8

Programming Flash Memory in Freescale S08/S12/CordFire MCUs Family

Yihuai Wang and Jin Wu
Soochow University
China

1. Introduction

The features of Flash memory include electrically erasable, no back-up power to protect data, in-circuit programming, high-density memory, low-cost and so on, which rapidly increase the using of Flash memory in embedded system.

The programming methods of Flash memory include Programmer Mode and In-Circuit Programmer Mode. Programmer Mode means erasing/programming Flash by programming tool (programmer), with the purpose of writing programs into MCU[1]. In-Circuit Programmer Mode means erasing/programming some region of Flash by MCU's internal programs during run time, with the purpose of saving relevant data and preventing from lost after power off. Take AW60/XS128/MCF52233 in Freescale 8/16/32bits S08/S12/ColdFire serials' MCUs for example, we elaborate the In-Circuit Programming method of Flash memory in this chapter. The programming method of the other MCUs in the whole Freescale S08/S12/ColdFire MCU family is similar. Besides, we discuss the protection mechanisms and security operations for AW60/XS128/MCF52233 Flash memory. Some instances are also provided in this chapter.

1.1 Flash memory characteristics

The most perfect memory should be a high-speed, non-volatile, low-cost and high-density memory. But only one or several specialties are implemented in general memory. With the maturity of its technology, flash memory has become an ideal memory in recent years. It is endowed with characteristics such as electrical erasure, data preservation without power supply, in-system programming, high storage density, low power consumption and low cost. These are just what MCU are expecting, because MCU with internal flash memory introduced in earlier years has some shortages in reliability and stability. With the maturity of the Flash technology, now more and more above characteristics are integrated to MCU and become an important part of it. Hence flash memory makes MCU progress enormously.

Flash memory is really a high-density, high-performance reading/writing memory with non-volatility, low-power and high-reliability and has the following characteristics comparing with old solid state memory.

[1] MCU—Microcontroller Unit

1. Non-volatility: Flash memory protects data without power supply the same as magnetic storage.
2. Easy-updating: Comparing with old EPROM[2], the electrical erasure of flash memory shortens the programming cycle for developers and makes end users' updating memory become true.
3. Low-cost, high-density and reliability: The parameters are much better than EEPROM (or E2PROM).

1.2 Flash memory program concepts

In many embedded systems, the memory which can protect program parameters and important data without external power supply is necessary as EEPROM previously was. The ColdFire MCU family provides the function of in-system programming of flash memory in user mode instead of EEPROM, hence making the circuit design simpler and cost lower.

However, different from the reading/writing of generic RAM, flash memory operations need two special processes—Erase and Program. The former, which converts all bits to 1, consists of mass erase and page erase. The latter, which converts bit to 0, can program only one word at a time. During erasing and programming, voltage higher than the power is usually needed and it is generated by Coldfire MCU inner electric charge pump. Besides, before programming, it should be insured that the program field has not been written after last erasure. That is, the field is blank (the content is $FF). So generally, erase should be carried out before perform.

1.3 In-circuit programming concepts of flash memory

In-circuit programming of flash memory in user mode (U-ICP) is a technique by which user programs stored in flash memory can modify data or programs also stored in the flash memory during run time. The electrically erasable characteristics of flash memory allow such programs to execute erase or write functions. This important branch of computer technology, an outgrowth of embedded systems development, makes it possible to update embedded programs, provide power-off protection and the recovery of important parameters, and modify the static parameters of embedded applications. In addition, U-ICP improves the expansibility and upgradeability of embedded systems. Portions of flash memory can substitute for the traditional EEPROM functions mentioned above, increasing system stability. And make the circuit design simpler and cost lower.

U-ICP was introduced to MCU technology by the semiconductor department of Motorola (now called Freescale) in 2000, and has been widely applied and developed ever since. However, different from the reading/writing of generic RAM, flash memory operations need two special processes—Erase and Program. The former, which converts all bits to 1, consists of mass erase and page erase. The latter, which converts bit to 0, can program only one word at a time. During erasing and programming, voltage higher than the power is usually needed and it is generated by Freescale S08/S12/CordFire MCU inner electric charge pump. Besides, before programming, it should be insured that the program field has not been written after last erasure. That is, the field is blank (the content is $FF). So generally, erase should be carried out before perform.

U-ICP can erase and reprogram other regions of flash memory by executing internal flash functions, but these may also prove unstable at high voltage. This problem, which is indicated

[2] EEPROM—Electrically Programmable Read-Only-Memory

in the data sheet of the Freescale S08/S12/CordFire MCU family, has not yet been solved by hardware design. It can be solved on the software level, by proper design of the U-ICP bottom driver program. So we describe an embedded software engineering rule that should improve the stability of the general erase and write functions in U-ICP for any flash device.

2. Programming flash in freescale MC9S08AW60 MCU

The flash memory in-circuit programming implement for 8bit MC9S08AW60 MCU will be explained in this section, as follows:

2.1 How to operate MC9S08AW60 flash memory

2.1.1 MC9S08AW60 Flash memory-mapping

S08 serials MCUs' addressable address space is 64Kbyte, which ranges from $0000 ~ $FFFF. This addressing range is divided into different sectors. Each sector has different function. The memory map of AW60 MCU is shown in Fig.1, which includes the address distribution of 2KB RAM[3], 2 parts of Flash memory and some I/O image registers.

As can be seen from the Fig.1, the Flash memory of AW60 is divided into two parts in this 64K memory address space. These addresses range from $0870～$17FF(3984 bytes) and $1860～$FFFF(59296 bytes). Among $1860～$FFFF only the addresses range from $1860～$FFAF can be used to erase and program user program. The addresses range from $FFB0～$FFBF are the 16 bytes non-volatile registers region and the addresses range from $FFC0～$FFFF are the 64 bytes interrupt vector region.

Flash memory is organized by page and row in the chip. The size of each page is 512 bytes. And the size of each row is 64 bytes. There are about 60K bytes of Flash memory address space in AW60, the page addresses are rounding 512 in $0000～$FFFF. For example, the first page's address of Flash memory in the 3984bytes region ($0870～$17FF) is $1000～$11FF, instead of $0870～$0A6F.

