) contain flash device specific files. This component is modified for any flash devices depending on the flash programming algorithm to be used. It consists of the following files:
  - algo\_impl.c functions to implement for the flash device flash algorithm, such as ID, erase\_sector, erase\_chip, write
- 3. User Files (Implement Algo Tests) contain flash device specific files. This component is considered to be modified for any flash devices depending on the flash programming algorithm to be used. It comprises the following files:
  - flash\_test.c- sample code with the flash unit test functionality implementation
  - flash\_device.h-custom flash device definition file
- 4. flash\_info.txt file. This file contains CodeWarrior Flash Programmer commands description.

To create the new algorithm for flash programming, make all changes to algo\_impl.c (flash device algorithm implementation) and flash\_device.h/flash\_test.c files (flash device tests).

### 3.2 Flash Tool Kit Build Targets

Several build targets are predefined in the Flash Tool Kit (FTK):

- Flash Algo Development flash algorithm development and test application. The **ELF** executable file, created in Flash Algo Development, should be used to develop, debug, and test the new CodeWarrior Flash Programmer algorithm.
- Flash Algo Release create flash algorithm applet. CodeWarrior Flash Programmer uses the **ELF** executable file, created in Flash Algo Release. This build target shares the flash device algorithm with the Flash Algo Development build target; it differs, however, because it cannot be debugged or tested (Figure 2.)



Flash Programmer API

### Figure 2. Flash Tool Kit Targets



# 4 Flash Programmer API

The CodeWarrior Flash Programmer communicates with the flash programming algorithm applet through four different commands:

- get ID
- erase sector
- erase chip
- program

The CodeWarrior Flash Programmer uses an exchange zone in target memory to communicate with the flash applet. The Flash Programmer Target Configuration specifies the Target Memory Buffer; the exchange zone is at the start of this buffer, as shown in Figure 3.



Flash Programmer		×
Flash Programmer Target Configuration Flash Configuration Erase / Blank Check Program / Verify Checksum	Target Configuration         Default Project:       Flash_ToolKit.mcp         Default Target:       Flash Algo Development         I Use Custom Settings         Processor Family:       All         Target Processor:       Generic         I use Target Initialization         C:\Program Files\Freescale\CodeWarrior for ColdFire V7.0\ColdFire_Support\Initia         Browse         Target RAM Memory Buffer         I arget Memory Buffer         Options         I arget Memory Buffer Size:         0x         00006000	
	Show Log Load Settings Save Settings Close	

Figure 3. Target Configuration Buffer Memory Area Start Address

In this application note, scratchMemStart is the starting address of this zone.

Depending on the actions the Flash Programmer requires of the applet, these exchange zone settings may differ.

### Parameter\_block\_t Structure

On the flash applet side, the commands from the CodeWarrior Flash Programmer go through the Parameter\_block\_t structure, mapped in memory, starting from the scratchMemStart address.

All commands from CodeWarrior Flash Programmer are already encoded in flash\_main.c file. This file can be used for the new flash programming algorithm without changes. After loading the flash applet to the target board, CodeWarrior Flash Programmer writes the startMemScratch address in the D7 register (Listing 1).

#### Listing 1. Parameter\_block\_t Pointer Initialization

```
void main(void)
{
    unsigned long num_errors;
    parameter_block_t *_params;
    long res=0;
#ifdef FLASH_ALGO TEST
```



Flash Programmer API

```
int testnumber = 0;
_params = (parameter_block_t *)(unsigned int)&data_1;
#else
    asm
    {
        move.l D7,res
    }
_params = (parameter_block_t *)res;
```

For the detailed description of the Parameter\_block\_t structure refer to Listing 2.

### Listing 2. Parameter\_block\_t Structure Details

```
typedef struct pb {
  unsigned long function;  /* What function to perform ? */
  pointer_t base_addr;  /* where are we going to operate */
  unsigned long num_items;  /* number of items */
  unsigned long result_status;
  pointer_t items;
  } parameter_block_t;
```

Listing 2 definitions:

- function command from CodeWarrior Flash Programmer to be executed
- base\_addr start address of the flash memory
- num\_items number of the data to be transferred from CodeWarrior Flash Programmer to the flash programming applet
- result\_status status of the command; through this field, the flash programming applet notifies CodeWarrior Flash Programmer about the status of the command being executed
- items start address of the data to be transferred from CodeWarrior Flash Programmer to the flash programming applet

ID

The CodeWarrior Flash Programmer uses the getting chip ID command right after the flash algorithm is loaded to the memory buffer to check if the applet runs. For the ID command, CodeWarrior Flash Programmer:

- loads the flash programming applet to the target board.
- sets the command ID, as shown in the function field of Listing 2.
- runs flash programming applet.
- waits while flash applet stops execution.
- checks the status of the command being executed, as shown in the result\_status field of Listing 2.



### fEraseChip

The full chip erase command is called by CodeWarrior Flash Programmer when a full chip erase is performed. For the fEraseChip command, CodeWarrior Flash Programmer:

- loads the flash programming applet to the target board.
- sets the command fEraseChip, as shown in the function field of Listing 2.
- runs the flash programming applet.
- waits while the flash applet stops execution.
- checks the status of the command being executed, as shown in the result\_status field of Listing 2.

**NOTE** Some flash devices do not support the full chip erase command. Check the flash device's specifications, available from the manufacturer.

### fEraseSector

The sector erase command is called by the CodeWarrior Flash Programmer when a set of sectors in flash memory must be erased. For the fEraseSector command, CodeWarrior Flash Programmer:

- loads the flash programming applet to the target board.
- sets the command fEraseSector. as shown in the function field of Listing 2.
- specifies number of blocks to be erased, as shown in the num\_items field of Listing 2.
- specifies start-up address of each block to be erased, as shown in the items field of Listing 2.
- runs the flash programming applet.
- waits while the flash applet stops execution.
- checks the status of the command being executed, as shown in the result\_status field of Listing 2.

### fWrite

The fWrite program buffer command is called by the flash programmer to program a set of values at a specific address. For the fEraseSector command, CodeWarrior Flash Programmer:

- loads the flash programming applet to the target board.
- sets the command fWrite, as shown in the function field of Listing 2.
- specifies number of bytes to be programmed, as shown in the num\_items field of Listing 2.
- specifies start-up address of data to be programmed, as shown in the items field of Listing 2.

- runs flash programming applet.
- waits while flash applet stops execution.
- checks the status of the command being executed, as shown in the result\_status field of Listing 2.

In this section, step-by-step instructions show you how to use the Flash Tool Kit to create a new CodeWarrior Flash Programmer flash programming algorithm for a flash device not integrally supported by the CodeWarrior software.

- 1. Store an original version of the Flash Tool Kit files from the CodeWarrior delivery.
- 2. Copy FlashToolKitTemplate and Common\_Files folders to a different working location.
- 3. Check the folder: Freescale\CodeWarrior for ColdFire V7.x\ColdFire\_Tools\FlashToolKit
- 4. Open Flash Tool Kit project
  - a) Locate the Flash Tool Kit project named Flash\_ToolKit.mcp in the FlashToolKitTemplate folder.
  - b) Open Flash\_ToolKit.mcp project with the CodeWarrior for ColdFire Development Studio as shown in Figure 4.
  - c) Upon being opened, Flash Tool Kit will appear as shown in Figure 1.
  - d) Ensure the Flash Algo Development build target is selected.
- 5. Correct derivative family ensure correct **Target Processor** is used in the **Debugger Settings Target Settings Panel**.
- 6. Configuration and memory files ensure the correct mem and cfg file are used for the connected hardware in the **Debugger Settings Target Settings Panel**.
  - a) For supported Freescale Evaluation Boards, you can use the debugger config files (\*.cfg), and the debugger mem files (\*.mem) available with the CodeWarrior Development Studio. Check the folder:

```
Freescale\CodeWarrior for ColdFire V7.x
\ColdFire_Support\Initialization_Files
```

b) For example, the configuration settings for the MCF5282EVB board are shown in Figure 5.