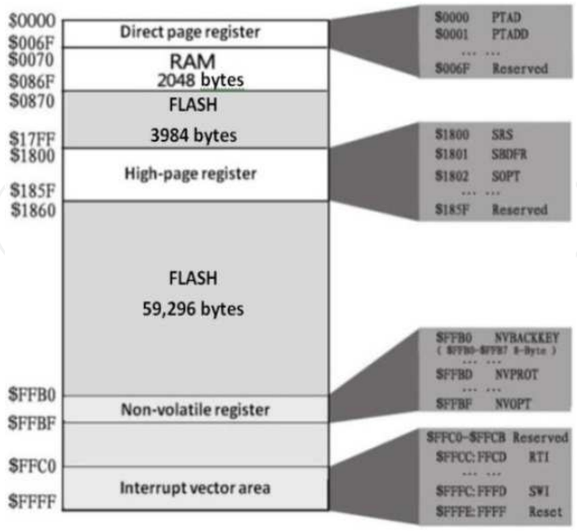


Fig. 1. The memory map of AW60 MCU

[3]RAM—Random Access Memory

For S08 serials MCU (AW60, etc.), we can do mass erasing operation for the Flash memory, or can erasing one page(512 bytes) from a certain start address. But we can't only erase a certain byte or some bytes which are less than 512 bytes. Noting this feature, it is important for the data arranging. The programming operation of AW60 is based on row (64 bytes). The data which can be programmed continuously at a time is only within one row. Certainly, the region that has not been erased can't be programmed.
As can be seen from the above, in order to program Flash memory, we should prepare a set of data and move them into RAM, then erase the corresponding region of Flash memory, so programming operation can be done. Because erasing /programming a certain byte of Flash memory will influence the follow-up one page, it is necessary to reasonably arrange the relevant data of erasing region before erasing/programming the Flash.

2.1.2 MC9S08AW60 FLASH registers and control bits

In AW60, Erasing and programming operations relate to registers such as FCDIV、FOPT、FCNFG、FPROT、FSTAT and FCMD. Their corresponding addresses are $1820、$1821、$1823、$1824、$1825 and $1826. For the detailed function and use of these registers, please refer to the Reference Manual "*MC9S08AW60 Data Sheet (HCS08 Microcontrollers)*"[1].

2.1.3 Flash programming procedure

1. The execution steps of Flash commands
   a. Write a data in an address of Flash. The address and data information will be locked into Flash interface. For blank check command, the data information is an arbitrary value; For page erase command, the address information is either one of the address in erase page (512 bytes) addresses; For blank check and mass erase commands, the address information is either one of the address in flash.
   b. Write the commands which are needed to be executed into FCMD.
   c. Execute commands. The FCBEF bit of FSTAT register is set, simultaneously execute the commands in the FCMD.
2. The flowcharts of the Flash programming

When programming flash, we need follow strict timing process. Fig.2 gives the programming flowchart with the other flash commands (not include the command "burst mode byte program"). Writing a byte with burst mode command is very different from the execution with other commands. Burst mode means a lot of continuous data need to be written into Flash. Each time a write command has been executed, the writing high voltage in flash will not be removed, which will speed up the write speed of data; But for other commands, the high voltage is given to ensure the command executing, and the high voltage is immediately removed when the command ended. The programming flowchart with burst mode command is shown in Fig.3.

3. Flash Memory Illegal Operations
   In the following processing, an error occurs, and the FACCERR bit is automatically set.
   a. Writing the flash memory before initializing FCDIV register.
   b. Writing the flash memory while FCBEF is not set.
   c. Writing the second command to the FCMD register before executing the previously written command。
   d. After write the flash memory, initializing the other flash control registers in addition to FCMD.

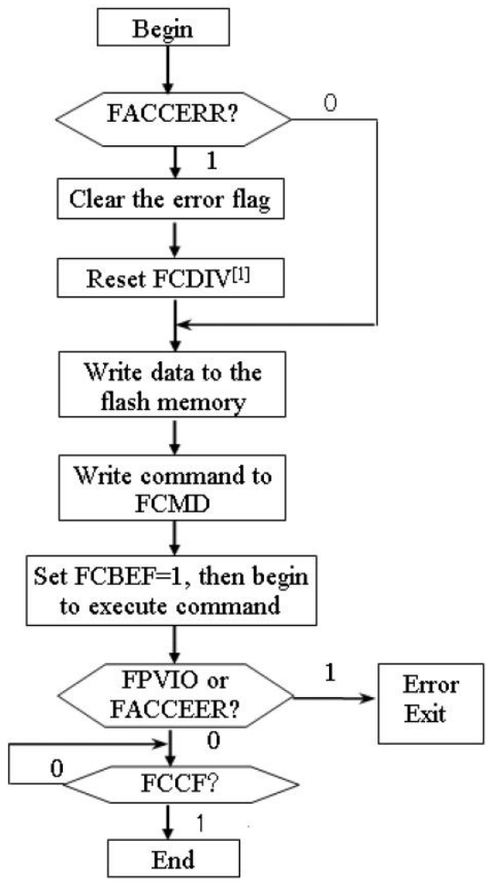


Fig. 2. AW60 Flash programming flowchart

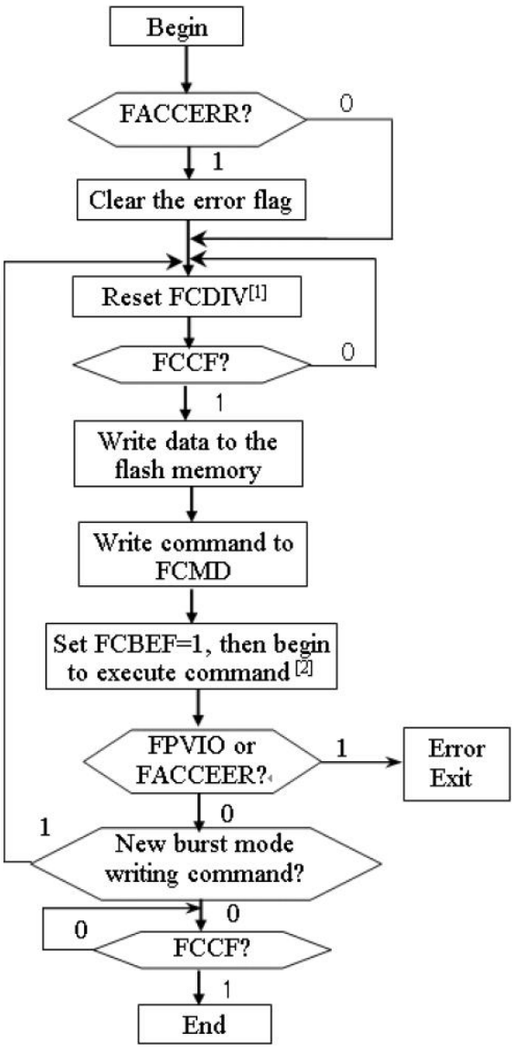


Fig. 3. AW60 Flash burst mode programming flowchart

e. Writing an invalid flash normal mode command to the FCMD register.
f. Try to operate the other registers in addition to FSTAT after finished writing command values to FCMD.
g. MCU enters into STOP mode when execute the command.
h. When MCU is in secure state, erase pages or write flash memory with background debug interface (If MCU is in encrypted state, we can only execute blank check or mass erase commands with background debug interface).
i. Aborting a command write sequence by writing 0 to the FCBEF flag.

2.2 MC9S08AW60 flash memory in-circuit programming instance

We first give the AW60's Flash programming subroutine in this section. Then put forward the Flash in-circuit programming instance in user mode. And verify the result by the serial communication mode with PC.

2.2.1 The erasing and programming c language subroutines of flash memory

For the Flash programming subroutines are not solidified in the internal monitoring ROM[4] of AW60, the initial loaded user program should contain the flash erasing and programming subroutines in order to do in-circuit programming for Flash. Because these subroutines resident in Flash, when running the erasing/programming subroutines, the whole Flash region will be added programming voltage that is higher than the normal operating voltage, which results in instable Flash region's reading, and may lead to program's error running. In order to make the erasing/programming subroutines running normally, these subroutines should be moved into RAM and run in RAM. Therefore, a buffer should be opened up in RAM to store these subroutines. The following sample program gives a convenient method to save the machine codes which are generated by erasing/programming subroutines in RAM. The machine codes include 57 bytes. Necessarily, you can directly call these codes to implement the Flash in-circuit erasing/programming. For the detailed Erasing and Programming subroutines, please refer to the program in our program directory "*..\Flash_Program\S08(AW60)-Flash*"[5]. These subroutines contain the following operations:

1. Some public operation of erasing/programming processing

For the codes of erasing/programming operation must run in RAM, so we write the C language program according to Fig.2. After being compiled, the corresponding machine code bytes are saved in the array PGM (volatile unsigned char PGM[57]). Therefore we need not copy the codes to RAM to realize the erasing/programming operations.

2. Page erasing subroutine (*Flash_PageErase*)

In this subroutine we should calculate the page's top address by the page number, and change the Flash command to erasing command 0x40, then call and execute the erasing codes.

3. Flash programming subroutine

In this subroutine we calculate the top programming address according to the page number and page offset, then change the Flash command to erasing command 0x20, and program flash one byte by one byte.

2.2.2 Programming essentials of erasing /programming subroutines

Using Flash in-circuit programming technology eliminates the need for external EEPROM, which not only simplify the circuit design, but also improve the stability of system. We compile the Flash programs and save them in Flash. When you need to use these codes, they will be copied to RAM. Just because of this special procedure, we put forward the following notes according to the experience which is accumulated in the actual programming and debugging, and project development process.

1. There are 57 bytes in RAM to store the erasing/programming machine codes, don't forget to calculate when using RAM.
2. The region that has been erased for one time and has not been programmed can be programmed by calling Flash programming subroutine again, but the region that has been programmed can't be programmed again if it hasn't been erased.

---

[4] ROM—Read Only Memory

[5] You can download the program directory "*Flash_Program*" in our website (http://sumcu.suda.edu.cn/flash.htm), which involves three Flash programming instances in three subdirectory ("*\S08(AW60)-Flash*", "\ S12X(XS128)-Flash" and "\ ColdFire(MCF52233)-Flash").

3. For we do erasing for one page (512 bytes) each time, so we should arrange the data reasonably to avoid wrong erasing.
4. The start address of the page should be defined according to the rules of the FPROT register.
5. It is invalid to do in-circuit programming for the protection block set in FPROT.

2.2.3 Validate flash memory implements

In order to more clearly understand the method of Flash memory in-circuit programming, we give a flash memory validating project. Its function is as the following: the MCU receives formatted data from PC[6] by SCI[7] tools and erases, programs or reads its flash memory. The PC software is any SCI testing tool.
Now, list some flash operating commands using the SCI debug (as shown in Table 1):

Commands	Functions
?	MCU sends some items to PC
E:8	Erase page 8
R:8:0:4	Read 4 bytes the word 0 of page 8
W:8:0:4:A,C,B,D	Write "ACBD" (4 bytes) to the word 0 of page 8
B:8,7,6,5,4,3,2,1	Set the Flash back door key, the password is "87654321"
M:8,7,6,5,4,3,2,1	Verify the Flash back door key, the password is "87654321"
D	Delete passwords
U	Encryption lift
P:8	Protect block, protect the addresses from 8 to 0xFFFF

Table 1. Flash operating commands using the SCI debug

Above examples only give the program data less than one page (512 bytes). Only slightly modify the program, you can program data that exceeds one page.

2.3 Protection mechanisms and security operations of MC9S08AW60 flash memory

2.3.1 Protection mechanisms

Being a non-volatile memory (NVM), flash memory may be used by programmers to store some important parameters and data. To prevent from erasing or programming these significant regions by accident, the MC9S08AW60 MCU supplies protection mechanisms for its flash memory. That's to say, it can't erase or program the protected region.
The Flash Protect Register (FPROT and NVPROT [1]) is interrelated with the protection mechanisms of S08 flash memory. And for the register's bit definition and programming method you can refer to the reference manual [1]. By setting this register, we can protect the Flash memory.

[6] PC—Personal Computer
[7] SCI—Serial Communication Interface

For the programming codes for setting block protection, please refer to the program in our program directory "*..\Flash_Program\S08(AW60)-Flash*"

2.3.2 Security operations

The debug module is added in the S08 series MCUs, which increased the practicality of the chip, and brought risks to the security of the chip. In order to ensure the safety of the chip, the security mechanisms are much more complex.

S08 series MCUs use hardware mechanisms to prevent unauthorized users trying to access the Flash and RAM memory data. Having been set security, Flash and RAM are all counted as secure resources. But the direct page register, high-end page register and background debugging module are all counted as unsecure resources. During executing process, we can access any memory data, but can't access secure sources by background debugging interface or unsafe method.

The security can be set by the nonvolatile data bit SEC01：SEC00 in FOPT. Table.2 gives the security state of MCU.

SEC01:SEC00	state
0:0	Secure
0:1	Secure
1:0	Unsecure
1:1	Secure

Table 2. Security state

1. Set MCU to Security Mode

To prevent the programs in the flash memory from being read out illegally, the MCU should be set in security mode. Two methods for locking the flash memory are shown in the following.

Method A. Lock the MCU by modifying the security configuration field in the file isr.c(that is, modify the values of the FOPT's address 0xFFBF and the key's address 0xFFB0~0xFFB7).

Method B. we can lock the flash memory by calling the custom subroutine Flash_Secure to modify relevant address matters when the program is running. By modify the content in the address of NVOPT, the value of this register are automatically loaded in FPOT while the system is set. For the detailed *Flash_secure* subroutine, please refer to the program in our program directory "*..\Flash_Program\S08(AW60)-Flash*"

2. Unlock from Security Mode

If we want to program locked S08 serials MCU again, we should unlock it. Here two methods are provided to unlock it.