Figure 4. Flash Tool Kit Project Opening

Open 🥐 🔀
Look in: 🧀 Flash Tool Kit Template 💽 🔶 🛅 🔻
<ul> <li>in</li> <li>Common_Files</li> <li>Flash_ToolKit_Data</li> <li>algo_impl.c</li> <li>flash_info.txt</li> <li>Flash_ToolKit.mcp</li> </ul>
Object name: Rash_ToolKit.mcp Open
Objects of All Files (*.*) Cancel

Figure 5. Hardware Configuration Settings

Flash Algo Developmen	it Settings [Flash_ToolKit.mcp]
Target Settings Panels	CF Debugger Settings
	Target Processor:       5282       Target OS; BareBoard         Image: Use Target Initialization File         {Compiler}ColdFire_Support\Initialization_Files\m5282evb.cfg       Browse         Image: Use Memory Configuration File       [Compiler]ColdFire_Support\Initialization_Files\m5282evb.mem       Browse
Cold In Call Incer     Custom Keywords     Custom Keywords     Debugger     Debugger Settings     GF Debugger Setti     CF Interrupt     Debugger PIC Setti     Source Folder Map ▼	Program Download Options         Initial Launch       Successive Runs         Executable       Image: Constant Data         Constant Data       Initialized Data         Initialized Data       Initialized Data         Uninitialized Data       Initialized Data
	Factory Settings         Revert         Import Panel         Export Panel
	OK Cancel Apply

- **NOTE** Where custom hardware design exists, the debugger configuration file and the memory mapping file must be written. In this case, the memory initialization for the flash device should be checked before trying to create the new flash programming algorithm. Refer to the information under Section 2.
- 7. Specify an alternate loading address.
  - a) The flash algorithm, a PIC\PID application, can run from anywhere in memory. An alternate loading address is where the flash applet code will be loaded and executed on the target board. This address can be either in internal or external RAM memory, as shown in Figure 6.



Figure 6. Alternate Loading Address Settings

📕 Flash Algo Developmen	t Settings [Flash_ToolKit.mcp]	? 🗙
Target Settings Panels	Debugger PIC Settings	
<ul> <li>C/C++ Preprocessor</li> <li>C/C++ Warnings</li> <li>ColdFire Assembler</li> <li>Code Generation</li> <li>ColdFire Processor</li> <li>Global Optimizations</li> <li>Linker</li> <li>ELF Disassembler</li> <li>ColdFire Linker</li> <li>Editor</li> <li>Custom Keywords</li> <li>Debugger</li> <li>Debugger Settings</li> <li>Remote Debugging</li> <li>CF Debugger Setti</li> <li>CF Interrupt</li> <li>Debugger PIC Setti</li> <li>Source Folder Map</li> </ul>	Alternate Load Add       0x20000500	
	Factory Settings         Revert         Import Panel         Export F	Panel
	OK Cancel A	pply

- b) The Alternate Load Address should match the address where the code is linked. By default, the flash programming algorithm will be compiled to start at address 0x500; refer to the TEXT start address value in the flash\_algorithm.lcg linker command file as shown in Figure 7.
- c) If the custom board's address space where you want to debug the applet is other than 0x0, the alternate address must be changed. For example: if RAM is allocated starting from address 0x2000000, the alternate loading address will be 0x2000000+0x500 = 0x20000500.

#### Figure 7. Code Start Address Definition in flash\_algorithm.lcf File

#	Sample Linker Command File for CodeWarrior for ColdFire
# # # # # # # # # #	NOTE: The debugger uses the Illegal Instruction Vector to stop. A small subroutine is written at the location VBR+0x408-VBR+0x40B to handle the exception. The Illegal Instruction Vector in the the vector table at VBR+0x10 is then pointed to it. When the debugger encounters an illegal instruction, it jumps to this subroutine, which ends with an RTE, then exits. Do not overwrite this area of memory otherwise the debugger may not exit properly.
м }	EMORY ( TEXT (RX) : ORIGIN = <mark>0x00000500</mark> , LENGTH = 0 # using External DRAM DATA (RW) : ORIGIN = AFTER(TEXT), LENGTH = 0

**NOTE** The value of scratchMemstart address is set from the user interface.



Figure 8. Code in the Flash\_Device.h File



- 8. Modify algo\_impl.c file
  - a) The flash algorithm functionality file algo\_impl.c shall be modified and filled with the correct programming commands, as recommended by the flash device manufacturer.
- 9. Modify ID function in algo\_impl.c file
  - a) By default, the ID function in algo\_impl.c file looks as shown in Listing 3.

Listing 3. ID Function Template in algo\_impl.c File

```
retval_t ID(parameter_block_t *p_pb)
{
    retval_t result = 0;
    volatile unsigned long* item_addr = (p_pb->items).l;
    /* Add code: the correct access size depending on the bus must be used for the base_addr */
    volatile unsigned short *base_addr = (p_pb->base_addr).w;
    /* Add code: first of all reset the device.
    The fID is not called in the new flash programmer plugin therefore
    the flash chip must always be bring into the read state.
    */
    /* Add code: read the device ID */
    /* we currently assume that we have the right value */
    /* anyway, the IDE have to care about the flash ID and compare with the xml file */
    return result;
}
```

- b) The following definitions pertain to Listing 3:
  - parameter\_block\_t \*p\_pb pointer to the parameter\_block\_t structure to be passed to the ID function

- retval\_t result of the function execution
- c) The correct command sequence should be created for the ID function based on the recommendations of the flash device manufacturer, as described in Section 6.1.1.
- 10. Modify erase\_sector Function
  - a) By default, the erase\_sector function in algo\_impl.c file appears as shown in Listing 4.

Listing 4. Function Template erase\_sector in algo\_impl.c

```
retval_t erase_sector(parameter_block_t *p_pb, unsigned long sect_index)
{
    int timed_out, got_it;
    retval_t result = 0;
   /* Add code: the correct access size depending on the bus must be used for the base_addr */
   volatile unsigned short *base_addr = ((unsigned short **)(p_pb->items).w)[sect_index];
    /* Add code: first of all reset the device.
    The fID is not called in the new flash programmer plugin therefore
    the flash chip must always be bring into the read state.
    * /
    /* Add code: erase one sector */
    /* Add code: wait for status */
    /* Add code: handle error (and timeout if needed) */
    /* Add code: put back the flash in read state */
    return result;
}
```

b) Listing 4 definitions:

- parameter\_block\_t \*p\_pb pointer to the parameter\_block\_t structure to be passed to the erase\_sector function
- unsigned long sect\_index index of the sector to be erased
- retval\_t result of the function execution
- c) Based on recommendations from the flash device manufacturer, the correct command sequence must be created for flash-sector erasing, as described in Section 6.1.2.
- 11. Modify erase\_chip Function
  - a) By default, the erase\_chip function in algo\_impl.c file looks as presented in Listing 5.

#### Listing 5. Function Template erase\_chip in algo\_impl.c File

```
retval_t erase_chip(parameter_block_t *p_pb)
{
    int errors = 0;
    retval_t result = 0;
    unsigned short stat;
    int got_it;
    /* Add code: the correct access size depending on the bus must be used for the base_addr */
    volatile unsigned short *base_addr = (p_pb->base_addr).w;
    /* Add code: first of all reset the device.
    The fID is not called in the new flash programmer plugin therefore
```





3

```
the flash chip must always be bring into the read state.
*/
/* Add code: erase one sector */
/* Add code: wait for status */
/* Add code: handle error (and timeout if needed) */
/* Add code: put back the flash in read state */
return result;
```

```
b) Listing 5 definitions:
```

- i) parameter\_block\_t \*p\_pb pointer to the parameter\_block\_t structure to be passed to the erase\_chip function
- ii) retval\_t result of the function execution
- iii) Create the correct command sequence for full-flash chip erasing based upon recommendations from the flash device manufacturer, as shown in Section 6.1.3.
- 12. Modify write function
  - a) By default, the write function in algo\_impl.c file looks as it appears in Listing 6.

```
Listing 6. Function Template write in algo_impl.c File
```

```
retval_t write(parameter_block_t *p_pb)
{
    int timed_out, got_it;
    unsigned long i;
    unsigned short stat;
   retval_t errors = 0;
   /* Add code: the correct access size depending on the bus must be used for the base_addr */
    volatile unsigned short *base_addr = (p_pb->base_addr).w;
    /* Add code: first of all reset the device.
    The fID is not called in the new flash programmer plugin therefore
    the flash chip must always be bring into the read state.
    */
    /* Add code: program the bytes pointed in the buffer : p_pb->items,
        they are p_pb->num_items bytes
        handle error (and timeout if needed) for each of the program sequence
    * /
    /* Add code: put back the flash in read state */
```

return errors;

b) Listing 6 definitions:

- i) parameter\_block\_t \*p\_pb pointer to the parameter\_block\_t structure to be passed to the write function
- ii) retval\_t result of the function execution
- c) Create the correct command sequence for flash device programming according to recommendations of the flash device manufacturer, as described in Section 6.1.4.



- 13. Flash programming applet unit testing
  - a) For flash programming algorithm testing, define custom flash device parameters in the flash\_device.h file. The following parameters should have correct definitions.
    - BASE\_FLASH\_ADDRESS ColdFire CPU view of the flash device's address
    - SECTOR\_ADDRESS\_OFFSET memory sector size
    - NUMBER\_ITEMS test parameter, which defines how much data is programmed during the flash program testing.
  - b) Refer to Listing 7.

**NOTE** Refer to the flash device manufacturer for the flash device memory organization. Refer to hardware description for the flash device addressing.

### Listing 7. Function Template write in algo\_impl.c File

```
/* Base Address of the flash */
#define BASE_FLASH_ADDRESS 0xFFE00000UL
/* Offset of the sector to erase for the test */
#define SECTOR_ADDRESS_OFFSET 0x4000UL
/* Number of bytes to program for the test
This parameter could not be more then Flash size
*/
#define NUMBER_ITEMS 1024
/* Set this to one if chip erase is supported */
#define HAS_CHIP_ERASE 1
```

14. Compile flash algo development target

- a) During new algorithm creation and testing, use the Flash Algo Development build target of the Flash Development Kit. Compile the Flash Algo Development target with needed modifications to flash\_algo.c for flash programming procedures. Compilation will result in creation of a new flashalgodev.elf file.
- 15. Flash algorithm unit test
  - a) To simplify flash programming algorithm creation and testing, flash test functionality is included with the Flash Tool Kit in the Flash Algo Development build target. Check the file flash\_test.c for it. Unit test functions contain basic functionality required for the flash programming; the following tests are performed:
    - i) check flash device's ID
    - ii) erase flash memory sector
    - iii) program flash memory sector with the predefined data (in sample code the incrementing counter is used)
  - b) Load the file flashalgodev.elf and run it on the target board. Check the tests results. As an example of the test working refer to Section 6.2.

- 16. Compile flash algo release target
  - a) When the flash programming algorithm for the new flash device works correctly (as confirmed in unit testing), compile the Flash Algo Release target. The output of the Flash Algo Release file flashalgorelease.elf must be copied to the following folder:

```
Freescale\CodeWarrior for ColdFire V7.x
```

```
\bin\Plugins\Support\Flash_Programmer\ColdFire
```

- 17. New flash device addition to the flash programmer
  - a) The application note *AN 3390 Adding Device(s) to the Flash Programmer* provides a detailed description about adding a new flash device. Refer to this application note and do the steps it describes. As an example refer to Section 6.3.
- 18. Set flash device configuration in flash programmer.
  - a) Do the following to set the flash device configuration correctly in CodeWarrior Flash Programmer:
    - i) The CodeWarrior IDE must be closed (if opened) and opened again for you to use new data from the updated FPDeviceConfig.xml file.
    - ii) Open the CodeWarrior Flash Programmer window. From CodeWarrior menu: Tools->Flash Programmer, as shown in Figure 9.

### Figure 9. Opening the CodeWarrior Flash Programmer



iii) Load Target Configuration settings. As an example, the modified file NewFlashDevice.xml is loaded, as shown in Figure 11.



Creating a New Flash Programming Algorithm

Figure 10. Target Hardware Connection Settings

Flash Programmer		×
Flash Programmer Target Configuration Flash Configuration Erase / Blank Check Program / Verify Checksum	Target Configuration       Default Project:       Flash_ToolKit.mcp       Default Target:       Flash Algo Development       Image:       Processor Family:       All	
	Target Processor:       Generic       Connection:       PEMICRO_USB         ✓       Use Target Initialization       ABATRON_SERIAL         ✓       Use Target Initialization       ABATRON_SERIAL         C:\Program Files\Freescale\CodeWarrior for ColdFire       V7.0\ColdFire         FREESCALE_USE-TAP       FREESCALE_USE-TAP         PEMICR0_CYCMAX_SERIAL       PEMICR0_CYCMAX_SERIAL         PEMICR0_CYCMAX_TCPIP       PEMICR0_CYCMAX_USB         PEMICR0_CYCMAX_USB       PEMICR0_CYCMAX_USB         PEMICR0_CYCMAX_USB       PEMICR0_CYCMAX_USB         Target Memory Buffer       Optio         Target Memory Buffer Address:       0x         02000000       Image: Enable Logging         Target Memory Buffer Size:       0x         000006000       Image: Verify Target Memory Writes	
	Show Log Load Settings Save Settings Close	

iv) Set the correct connection to the target, as shown in Figure 10.





Load Settings F	ile	? 🛛
Look in:	ColdFire	
My Recent Documents	FPDefaultConfig.xml FPDeviceConfig.xml M5206EC3_EXTFLASH.xml M5208EVB_EXTFLASH.xml M5208EVBE_EXTFLASH.xml M5208EVBE_EXTFLASH.xml M5235EVB_EXTFLASH.xml	M5373EVB_EXTFLASH.xml M5407C3_EXTFLASH.xml M5475EVB_EXTFLASH.xml M5475EVB_EXTFLASH_CODE_16Mb.xml M5475EVB_EXTFLASH_CODE_32Mb.xml M5475EVB_EXTFLASH.xml
My Computer	M5249C3_EXTFLASH.xml M5253EVBE_EXTFLASH.xml M5271EVB_EXTFLASH.xml M5272C3_EXTFLASH.xml M5275EVB_EXTFLASH.xml M5282EVB_EXTFLASH.xml M5307C3_EXTFLASH.xml M5329EVB_EXTFLASH.xml	M5475EVBE_EXTFLASH_CODE_16Mb.xml M5485EVB_EXTFLASH_CODE_16Mb.xml M5485EVB_EXTFLASH_CODE_16Mb.xml M5485EVB_EXTFLASH_CODE_32Mb.xml M5485EVBE_EXTFLASH_Xml M5485EVBE_EXTFLASH_CODE_16Mb.xml M52277EVB_EXTFLASH_Xml M54455EVB_EXTFLASH.xml
My Network Places	M5329EVBE_EXTFLASH.xml         File name:       M5282EVB_EXTFLASH:         Files of type:       Flash Programmer Setting	MCF5211_INTFLASH.xml xml Qs files (*xml) Cancel

v) The flash device configuration checks the loaded configuration data for correspondence. As an example for the AMD16x1 device, refer to Figure 12.