Method A. Use the BDM interface of our writer, mass erase the locked MCU. (The writer is designed by our lab.)

Method B. Call the subroutine *Flash_KEY_Match* to erase password or flash by memory-resident program. For the detailed *Flash_KEY_Match* subroutine, please refer to the program in our program directory "*..\Flash_Program\S08(AW60)-Flash*"

3. Programming flash memory in freescale S12XS128

The flash memory in-circuit programming implement for 16bit S12XS128 MCU will be explained in this section, as follows:

3.1 How to operate S12XS128 flash memory

3.1.1 The paging mechanism and MMC module in XS128 flash

1. The paging mechanism of S12XS memory

Take XS128 of S12XS serials MCU for example, XS128 contains 8KB RAM, 8KB D-Flash and 128KB P-Flash. But the basic address line of S12XS serials MCUs is 16bits, which determine its addressing scope range from 0x0000~0xFFFF. So the size of addressing space is 2^{16}B=64KB. That is, in most case MCU can only "see" these 64KB memory space.

As shown in Fig.4, the 64KB address space in S12XS serials MCUs is divided into four parts: I/O register, data Flash memory (D-Flash, also called as EEPROM), RAM and program Flash memory (P-Flash, directly called as Flash). The I/O register region ranges from 0x0000~0x07FF (2KB). D-Flash region ranges from 0x0800~0x0FFF (2K). RAM region ranges from 0x1000~0x3FFF (12K). P-Flash region ranges from 0x4000~0xFFFF (48K).

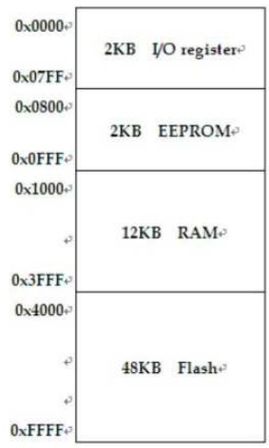


Fig. 4. S12XS's 64KB address space

In order to expand the memory space when using 16 bits address line, S12XS serials MCU integrates MMC (Memory Mapping Control) module, which expand the addressing space from 64KB (16bits) to 8MB (23bits) by using paging management mechanism.

Address analyzing and addressing are managed by the MMC module in XS128. The main functions of MMC module include address mapping, controlling the operation mode of MCU, multi-agent (MCU and BDM) priority addressing, choosing the internal resource and controlling internal bus (which include memory space and peripheral resources) etc.

When we provide a certain address, whether it is a local 16bits address or a global 23 bits address, it will be analysized by MMC and assigned automatically to PPAGE or EPAGE, then gain a remainning 16bits address so that 16bits machine can directly calculate the address space and addess. Without these registers, 16bits machine should calculate twice to deal with 23bits address. Using these registers can improve the addressing efficiency. The whole procedure is automatically completed by MMC without user's special care. Users only need to provide correct address.

There is a Global Page Index Register (GPAGE) in MMC. The highest bit of this register is fixed to 0, so GPAGE actually become a 7-bits register. MCU expands its 16 bits address to be 23 bits by means of GPAGE register. The 23 bits global address is composed of 7 bits GPAGE value [22:16] and CPU local address [15:0]. Meanwhile, specified 23 bits address read/write instructions are added in the instruction system of CPU. Only when CPU

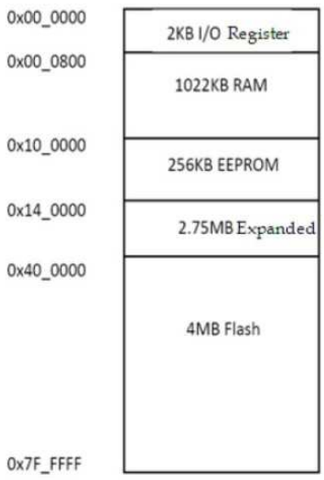


Fig. 5. S12XS's 8MB expanded address space

performs a global command GPAGE register is used. GPAGE provides a method to addressing 8MB space by using 23 bits global address. At this time the 8MB continuous addresses are distributed as Fig.5. The specific distribution is also shown as below:

0x00_0000~0x00_07FF	2KB I/O register address space
0x00_0800~0x0F_FFFF	64KB×16-2KB=1MB-2KB RAM space
0x10_0000~0x13_FFFF	64KB×4=256KB D-Flash space
0x14_0000~0x3F_FFFF	64KB×44=2816KB unused space
0x40_0000~0x7F_FFFF	64KB×64=4MB Flash space

Besides, in order to manage and use D-Flash, RAM and P-Flash, MMC adds three memory page registers: Data FLASH Page Index Register (EPAGE)[2], RAM Page Index Register (RPAGE)[2] and Program Page Index Register (PPAGE)[2], which are used to addressing corresponding expanded region. CPU opens up several windows in its 64KB addressing space. By using above page registers, CPU can map the memory space out of 64KB into these windows in 64KB space at any time. Meanwhile the window which is not used temporarily will be exchanged out. By using this method CPU can expand its addressing space. Besides, these page registers are also used to addressing the global address.

2. Paging memory mapping of XS128

For specific chip, not all the address spaces correspond to actual physical memory. For example, XS128 involves 8KB RAM, 8KB D-Flash and 128KB P-Flash. The address spaces used by these actual physical memories have been determined when chip is designed.

The global addresses of 8KB RAM in XS128 range from 0x0F_E000~0x0F_FFFF. RPAGE=0xFE~0xFF. When chip is reset, the default value in RPAGE is 0xFD, which is a invalid value. That is, addressing 0x1000~0x1FFF will make mistakes and produce illegal address interrupt. When MCU is initialized, we can initialize RPAGE to be 0xFE. So directly addressing 0x2000~0x2FFF will be same to addressing with global address 0x0F_E000~0x0F_EFFF. The global addresses of 8KB D-Flash range from 0x10_0000~0x10_1FFF, EPAGE=0x00~0x07. And the global addresses of 128KB P-Flash range from 0x7E_0000~0x7F_FFFF, PPAGE=0xF8~0xFF. Only these memory address resources mentioned above can be used in actual programming. The operation for the memory addresses outside of these addresses has no meaning.

3. The conversion of Local address, Logical address and Global address

Correctly comprehending the Local address, Logical address and Global address is the foundation to flexibly apply XS128 Flash module. In fact, for 16bits address line MCU, logical address is just the traditional 64Kb address space 0x0000~0xFFFF. Logical address is the expanded 24bits address. Its general format is 0xXX_XXXX. The two hexadecimal bits before "_" are the value of PPAGE or EPAGE (Page number). The other four hexadecimal bits behind "_" are the corresponding window address. For example, 0xFE_8000 is a logical address. 0xFE is P-Flash's page number, and 0x8000 is the P-Flash's corresponding window address in 64KB memory space. Global address is the physical space's address used to save data. The global address of XS128 has 23 bits. That is, range from 0x00_0000~0x7F_FFFF. For example, if PPAGE=0xFE, addressing any address in the local address space 0x8000~0xBFFF actually means addressing the logical address space 0xFE_8000~0xFE_BFFF, while the corresponding global address space is 0x7F_8000~0x7F_BFFF. These two addresses are equivalent, and they are only two kinds of index patterns.
The conversion of logical address and global address in P-Flash is shown as Fig.6

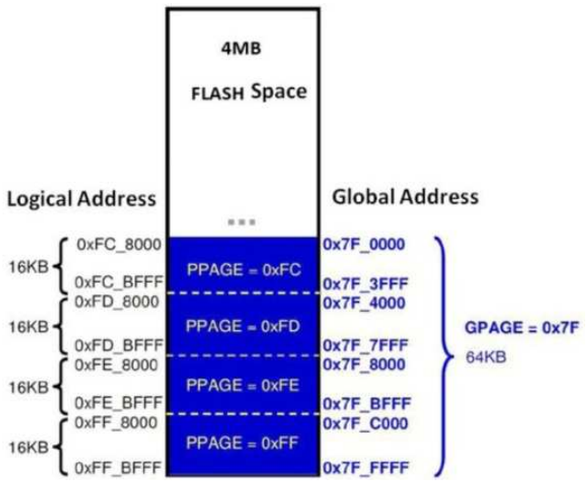


Fig. 6. The conversion of logical address and global address

As shown in Figure.6, the logical address ranges from 0xFC_8000~0xFC_BFFF. The corresponding value of PPAGE is 0xFC. When the logical address is converted into corresponding global address, the top bit [22] in the 23bits address is fixed to 1, the following 8bits [21:14] are the value of PPAGE, that is 0xFC. The lower 14bits are local address, which ranges from 0x0000~0x3FFF. So the corresponding global address ranges from 0x7F_0000~0x7F_3FFF.
On the contrary, the global address ranges from 0x7F_0000~0x7F_3FFF. The corresponding value of PPAGE is the value of [21:14] (0xFC). Bit [22]=1 means a P-Flash page's global address. The lower 16bits of logical address ranges from 0x8000~0xBFFF (P-Flash window address region). So the corresponding logical address ranges from 0xFC_8000~0xFC_BFFF.
Fig.7 provides the relation between local address and global address. The local address of P-Flash in XS128 ranges from 0x4000~0xFFFF. 0x8000~0xBFFF is P-Flash window address region. Its corresponding global address region is 0x7E_0000~0x7F_FFFF. 0x4000~0x7FFF and 0xC000~0xFFFF can directly addressing the global physical addresses (0x7F_4000~0x7F_7FFF and 0x7F_C000~0x7F_FFFF). The local address of D-Flash range from 0x0800~0x0FFF, which is used for EPAGE address mapping window. And the corresponding global address for this window range from 0x10_0000~0x10_1FFF.

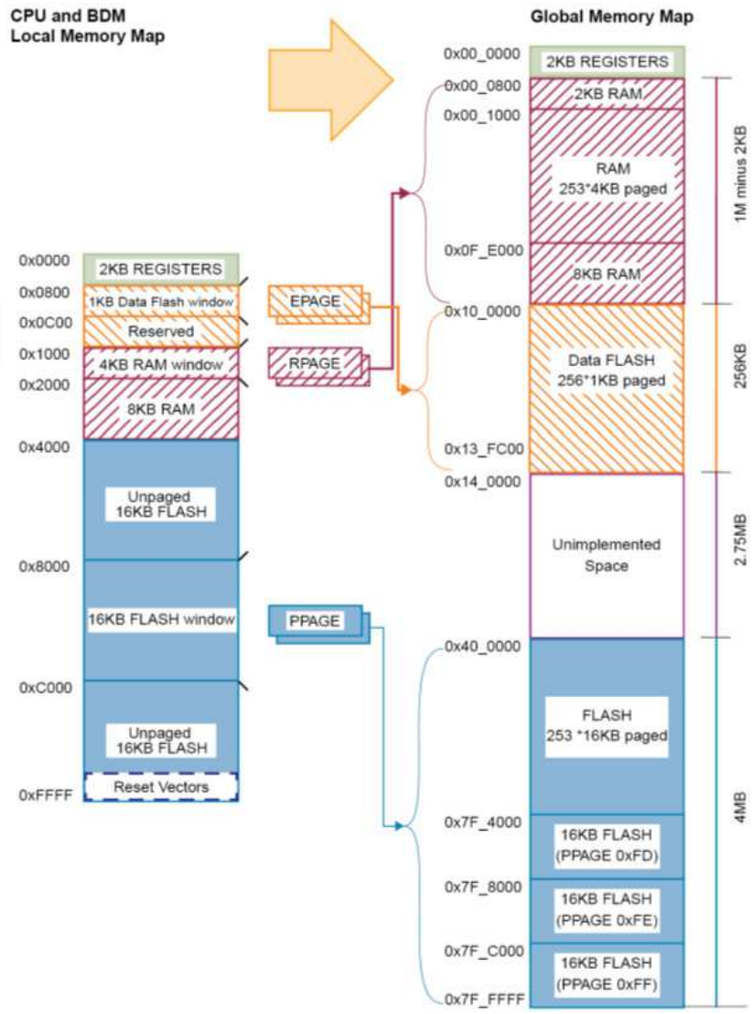


Fig. 7. The relation between the local address and global address in XS128

3.1.2 S12XS128 flash memory registers

In XS128 MCU, the relative registers for Flash programming include general registers and dedicated registers. Setting the general register can simultaneously set the characteristics of two Flash parts. While the dedicated register can only give service to a single Flash part at a certain time interval, the corresponding dedicated registers of the two Flash parts share the same address, so we should illustrate which Flash part is operated before using it.

There are 5 registers used in erasing and programming operation, which include FCLKDIV, FCNFG, FSTAT, FCCOBIX /FCCOB etc. FCLKDIV and FCNFG are general registers. FSTAT and FCCOBIX/FCCOB are dedicated registers. For the detailed function and use of these registers, please refer to the Reference Manual "*MC9S12XS256 Reference Manual*"[2].

3.1.3 XS128 special command mode NVM

For Loading of Flash commands, XS128 is different from the other Freescale MCUs (include DG128). The other MCUs mostly use a command register, which can be writen erasing/programming command codes directly. However, XS128 improve the previous

mechanism. It loads the commands and parameters by using FCCOBIX register cooperate with FCCOB register.