Figure 12. CodeWarrior Flash Programmer Flash Device Configuration

Flash Programmer		
Flash Programmer Target Configuration Flash Configuration Erase / Blank Check Program / Verify Checksum	Flash Device Configuration         Flash Memory Base Address:       0x       ffe00000         Device:       Organization:       Sector Address Map:         AM29LV004BB AM29LV004BT AM29LV008BB AM29LV008BB AM29LV008BB AM29LV008BT AM29LV008BT AM29LV008BT AM29LV008BT AM29LV008D AM29LV008D-HAWK AM29LV008D-HAWK AM29LV085D-HAWK AM29LV085D-HAWK AM29LV08BB AM29LV108BB AM29LV116BB AM29LV116BB AM29LV116BB AM29LV116BB AM29LV160BB AM29LV160BB AM29LV160BB AM29LV200B       Image: Constraint of the	
	Show Log Load Settings Save Settings Clos	se

- 19. Erase and blank check the flash device
  - a) Check that the flash device can be erased by clicking the **Erase** button in the **Erase/Blank Check** window. Check that the Blank Check test passed: click the **Blank Check** button in the **Erase/Blank Check** window, as shown in Figure 13.



Flash Programmer			
Flash Programmer	Erase / Blank Check Flash		
Target Configuration Flash Configuration <u>Frase / Blank Check</u> Program / Verify Checksum	✓ All Sectors	Erase/Blank Check Sectors Individually	
	, 	Direct Objects	
	Erase	Biank Check	
	Show Log Load Settings	Save Settings	Close

🖿 Flash Programmer		×
Flash Programmer	Erase / Blank Check Flash	
Target Configuration Flash Configuration <u>Erase / Blank Check</u> Program / Verify Checksum	✓ All Sectors       ✓ Erase/Blank Check Sectors Individually	
	Status: Blank Check Completed Successfully Details	
	Erase Blank Check	
	Show Log Load Settings Save Settings Close	



b) In the log file, check that the correct flash programming algorithm is used for NewFlashDevice programming. Click the **ShowLog** button in **Erase/Blank Check** window. Refer to the example log in Figure 14.

#### Figure 14. Flash Programming Log Window

Flash Programmer Log	
:Using Algorithm: amd16×1.elf	Ĩ
Target Configuration Settings Connection: PEMICRO_USB Target Processor: 5282 Target Init File: D:\autobuild\Codewarrior71_21208\ColdFire_Support\Initialization_Files\ m5282evb.cfg	
ConnectingConnected	
Execute: Erase Timestamp: Mon Feb 18 14:15:48 2008 Flash Device: AM29LV160DB Flash Mem Start Addr: 0.xFFE00000 Flash Mem End Addr: 0.xFFFFFFF Informing other connection Clients that the target is being 'clobbered'. Loading Flash Device Driver at: 0.xF00001800 Flash Driver Buffer 15 at; 0.0000000 Flash Driver Buffer 15 at; 0.00000000 Flash Driver Buffer 15 at; 0.00000000000000000000000000000000000	
Flash Driver Buffer Size is: 0x000FE7FF Done Initialization Command Succeeded Erasing Sector 0xFFE00500 to 0xFFE03FFF Erasing Sector 0xFFE04000 to 0xFFE03FFF Erasing Sector 0xFFE06000 to 0xFFE05FFF	
Erasing Sector 0xFFED0000 to 0xFFEDFFF Erasing Sector 0xFFED0000 to 0xFFEFFF Erasing Sector 0xFFE20000 to 0xFFE2FFFF Erasing Sector 0xFFE40000 to 0xFFE3FFF Erasing Sector 0xFFE40000 to 0xFFE3FFF Erasing Sector 0xFFE40000 to 0xFFE3FFF Erasing Sector 0xFFE50000 to 0xFFE3FFF Erasing Sector 0xFFE50000 to 0xFFE3FFF Erasing Sector 0xFFE30000 to 0xFFE3FFF Erasing Sector 0xFFE30000 to 0xFFE3FFF Erasing Sector 0xFFE30000 to 0xFFE3FFF Erasing Sector 0xFFE30000 to 0xFFE3FFFF Erasing Sector 0xFFE30000 to 0xFFE3FFFF Erasing Sector 0xFFE30000 to 0xFFE3FFFF Erasing Sector 0xFFE30000 to 0xFFE3FFFF	
Close	

20. Programming test:

a) Different-sized binary S-record files are available in the Flash Toolkit delivery to check whether the flash device can be programmed. The path to the S-record files is:

```
Freescale\CodeWarrior for ColdFire V7.x
\ColdFire_Tools\FlashToolKit\SrecTestFiles
```

- i) They are: 64k\_at\_0.S, 128k\_at\_0.S, 256k\_at\_0.S, 1M\_at\_0.S, 2M\_at\_0.S and 4M\_at\_0.S. Depending on the file name, test files are differently sized. For example file 256k\_at\_0.S is 256 Kilobyte-sized and is linked to the 0 start-up address.
- b) Choose the Program/Verify sub-menu in CodeWarrior Flash Programmer. For programming the AMD 16x1, the 1M\_at\_0.S file is used, as shown in Figure 15.



🖪 Flash Program	nmer	Σ
Flash Programmer	Program / Verify Flash	
Load Settings F	ile	? 🛛
Look in:	TestSrecFiles	Browse
My Recent		
Documents	■ 16k_at_0.S ■ 16k_at_0.S	
Desktop	32k_at_0.S     64k_at_0.S	
<b>&gt;</b>	□ 128k_at_0.S □ 256k_at_0.S □ c12k_at_0.S	
My Documents	画 512K_a(_0.5	
My Computer		
		Details
My Network Places	File name: 1M_at_0.S	▼ Open
,	Files of type: All Files (*.*)	Cancel
	Show Log Load Settings Save Se	ettings Close

Figure 15. Dummy file for Flash Program Testing Selection

c) Specify **Restrict Address Range** and **Apply Address Offset** for the flash device being used (Figure 16.)

Figure 16. Restrict Address Range and Apply Address Offset Settings

Flash Programmer		<
Flash Programmer Target Configuration Flash Configuration Erase / Blank Check Program / Verify Checksum	Program / Verify Flash         ✓ Use Selected File         C:\Program Files\Freescale\CodeWarrior for ColdFire V7.0\ColdFire_Tools\FlashTo         Browse         File Type:       Auto Detect         ✓ Restrict Address Range       ✓ Apply Address Offset         Start:       0x fffe00000         End:       0x fffe00000         Flash Base Address:       0xffe00000	
	Status: Program Command Succeeded Details	
	Program Verify	
	Show Log Load Settings Save Settings Close	



- d) Figure 16. definitions:
  - i) Restrict Address Range address range of the flash device
  - ii) **Apply Address Offset** start address, where the test data is programmed in the flash; it should be the flash device start address.
- e) Check that the flash is programmed and the **Program Command Succeeded** message is shown after clicking the **Program** button, as shown in Figure 16.

**NOTE** In case of the flash device cannot be programmed, check: 1) successful erasure of flash device; 2) hardware connection correctly setup.

- 21. Verify programmed data:
  - a) If the flash device is successfully programmed, perform the **Verify** command (Figure 16.) Check that the status is: **Verify Command Succeeded**.
  - b) If all tests pass correctly, you have completed creation of a the new flash programming algorithm. The new flash device can be programmed with CodeWarrior Flash Programmer without limitation.

# 6 Flash Programming Examples

### 6.1 Flash Programming Algorithm for AMD 16x1 Flash Devices

The AMD\_16x1\_Example.mcp project (Figure 17.) illustrates how the Flash Development Kit is used to program AMD 16x1 flash algorithms.

Figure 17.	ColdFire	CodeWarrior	Main	View of	AMD_	_16x1_	Exampl	e.mcp	Project

AMD_16x1_Example.mcp			
🍽 Flash Algo Development 🔽 🔝 😽 🍊	<b>\$</b> •		
Files Link Order Targets			
💉 File	Code	Data 🔞	🕊 🔺
📓 flash_info.txt	n/a	n/a •	<b>x</b>
🖃 🎰 SDK Common Files - No Modification Need	360	20 •	• 🔳
	48	0•	• 🔳
📲 exit.c	12	0•	• 🔳
📲 flash_algorithm.lcf	n/a	n/a •	
📲 flash_commands.h	0	0•	
📲 flash_main.c	300	20 •	• 🔳
🔤 🛐 generic.h	0	0•	
🖃 🥽 User Files - AMD Algo example	496	0•	• 🔳
algo_impl.c	496	0•	+ 🔳
🖃 🥽 User Tests - Implement Algo test	320	1028 •	• 🔳
📲 flash_device.h	0	0•	
flash_test.c	320	1028 •	• •
10 files	1176	1048	



# 6.1.1 Implementation: ID Function Implementation for AMD 16x1 Flash Devices

The sequence for getting the Manufacturer ID and Device ID, based on the AMD flash specification, is shown in Figure 18.