The essence of NVM command mode is using the indexed FCCOB register to provide a command code and relevant parameter for memory controller. Users first according to need to set up all needed FCCOB registers domain, then initialize the execution of command by setting the CCIF bit in FSTAT register. When users clear the CCIF bit in FSTAT register, all the parameters in FCCOB register will be locked, which can't be modified before the completion of command execution. (When command finished, CCIF is set to be 1).

In NVM command mode, the general command formats of FCCOB are shown as Fig.8

CCOBIX[2:0]	Byte	FCCOB Parameter Fields (NVM Command Mode)
000	HI	FCMD[7:0] defining Flash command
	LO	0, Global address [22:16]
001	HI	Global address [15:8]
	LO	Global address [7:0]
010	HI	Data 0 [15:8]
	LO	Data 0 [7:0]
011	HI	Data 1 [15:8]
	LO	Data 1 [7:0]
100	HI	Data 2 [15:8]
	LO	Data 2 [7:0]
101	HI	Data 3 [15:8]
	LO	Data 3 [7:0]

Fig. 8. The general command formats of FCCOB

Users can load commands by assigning FCCOB and FCCOBIX register according to the specific command formats. For the detailed programming method, please refer to the follow-up section which gives specific erasing/programming subroutines.

3.1.4 Flash programming procedure

In general, the erasing/programming operation of Flash involves four steps as below.

1. Set FCLKDIV register

For the detailed setting, please refer to the introduction of FCLKDIV register in the above section.

Caution: If the frequency is less than 1MHz, the Flash erasing/programming operation will be unsuccessfully. Too high setting of FDIV may damage Flash memory module. But too low setting may lead to unsuccessfully erasing and incomplete programming for Flash memory units. So users should choice appropriate Clock Divider.

2. Set the corresponding commands and parameters for FCCOB and FCCOBIX registers as needed
3. Set the CCIF bit in FSTAT register
4. Judge whether errors occur during the running of commands

Fig.9 gives a general Flash programming flowchart. According to this flowchart we can erase/program Flash successfully. We only need to pay attention to that part of Flash codes for P-FLASH operation should be moved in RAM.

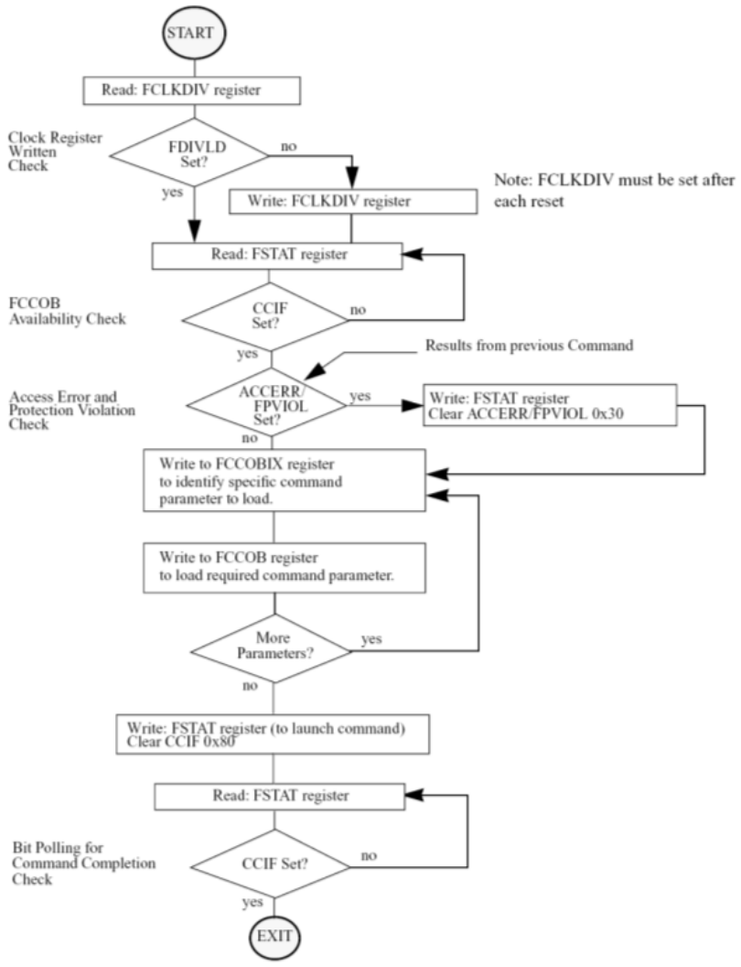


Fig. 9. A general Flash programming flowchart

3.2 XS128 D-FLASH in-circuit programming instance

We provide a D-FLASH In-circuit Programming Instance in our program directory "*..\Flash_Program\ S12X(XS128)-Flash*", which contains the following parts:

3.2.1 Preparation for D-FLASH programming

XS128 contains 8Kb D-FLASH spaces, which is divided into 8 pages (1KB/page). The minimum erasable unit in programming is a sector, which is 256 bytes. There are 32 sectors in D-FLASH. For the detailed blocking codes, please refer to the head file *EEPROM.h* in our program directory "*..\Flash_Program\ S12X(XS128)-Flash*".

3.2.2 Some common operation for erasing/programming procedure

The erasing/programming programs for D-FLASH need not run in RAM. For this kind of memory mode with multi paging mechanism, it is necessary to design a function which calculate the specific address with sector number and block number. Besides, we should set the FCLKDIV register before erasing/programming D-FLASH. And we also should detect the error flags to judge whether command run successfully.

3.2.3 Erasing subroutine

First we calculate the first address of erasing sector by sector number, load this first address and the D-FLASH sector erasing command CMD_D_ERASE_SECTOR (0x12) into FCCOB register by NVM command mode. Then execute the erasing command.

3.2.4 Programming subroutine

Calculate the first address of the programming sector by sector number and offset block number, load this first address and the D-FLASH sector programming command CMD_D_PROGRAM (0x11) into FCCOB register by NVM command mode. Then execute the programming command.

3.2.5 Reading/writing data

In regard to the reading/writing for content of Flash region, special additional remarks should be provided here.

The address space of Flash usually corresponds with multiple addresses. Here we elaborate the using method of these addresses, which apply some technique of C language.

Reading Flash can adopt the following 3 addressing patterns.

1. Addressing by Local Address. That is, addressing through 64KB address space. The addresses range from 0x0000~0xFFFF.
   For example: **Data= *(volatile uint8 *)0x0400;**
2. Addressing by Logical Address (Global Logical Address). The addresses cooperated by EPAGE range from 0x0800~0x0c00, which can be addressed by the format "__eptr". Caution: "__eptr" includes two underlines.
   For example: **Data= *(volatile uint8 * __eptr)0x00_0800;**
3. Addressing by Global address (Global Physical Address). According to the actual physical location of the whole memory, access the memory with the format "__far". Caution: "__far" includes two underlines.
   For example: **Data= *(volatile uint8 * __far)0x10_0000;**

Caution: A sector is the minimum unit to erase. For D-FLASH, the minimum size is 256 bytes.

3.3 XS128 P-FLASH in-circuit programming instance

XS128 contains 128Kb P-FLASH spaces, which is divided into 8 pages (16KB/page). The minimum erasable unit in programming is a sector, which is 1024 bytes. There are 128 sectors in P-FLASH. The programming procedure of P-FLASH is similar to that of D-FLASH. So we omit the detailed description for this P-FLASH programming instance. For the detailed program codes, please refer to the program in our program directory "*..\Flash_Program\ S12X(XS128)-Flash*".

3.4 Protection mechanisms and security operations Of XS128 flash memory

3.4.1 Protection mechanisms

The registers relate with XS128 Flash's protection mechanisms include FPROT (Flash Protection Register) and DFPROT (D-Flash Protection Register). After set the protection registers, the protected region can't be erased or programmed.

3.4.2 Security operations

The debugging module in XS128 improves the practical applicability of MCU, but simultaneously brings about hidden danger to the security of MCU. The common users may

easily steal the programs from MCU by BDM. In order to prevent software piracy, XS128 brings in complex security mechanism to guarantee the security of MCU. When the MCU is encrypted, the common users can't read any content in memory by BDM[8] (Only messy codes can be read.) But the programs running in MCU can access arbitrary resources of MCU, and can decrypt MCU by using the back-door key access mechanism provided by MCU.

1. Set MCU to Security Mode

To prevent the programs in the flash memory from being read out illegally, the MCU should be set in security mode. The corresponding register is FSEC (Flash Security Register). If it is reset, FSEC register automatically load value from the configuration address 0x7F_FF0F. All bits of FSEC are related to the security of device, and these bits are read-only.

2. Unlock from Security Mode

a. Can't unlock by BDM

As manual states we can't unlock MCU by BDM with backdoor key access mechanism. Facts also show that we can't unlock MCU and obtain valid data by BDM. It is worth noting that we can entirely erase the locked MCU by BDM, while the flag bit FPVIOL of FSTAT register will be set. If we don't want to secure MCU now, we should program immediately by changing the later two bits of the byte in 0x7F_FF0F as the value 1:0. So after the next reset MCU will be in unlocked state.

b. The only way to unlock MCU—using backdoor key access mechanism.

Programs like buried treasure locked in chip. Treasure pretenders have tried every means to get it, but they are always blocked by an indestructible security door. Only intelligent master owns the key to open this security door. This is the so-called backdoor key access mechanisms.

How to start and use this kind of mechanism?

First, 8 bytes backdoor key together with the programs should be programmed into MCU. That is, 8 bytes key should be successively programmed into the addresses 0x7F_FF00~0x7F_FF07.

After that, the bits KEYEN[1:0] of FSEC register should be set as the value 10 to enable the backdoor key access mechanism.

Concerning how to unlock:

First, prepare to match the key. This step will use the FLASH backdoor key comparison command 0x0C. The backdoor key comparison command and 8 bytes key can be set to FCCOB. And setting flag bit CCIF can enable the comparison. If the comparison is successful, the security state will temporarily be unlocked. If the comparison is unsuccessful, the next comparison can be done only after reset, otherwise none operation can be done. Besides, if the comparison is successful, SEC[1:0] will be 10 which means unlocked state. If at this time users want to disable the encryption function, the bits KEY[1:0] should be set as disabled state to disable the backdoor key comparison function.

4. Programming flash in freescale MCF52233 flash

The flash memory in-circuit programming implement for 32bits MCF52233 MCU will be explained in this section, as follows:

[8] BDM-- Background Debug Monitor

4.1 How to operate coldfire flash memory

4.1.1 The basic concepts of MCF52233 flash memory

ColdFire Flash Module (CFM) is made up of 4 arrays, and each consists of 32K*16 bits, thus composing a flash memory space of 256 Kbytes, as is shown in Fig.10. Inner flash controller needs 2 cycles to access to the flash memory, but since across accessing enabled, it can read the flash consecutively with a higher frequency. Only one cycle is needed for reading each word.

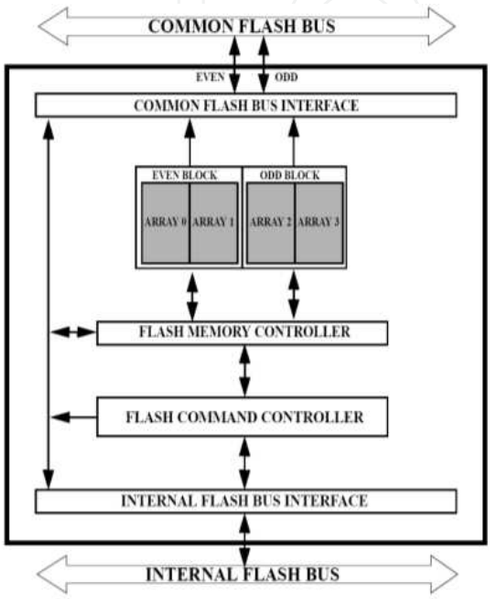


Fig. 10. CFM Block Diagram

In MCF52233, the 256KB flash memory space is divided into 32 8KB sectors. Each section has 4 pages and each page is of 2KB. When programming, note that the erase is carried out by page. That is to say, at least one page needs to be erased at a time. 2 words (4 bytes) are performed at a time.

The 32-bit MCF52233 has 32 address buses, and can address 4GB space. In principle, the initial address of MCF52233 is alterable. By setting the corresponding register, the 256KB 32-bit flash can be located to any continuous space. However, in practice its start address is set to 0x0000_0000. And it is suggested not to alter the address.

4.1.2 ColdFire flash memory registers

Erasing and programming relate to registers such as FLASHBAR, CFMCLKD, CFMMCR, CFMPROT, CFMSEC, CFMUSTAT and CFMCMD. For the detailed function and use of these registers, please refer to the Reference Manual "***MCF52235 ColdFire integrated*** *Microcontroller Reference Manual*"[3].

4.1.3 ColdFire flash memory erase and program implements

For ColdFire MCU, the entire flash memory or only one page (2KB) at the start address can be erased. That is, more than one byte or 2KB is erased at a time. To perform, a row of data should be prepared and put into the RAM first. Only after erasing the corresponding region in Flash memory can perform be carried out. Furthermore, the erasing or performing of any

byte influences the page it is in, so before that it is necessary to arrange relevant data in the erasing region by linking files. In other words, the page which is being programmed cannot be erased. Below is a detailed procedure of Coldfire Flash memory erase and program. The corresponding sub-program instances are also provided in our program directory "*..\Flash_Program\ ColdFire(MCF52233)-Flash*".