		-		 
Figure 18	ID Command	Sequence	for the	Flash
riguic ro.		ocquentoe		1 1001

	ę٩			Bus C	cles		
Command Sequence		Fli	rst	Sec	ond	Thir	d
(Note 1)	o	Addr	Data	Addr	Data	Addr	Data
Manufacturer ID	4	555	AA	2AA	55	(BA)555	90
Device ID	6	555	AA	2AA	55	(BA)555	90

### Listing 8. ID Function Sample Code for AMD Flashes

```
retval_t ID(parameter_block_t *p_pb)
{
    volatile unsigned short *baseaddress = (p_pb->base_addr).w;
   retval_t result = 0;
    /* reset */
    *(baseaddress) = (unsigned short)0xF0F0;
    /* setup for get id */
    *(baseaddress + 0x555) = (unsigned short)0xAA;
    *(baseaddress + 0x2AA) = (unsigned short)0x55;
    *(baseaddress + 0x555) = (unsigned short)0x90;
#ifdef FLASH_ALGO_TEST
    /* get id */
           = *(baseaddress);
   mf_id
            = *(baseaddress + 1);
   part_id
#endif
    /* read mode again */
    *(baseaddress) = (unsigned char)0xF0;
   return result;
}
```

When using the Algo Development build target, the device ID and manufacturer's ID are read from the flash device and stored in the part\_id and mf\_id variables (Listing 8). Check these during the flash algorithm testing.

### 6.1.2 Implementation: Function erase\_sector for AMD 16x1 Flash Devices

The sequence for the Sector Erase command implementation, based on the AMD flash specification, is shown in Figure 19.



#### Figure 19. Sector Erase Command Sequence for AMD Flash

	ŝ					В	us Cycl	les (Notes	2-5)				
Command Sequence	ycle	First		Second		Third		Fourth		Fifth		Sixth	
(Note 1)	0	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Sector Erase	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30

Refer to the actual encoding of the erase\_sector function for AMD flashes in Listing 9.

#### Listing 9. Function erase\_sector Sample Code for AMD Flashes

```
retval_t erase_sector(parameter_block_t *p_pb, unsigned long sect_index)
{
   volatile unsigned short *sectoraddress = ((unsigned short **)(p_pb->items).w)[sect_index];
    volatile unsigned short read;
    retval_t result = 0;
    /* first of all reset the device. The fID is no longer called in the new
    flash programmer plugin (it was used in the old AMC MWX-ICE) therefore
    the flash chip must always be bring into the read state.
    */
    /* reset sector */
    *(sectoraddress) = (unsigned short)0xF0F0;
    /* erase sector */
    *(sectoraddress + 0x555) = (unsigned short)0xAA;
    *(sectoraddress + 0x2AA) = (unsigned short)0x55;
    *(sectoraddress + 0x555) = (unsigned short)0x80;
    *(sectoraddress + 0x555) = (unsigned short)0xAA;
    *(sectoraddress + 0x2AA) = (unsigned short)0x55;
    *(sectoraddress) = (unsigned short)0x30;
    read = *(sectoraddress);
    /*
   Wait for the status value to be read from *addr or
   how_long ticks to pass. If how_long ticks pass,
    a non-0 value will be returned.
    On the AMD chips, DQ7 is inverted until the embedded
    algorithm is completed when it flips to the correct
    value. The parameter 'hi' will indicate whether that
    value is set or cleared.
    * /
    while ((read & 0x0080) != 0x0080)
    {
        read = *(sectoraddress);
    }
    /* read mode again */
    *(sectoraddress) = (unsigned char)0xF0;
    return result;
}
```



### 6.1.3 Implementation: Function erase\_chip for AMD 16x1 Flash Devices

The sequence for the **Chip Erase** command, based on the AMD flash specification, is shown in Figure 20. and Listing 10.

#### Figure 20. Chip Erase Command Sequence for AMD Flash

	ŝ	Bus Cycles (Notes 2-5)											
Command Sequence		First		Second		Third		Fourth		Fifth		\$b	(th
(Note 1)	o	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Chip Erase	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10

Listing 10. Function erase\_chip Encoding for AMD Flashes

```
retval_t erase_chip(parameter_block_t *p_pb)
{
    int errors = 0;
    retval_t result = 0;
    unsigned short stat;
    unsigned short mask = (unsigned short)DQ7;
unsigned short masked_src = (unsigned short)DQ7;
    int got_it;
   volatile unsigned short *base_addr = (p_pb->base_addr).w;
    /* first of all reset the device. The fID is no longer called in the new
     flash programmer plugin (it was used in the old AMC MWX-ICE) therefore
    the flash chip must always be bring into the read state.
    */
    *base_addr = (unsigned short)0xF0F0;
/* erase chip */
    *(base_addr + 0x555) = (unsigned short)0xAA;
    *(base_addr + 0x2AA) = (unsigned short)0x55;
    *(base_addr + 0x555) = (unsigned short)0x80;
    *(base_addr + 0x555) = (unsigned short)0xAA;
    *(base_addr + 0x2AA) = (unsigned short)0x55;
    *(base_addr + 0x555) = (unsigned short)0x10;
/* Wait for status operation */
   mask &= 0x0080;
                            /* Only dq7 flips */
   masked_src &= 0x0080;
   while (1)
    {
        if ( (*base_addr & mask) == masked_src )
        {
        break;
        }
    }
    /* return to read arry mode */
    *base_addr = (unsigned char)READ;
    return result;
}
```



### 6.1.4 Implementation: Function write for AMD 16x1 Flash Devices

In terms of AMD flash devices specification, the write function realizes the **Program** command. The sequence for the Program command, according to the AMD specification, is shown in Figure 21.

Figure 21. Program Command Sequence for AMD Flash

Command Sequence	ŝ			Bus C	ycles		
	ycle	First		Second		Third	
	0	Addr	Data	Addr	Data	Addr	Data
Program	4	555	AA	2AA	55	555	AD

Refer to the actual encoding of the write function for AMD flashes as shown in the file amd\_16x1\_sample.c in Listing 11.

Listing 11. Sample write Function Code for AMD Flashes

```
retval_t write(parameter_block_t *p_pb)
       {
          int timed_out, got_it;
          unsigned long i;
          unsigned short stat;
          retval_t errors = 0;
          unsigned short mask = (unsigned short)DQ7;
       unsigned short masked_src = (unsigned short)DQ7;
          volatile unsigned short *base_addr = (p_pb->base_addr).w;
          unsigned short *buffer = (p_pb->items).w;
          unsigned long buffer_len = p_pb->num_items;
          unsigned long how_many = buffer_len / sizeof(unsigned short);
           if ( buffer_len % sizeof(unsigned short) ) {
               /* we need to fill the remaining bytes with 'ff' -- this assumes
              byte accesses to DRAM will work */
               char *p = (char *)((unsigned long)buffer + buffer_len);
               *p++ = '\xff';
               how_many++ ;
           }
           /* first of all reset the device. The fID is no longer called in the new
           flash programmer plugin (it was used in the old AMC MWX-ICE) therefore
           the flash chip must always be bring into the read state.
           */
           *base_addr = (unsigned short)RESET;
           for (i = 0; (i < how_many) && !errors; i++) {</pre>
               unsigned short *c = (unsigned short*)((unsigned long)base_addr & ~0x1fff);
               *((c) + 0x555) = 0xaa;
               *((c) + 0x2aa) = 0x55;
               *((c) + 0x555) = 0xa0;
               *base_addr = *buffer;
       /* Wait for status operation */
                                   /* Only dq7 flips */
          mask &= 0x0080;
          masked_src = (unsigned short)((unsigned char)DQ7 & *buffer);
          masked_src &= 0x0080;
```



}

Flash Programming Examples

```
while ( 1 )
{
    if ( (*base_addr & mask) == masked_src )
    {
        break;
    }
}
base_addr++;
    buffer++;
}
/* go back to the last access */
--base_addr;
/* read mode again */
*base_addr = (unsigned char)0xF0;
return errors;
```

# 6.2 AMD 16x1 Flash Programming Algorithm Unit Testing

This section illustrates an example flash test application working with AMD 16x1. The flash programming applet is tested on a Freescale M5282EVB with an Am29PL160CB flash device.

### 6.2.1 Flash Testing Setup

Use the Algo Development target — shown in Figure 22. — to run the flash programming test application.

Figure 22. Targets in AMD\_16x1\_Example.mcp Project

	- ×
AMD_16x1_Example.mcp	
🔹 Flash Algo Development 🖃 🏥 😻 🗸	🏽 💺
Files Link Order Targets	
Targets	X
₩9, Flash Algo Development ④, Flash Algo Release	4
2 targets	
<	>

Upon loading, the application stops at the \_flash\_start() function as shown in Figure 23.



Figure 23. Unit Test Application Start-up Point



### 6.2.2 Test One: Read Manufacturer and Device ID

After the Run command is executed the application stops at the first test check point, as shown in Figure 24.

Figure 24. Read Manufacturer and Device ID

🖿 flashalgodev.elf (Thread 0x0)	
💺 = 🗶 🗇 🗗 🔜 🗑	
Stack	Ð
main	-
▶ ID	
result 1024 0xF0100E08	
mf_id 0 0xF0100846	
part_id 0 0xF0100844	-
▼	
Source: C:\Program Files\Freescale\CodeWarrior for ColdFire V7.0\ColdFire_Tools\FlashToolKit\AMD16x1Example\algo_impl.c	_ 🗉
extern volatile unsigned short mf_id; extern volatile unsigned short part_id; #endif	<u> </u>
#define DQ7 0x0080080 #define DQ5 0x00200020	
Intelligent identification function Note that we always return long values.	
retval_t ID(parameter_block_t <b>*p_pb</b> ) - {	
<pre>- volatile unsigned short *baseaddress = (p_pb-&gt;base_addr).w; - retval_t result = 0;</pre>	
<pre>/* reset */ - *(baseaddress) = (unsigned short)0xF0F0;</pre>	
<pre>/* setup for get id */     *(baseadtress + 0x555) = (unsigned short)0xAA;     *(baseadtress + 0x2AA) = (unsigned short)0x55;     *(baseadtress + 0x555) = (unsigned short)0x90;</pre>	
<pre>#ifdef FLASH_ALGO_TEST</pre>	
<pre>- mf_id = *(baseaddress); - part_id = *(baseaddress + 1); #endif</pre>	
<pre>/* read mode again */ - *(baseaddress) = (unsigned char)0xF0;</pre>	
- return result; - }	
0 J Line 25 Col 11   Source ▶ ◀	



The results of Test One display the manufacturer ID code  $0 \times 01$  (for AMD) and the device ID code  $0 \times 2245$  (for the Am29PL160CB flash device). This confirms basic read/write functionality of the flash devices.

### 6.2.3 Test Two: Erase a Sector

With another **Run** command execution the application stops at the Test Two check point, as shown in Figure 25. In Test Two the sector number one of the flash memory is erased. From the sample flash device definition for AMD 16x1 we have: BASE\_FLASH\_ADDRESS equal to 0xFFE00000 and the SECTOR\_ADDRESS\_OFFSET equal to 0x4000. Thus for sector 1, flash memory is erased starting at address 0xFFE04000 in memory. To check that the **Erase a Sector** command works correctly the memory window was opened with the memory region starting at address 0xFFE04000. Upon erasure, flash memory sector contains 0xFFFFFFF data in its memory.

### Figure 25. Erase Sector Functionality Check Point

🔳 flashalgo	dev.elf M	emory 1							
Display:	sectoraddr	BSS					View: F	Raw data	•
Address FFE04000 FFE04010 FFE04020 FFE04030 FFE04030 FFE04040 FFE04050 FFE04060 FFE04070 FFE04080 Word Size:	Hex: FFE FFFFFFFF FFFFFFFF FFFFFFF FFFFFFF FFFFFF	03800:FFEC FFFFFFF FFFFFFF FFFFFFF FFFFFFF FFFFFF	14800 F FFFFF F FFFFF F FFFFFF F FFFFF F FFFFF F FFFFF F FFFFF	FFF FFF FFF FFF FFF FFF	FFFFFFF FFFFFFF FFFFFFF FFFFFFF FFFFFFF	A A			* * *
🔳 flashalgo	dev.elf (T	hread Ox	0)						
💺 🔳 🗙	0 B	<u>ф</u>   🗉	×Y 🏶						
<mark>§ Stack</mark> main ♦ check_test			<u> </u>		<ul> <li>Variables: Live</li> <li>id</li> <li>testnumber</li> <li>mf_id</li> <li>part_id</li> </ul>	Value 0 2 0 0	L	ocation 0xF0100E0 0xF0100E1 0xF010084 0xF010084	208 <u>^</u> 14 14 14
▼			<u> </u>						
Source: I /* Tes: -	D:\Program F t 1: Rea case 1 as /*	iles\Freesc d Manuf .: m { hal check	ale\CodeV acture t;}; the va	vani er a clue	or for ColdFire V7.0" and Device 1 es of mf_id	\ColdFire_ ID */ and pa	Tools\Flash	To\flash_t here abo	
- - - - * Tes	id id br t 2: Era case 2 as /*	l = mf_i l = part reak; use a se :: sm { hal check	d; ;_id; ector * lt;}; the se	⊧∕ ecto	or you want	to era	ase has	been er	ase
- 0 ⊾ Line 121	Col 26	eak; Source	• •						•



The results of Test Two show that flash memory, starting at address 0xFFE04000, is erased. This confirms that the sector erase function works correctly.

### 6.2.4 Test Three: Program Flash Memory

Another execution of the Run command stops the test application at the Test Three check point, as shown in Figure 26. Test Three fills sector one in flash memory with an incremental counter. The number of the bytes written to flash memory is determined by the NUMBER\_ITEMS parameter in flash\_device.h file. To verify that the flash programming algorithm works correctly and the write function of the flash performs correctly, check the memory region of flash sector one. In this case, check the memory starting at address 0xFFE04000.

### Figure 26. Program Functionality Check Point

💼 flashalgo	dev.elf Memory 1				
Display:	0xFFE04000		View	Raw data	-
	N Hex: FFE03800;FFE04800		Ascii		
FFE04000	00010203 04050607 08090A	OB OCODOEOF			~
FFE04010	10111213 14151617 18191A	18 1C1D1E1F			-
FFE04020	20212223 24252627 28292A	28 2C2D2E2F	1"# \$%8/1	0*+/	
FFE04030	30313233 34353637 38393A	3B 3C3D3E3F	0123 4567	89:; <=>?	
FFE04040	40414243 44454647 48494A	4B 4C4D4E4F	@ABC DEFG	HIJK LMNO	
FFE04050	50515253 54555657 58595A	SB SCSDSESF	PQRS TUVW	1X72[\]A_	
FFE04060	60616263 64656667 68696A	6B 6C6D6E6F	`abc defg	hijk 1mno	
FFE04070	70717273 74757677 787974	78 7C707E7F	pgrs tuvw	/×yz{ ]}~+	-
FFE04080	80818283 84858687 888984	8B 8C8D8E8F			~
Word Size	32 👻				
🔲 flashaloo	dev.elf.(Thread.0x0)				
a nasnago	devicer (riffedd oxo)				لمار
💺 🔳 🗙	다 다 다 🗉 📼 🕷				
Stack	8	🛛 🔊 Variables: Live	Value	Location	20
main	*	id	0	0xF0100E08	-
check_test		testnumber	3	0xF0100E14	
		mf_id	0	0xF0100846	
		part_id	0	0xF0100844	
	-				-
~					
Source:	C:\Program Files\Freescale\CodeW	arrior for ColdFire V7.0V	ColdFire Tools\Fla	shT\flash_test.	ല
/* Tes	t 3: Write NUMBER ITE	MS to the sect	or */		-
	case 3:				
-	<pre>asm { halt;};</pre>				
	/* check progra:	mming was succ	estull */		
/* Tes	t 4: Erase the sector	*/			
	case 4:				
-	asm { halt;};				
-	<pre>/* check the set break;</pre>	ctor you want	to erase na	s deen era:	St
	Dictar,				
#if_HA	S_CHIP_ERASE == 1				
/* Tes	t 5: Erase the chip *				
-	asm { halt:}				
	<pre>/* check erase *</pre>	was succesfull	*/		
-	break;				
#endif					-
0 Line 128	Col 1 Source 🕨 🗸				

The results of Test Three show that flash memory, starting at address 0xFFE04000, is changed and contains an increment by one datum. This confirms that the write function works correctly.

### 6.2.5 Test Four: Erase a Sector

Erase Sector One of flash memory again to check that the above results are not produced in error (Figure 27.)



💼 flashalgodev.elf Memory 1			
Display: 0xFFE04000		View	🛙 Raw data 🖉 💌
Address         Hex: FFE03800:FFE04800           FFE04000         FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	FF FFFFFFF FF FFFFFFF FF FFFFFFF FF FFFFFF	Ascii	**
flashalgodev.elf (Thread 0x0)			
▶ = × 다 관 쇼 □ ∞ 로			
■ stack ビ main ◆ check_test	id testnumber mf_id part_id	Value 0 4 0 0	Location         E           0xF0100E08         2           0xF0100E14         2           0xF0100846         2           0xF0100844         2
Surce: C:\Program Files\Freescale\CodeW.  Test 3: Write NUMBER_ITEN  case 3:     asm { halt;};      * Test 4: Erase the sector  case 4:	arrior for ColdFire V7.0 IS to the sect ming was succ */	ColdFire_Tools\FL or */ esfull */	ashT\flash_test.c_E
<pre>- asm { halt;};</pre>	otor you want / was succesfull	to erase ha	xs been eras:
O ⊾ Line 133 Col 1 Source ► ◀			

Figure 27. Erasing Sector One of Flash Memory After Programming

Check the memory starting at address 0xFFE04000 to verify that the flash memory region erased correctly. In this example, since the modified memory region contains 0xFFFFFFFF, sector data was erased successfully.

### 6.2.6 Flash Testing End Point

After finishing, the flash test application goes to the flash\_exit() end point, as shown in Figure 28.

### Figure 28. Flash Testing End Point

💼 flashalgodev.elf (Thread 0x0)	<
★ = ★ G G G ★ = ★	
Stack	1
Filash_exit	1
	2
<b>V</b>	-
Source:C:\Program Files\Freescale\CodeWarrior for ColdFire V7.0\ColdFire_Tools\FlashToolKit\\exit.c_ 🗉	ונ
<ul> <li>* Flash Programming SDK</li> <li>* Copyright © 2007 Freescale Semiconductor. All rights reserve</li> <li>*/</li> </ul>	5
asm void flash_exit(void);	
asm void flash_exit(void)	
<pre>' /* setup the stack pointer */ - + nop - halt</pre>	
loop: bra _loop - illegal	
- rts	
}	
	-1
O Line 10 Col 1 Source ► <	



When this test is complete, you can use the flash programming algorithm with the CodeWarrior Flash *Programmer.* 

# 6.3 Adding Flash Device Configuration to FPDeviceConfig.xml

To add a new device to the CodeWarrior Flash programmer add the new structured table of the flash device information to the master configuration file (FPDeviceConfig.xml). This master file is found in the following folder:

```
Freescale\CodeWarrior for ColdFire V7.x
\bin\Plugins\Support\Flash_Programmer\ColdFire\ (Windows)
```

NewFlashDevice is the new flash device that works with the flashalgorelease.elf executable, as shown in Listing 12.

Listing 12. Configuration Settings for NewFlashDevice in FPDeviceConfig.xml File

<comment></comment>
*****
# This configuration is for the NewFlashDevice #
*****
<device></device>
<name>NewFlashDevice</name>
<manufacturerid>01</manufacturerid>
<chiperase>TRUE</chiperase>
<sectorcount>11</sectorcount>
<sectorname>0</sectorname>
<sectorname>1</sectorname>
<sectorname>2</sectorname>
<sectorname>3</sectorname>
<sectorname>4</sectorname>
<sectorname>5</sectorname>
<sectorname>6</sectorname>
<sectorname>7</sectorname>
<sectorname>8</sectorname>
<sectorname>9</sectorname>
<sectorname>10</sectorname>
<sectorstart>000000</sectorstart>
<sectorend>003fff</sectorend>
<sectorstart>004000</sectorstart>
<sectorend>005fff</sectorend>
<sectorstart>006000</sectorstart>
<sectorend>007fff</sectorend>
<sectorstart>008000</sectorstart>
<sectorend>03ffff</sectorend>
<sectorstart>040000</sectorstart>
<sectorend>07ffff</sectorend>
<sectorstart>080000</sectorstart>
<sectorend>0bffff</sectorend>
<sectorstart>0c0000</sectorstart>
<sectorend>0fffff</sectorend>
<sectorstart>100000 </sectorstart>
<sectorend>13ffff</sectorend>
<sectorstart>140000</sectorstart>
<sectorend>17ffff</sectorend>
<sectorstart>180000</sectorstart>
<sectorend>1bffff</sectorend>



```
<sectorstart>1c0000</sectorstart>
<sectorend>1fffff</sectorend>
<organizationcount>1</organizationcount>
<organization>1Mx16x1</organization>
<id>2245</id>
<algorithm>flashalgorelease.elf</algorithm>
</device>
<comment>
```

Listing 12 definitions:

- NewFlashDevice the name of the new flash device for which the new flash programming algorithm is created
- flashalgorelease.elf name of the flash programming algorithm that will be used to program flash device

The NewFlashDevice configuration appears in the Flash Programmer window, as shown in Figure 12.

### 6.4 Updating the Flash Settings xml File for Flash Programmer

Update the custom board configuration file to have correct configuration settings that work with the new flash device.

For example, the sample flash programming applet for AMD 16x1 is compiled to work with the MCF5282EVB EVB board. A new file named NewFlashDevices.xml is generated based on M5282EVB.xml and is modified to work with the NewFlashDevice, as described in the FPDeviceConfig.xml file. Refer to Section 6.3. If another board is used, find the [BOARD].xml file for that board; this file can be used as a template to create one to work with the new flash device. Check the folder:

```
Freescale\CodeWarrior for ColdFire V7.x
\ColdFire_Support\bin\Plugins\Support\Flash_Programmer\ColdFire
```

To create NewFlashDevices.xml file for MCF5282EVB EVB board:

- 1. Open the M5282EVB.xml file with the CodeWarrior's Flash Programmer as shown in Figure 29. If other hardware is used open the [BOARD].xml configuration file, where BOARD is the name of the core used.
  - a) For target configuration [BOARD].xml files check the folder:

Freescale\CodeWarrior for ColdFire V7.x
\bin\Plugins\Support\Flash\_Programmer\ColdFire



Figure 29. Opening the M5282EVB.xml File



- 2. Check the target configuration settings as shown in Figure 30. The following parameters should be set correctly for the hardware used:
  - a) Target Processor set the target processor name correctly for the hardware used.
  - b) Use Target Initialization set the correct path to the target processor configuration file. In the provided example for the 5282EVB board, the M5282EVB.cfg file is used. For the processor configuration files please check the folder:

```
Freescale\CodeWarrior for ColdFire V7.x
\ColdFire_Support\Initialization_Files
```

- c) Target Memory Buffer Address target RAM memory start-up address.
- d) Target Memory Buffer Size target RAM memory size.



ash Programmer	Target Configuration
h Configuration h Configuration e / Blank Check gram / Verify cksum	Default Project: Default Target:
	Use Custom Settings
	Processor Family: All Target Processor: 5282 Connection: PEMICRO_USB Use Target Initialization CodeWarrior for ColdFire V7.0\ColdFire_Support\Initialization_Files m5282evb.cfg Browse
	Target RAM Memory Buffer Target Memory Buffer Address: 0x f00000000 F Enable Logging
	Target Memory Buffer Size: 0x 0000ffff

Figure 30. Checking Target Configuration Settings in [BOARD].xml File

3. If configuration is correct for the targeted hardware save the target board configuration as NewFlashDevices.xml, as shown in Figure 31.





Figure 31. Saving Target Hardware Configuration Settings for NewFlashDevices.xml File

4. Modify the NewFlashDevices.xml file as shown in Figure 32.







- 5. Check that the following parameters are set correctly for the NewFlashDevice in the NewFlashDevices.xml file:
  - a) <targetprocessor> set target processor name correctly for the hardware used.
  - b) <targetinitfile> set the correct path to the target processor configuration file. In provided example for 5282EVB board, M5282EVB.cfg file being used. For the processor configuration files please check the folder:

```
Freescale\CodeWarrior for ColdFire V7.x
\ColdFire_Support\Initialization_Files
```

c) <organization> - check the hardware organization for the new flash devices. In the provided example for AMD 16x1, one 1 Mbyte 16x1 AMD flash device is present on the tested 5282EVB board.



- d) <flashstart> check the hardware organization for the start-up address of the flash device being used. Check the same data for the <restrictaddrrangestart> and <addrstart> parameters.
- e) <addrsize> set correct size of the new flash device.
- f) <flashend> set the end address of the flash device. This parameter can be calculated as a <flashstart> + <addrsize>. Check the same data for the <restrictaddrrangeend> and <addrstart> parameters.
- g) After you change FPDeviceConfig.xml and the custom configuration file (NewFlashDevices.xml in this example), *the CodeWarrior Flash Programmer is ready to work with the new flash device.*

# 7 Chip Makers' Flash Programming Recommendations

In general, flash programming algorithms from different flash manufacturers are similar. Most manufacturers use the same algorithms for programming flash devices. For example, the same algorithms may be used for programming AMIC 16x1 flash devices and AMD 16x1 flashes. Most of the flash manufacturers have sample flash programming algorithms on the web.

### 7.1 Alliance Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Manufacturer's site: <u>http://www.alsc.com</u>.

### 7.2 AMIC Flash Devices

Depending on the particular flash device, the same flash programming algorithms available for AMD (Spansion) or Atmel should be usable. Check flash device specifications available from the manufacturer. Manufacturer's site: <u>http://www.amictechnology.com</u>.

# 7.3 AMD (Spansion) Flash Devices

Flash programming algorithms are already supported in CodeWarrior Flash Programmer. AMD does not produce its own flash devices any more (Fujitsu and AMD founded Spansion for flash manufacturing). Manufacturer's site: <u>http://www.spansion.com</u>.

### 7.4 Atmel Flash Devices

Due to the lack of testing hardware, flash programming algorithms for Atmel are not supported in the CodeWarrior ColdFire Flash Programmer. The flash programming algorithm for Atmel flash devices, however, is similar to the algorithm used for AMD flash programming.

Refer to the Atmel flash device specification for the correct programming commands and device ID variables. An application note, *Programming Atmel's AT29 Flash Family*, is available from the manufacturer's web site. The application note contains code examples for Atmel flash device programming. Manufacturer's site: <u>http://www.atmel.com</u>.



Chip Makers' Flash Programming Recommendations

# 7.5 Catalyst Flash Devices

Intel's flash programming algorithms should work. Check flash device specifications available from the manufacturer. Most of the flash devices from Catalist are fully identical to the flash devices from Intel. For example: the CAT28F001 from Catalist is the equivalent of Intel's IN28F001. Manufacturer's site: <u>http://www.catsemi.com/index.html</u>.

# 7.6 EON Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Most of the flash devices from EON have direct references to AMD flash devices. Manufacturer's site: <u>http://www.eonsdi.com</u>.

# 7.7 Fujitsu Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Fujitsu no longer produces its own flash devices. Manufacturer's site: <u>http://www.spansion.com</u>.

# 7.8 Hyundai Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Hyundai founded a new semiconductor company named Hynix. Most of the flash devices from Hynix make direct reference to AMD flash devices. Manufacturer's site: <u>http://www.hynix.com</u>.

# 7.9 Intel Flash Devices

Flash programming algorithms for Intel flash devices are already supported in CodeWarrior Flash Programmer. Manufacturer's site: <u>http://www.intel.com</u>.

# 7.10 Micron Flash Devices

Intel's flash programming algorithms should work. Check flash device specifications available from the manufacturer. Most of the flash devices from Micron make direct reference to Intel flashes. Manufacturer's site: <u>http://www.micron.com</u>.

# 7.11 MXIC Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Most of the flash devices from MXIC make direct reference to AMD flash devices. The document *MX\_FlashSampleCode.pdf* is available from the MXIC web site. The document contains sample flash programming code suitable for both MXIC and AMD flash devices. Manufacturer's site: <u>http://www.mxic.com.tw</u>.



### 7.12 Samsung Flash Devices

The CodeWarrior flash programmer does not support flash programming algorithms for Samsung. Samsung uses its own algorithm — incompatible with other vendors — for flash programming. A sample programming algorithm for Samsung flash devices, presented as a CodeWarrior for ARM project, is available from the manufacturer's web site. Manufacturer's site: <u>www.samsung.com/products/</u><u>semiconductor/OneNAND</u>.

### 7.13 Sharp Flash Devices

Intel's flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Manufacturer's site: <u>http://www.sharpsma.com</u>.

### 7.14 Spansion Flash Devices

Flash programming algorithms for Spansion flash devices are already supported in CodeWarrior Flash Programmer. Manufacturer's site: <u>http://www.spansion.com</u>.

### 7.15 SST Flash Devices

Depending on the particular flash device, the same flash programming algorithms available for AMD (Spansion), Atmel, or Intel should be usable. Check flash device specifications available from the manufacturer. Manufacturer's site: <u>http://www.sst.com/about</u>.

### 7.16 ST Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Sample flash programming application code is available for downloading from the ST web site. Manufacturer's site: <u>http://www.st.com</u>.

# 7.17 Toshiba Flash Devices

Intel's flash programming algorithms should work. Check flash device specifications available from the manufacturer. Manufacturer's site: <u>http://www.semicon.toshiba.co.jp/eng</u>.

# 7.18 White Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer. Manufacturer's site: <u>http://www.wedc.com</u>.

# 7.19 Winbond Flash Devices

AMD's (Spansion's) flash programming algorithms should be usable. Check flash device specifications available from the manufacturer.Manufacturer's site: <u>http://www.winbondusa.com/mambo/content/view/</u>289/553.



#### 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 or +1-480-768-2130 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 or +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:

Preescale Semiconductor Literature Distribution Center P.O. Box 5405 Denver, Colorado 80217 1-800-521-6274 or 303-675-2140 Fax: 303-675-2150 LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3389

08 February 2008

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 licenses 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 that 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.

Freescale<sup>™</sup> and the Freescale logo are trademarks of Freescale Semiconductor, Inc. CodeWarrior<sup>™</sup> is a trademark or registered trademark of Freescale Semiconductor, Inc. StarCore® is a registered trademark of Freescale Semiconductor, Inc. in the United States and/or other countries. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2006-2008. All rights reserved.