1. Common Operations for Erase and Program
   a. If the CFMCLKD register is written, the DIVLD bit is set automatically. If the DIVLD bit is 0, the CFMCLKD register has not been written since last reset. No command can be executed if the CFMCLKD register has not been written.
   b. Before starting a command write sequence, the ACCERR and PVIOL flags in the CFMUSTAT register must be cleared.
2. Erase

Step 1. set the clock frequency division by writing the CFMCLKD register. Clear error flags, and set the sector number. These operations take place at the beginning of all operations, and have been packaged into a subroutine which can be called directly.
Step 2. locate the sector to be erased. Write a value to any location in that sector.
Step 3. write 0x40 to command register CFMCMD (section 10.2.1 "CFM Registers").
Step 4. write a "1" to the command buffer empty interrupt flag (CBEIF) of register CFMUSTAT. This clears the flag and launches the flash command described in step three.
Step 5. wait for the command to be accomplished. This is indicated by the command complete interrupt flag (CCIF), which is also located in status register CFMUSTAT. This bit is set when the command is completed.

3. Program

If we need to write some words to a specific start address in flash memory (note: the address should be clean—non-written), detailed steps are as follows.

Step 1. is the same as in the erasing operation.
Step 2. set the start address. The process of writing words is then divided into sub-steps as follows:
   Step A. select a word (provide the source address and the target address).
   Step B. Step B, write 0x20 to command register CFMCMD (section 10.2.1 "CFM Registers").
   Step C. write a "1" to CBEIF in register CFMUSTAT, clearing the flag bit and executing the flash command.
   Step D. wait for the command to be accomplished (the CBEIF flag of register CFMUSTAT is 1), meanwhile the next command is receivable only.
   Step E. if data remain to be written, increase the source and target addresses then go to step B.

Notes: the register CFMCLKD is set only once anterior erase operation and in no any program case. Don't erase any region which stores codes.

4. Flash Memory Illegal Operations
   a. Writing to the flash memory before initializing CFMCLKD; Writing to the flash memory while CBEIF is not set; Writing to a flash block with a data size other than 32 bits; After writing to the even flash block, writing an additional word to the flash memory during the flash command write sequence other than the odd flash block; Writing an invalid flash normal mode command to the CFMCMD register (out of the 5 values); Writing to any CFM register other than CFMCMD after writing to the flash memory; Writing a second command to the CFMCMD register

before executing the previously written command;. Writing to any CFM register other than CFMUSTAT (to clear CBEIF) after writing to the command register, CFMCMD; entering stop mode with some commands uncompleted. Upon entering STOP mode, any active command is aborted; Aborting a command write sequence by writing a 0 to the CBEIF flag after writing to the flash memory or after writing a command to the CFMCMD register but before the command is launched.

b. The PVIOL flag is set during the command write sequence if any of the following illegal operations are performed, causing the command write sequence to immediately abort: Writing a program command if the address to program is in a protected flash logical sector; Writing a page erase command if the address to erase is in a protected flash logical sector; Writing a mass erase command while any protection is enabled. If a read operation is attempted on a flash logical block while a command is active on that logical block (CCIF=0), the read operation returns invalid data and the ACCERR flag in the CFMUSTAT register is not set.

For predigesting programming, various illegal operation types listed above are ignored in practice and are simply classified as: completed or aborted.

4.2 Validate ColdFire flash memory implements

The validate ColdFire flash memory application in our network site is as the following: the MCU receives formatted data from PC by SCI interface and erases, programs or reads its flash memory. The PC software is SCI debug or our testing tool. For the detailed codes and running windows, please refer to the program in our program directory "*..\Flash_Program\ ColdFire(MCF52233)-Flash*".

Now, list some flash operating commands using the SCI debug:

Commands	**Functions**
?	**MCU sends some items to PC**
E:8	**Erase page 8**
R:8:0:4	**Read 4 bytes the word 0 of page 8**
W:8:0:4:A,C,B,D	**Write "ACBD" (4 bytes) to the word 0 of page 8**
P:8,7,6,5,4,3,2,1	**Encrypt Flash and the password is "87654321"**
D	**Delete passwords**

Above examples only give the program data less than one page (2048 bytes). Flash memory application for data that exceeds one page can been found in the aforesaid network site.

4.3 CFM protection mechanisms and security operations

4.3.1 CFM protection mechanisms

The CFMPROT register (refer to the reference manual[3]) is interrelated with the protection mechanisms of ColdFire flash memory which is divided to 32 sectors and each controlled by a flag of CFMPROT—the sector is presumed as in protected state while the corresponding flag is set to 1—there will be Illegal when erasing or programming the sector. Note to get it back to the protected state after erasing or programming the sector. In erase subroutine *Flash_Page_Erase* and program subroutine *Flash_Page_Write*, the Flash_Protect(page,FALSE) releases the sector from the protected state, but the MCF_CFM_CFMPROT = 0xffffffff reverses that.

4.3.2 CFM security operations

The ColdFire 0x0400~0x0417 is the flash configuration field whose security word is read automatically after each reset and is stored in the CFMSEC register. If the low 2 bytes of the

CFMSEC register offset (0x0414~0x0417) in the file vectors.s is equal to 0x4ac8, the MCU is in its security mode and programs in the flash memory can't be read, erased or programmed by 32-bit ColdFire programming writer in the BDM mode. Whereas it allows the password matching while the high 2 bytes is 0xc000. If the 32-bit ColdFire programming writer is set in JTAG mode, the password can be released by erasing the page 0 (the programs in the flash memory can't be used any longer), and then the flash memory can erase or program in the BDM mode again.

1. Set MCU to Security Mode

To prevent the programs in the flash memory from being read out illegally, the MCU should be set in security mode. Two methods for locking the flash memory are shown in the following.

Method A. Lock the MCU by modifying the security configuration field in the file vectors.s.

Method B. we can lock the flash memory by calling the custom subroutine Flash_Secure to modify relevant address matters when the program is running. For the locked subroutine, please refer to the program directory "*..\Flash_Program\ ColdFire(MCF52233)-Flash*".

2. Unlock from Security Mode

We must unlock the MCU first then can write into the program if it has been locked, because locked ColdFire family can't be mass erased by BDM. And here two methods are provided to unlock it.

First, after setting the writer into JTAG mode, mass erase the locked MCU. Refer to "32-bit ColdFire writer" in our network site (*http://sumcu.suda.edu.cn*) for details.

Second, call the subroutine Flash_Delete_Key to erase password or flash by memory-resident program. (the subroutine Flash_Delete_Key is shown in the program directory "*..\Flash_Program\ ColdFire(MCF52233)-Flash*")

5. Reference

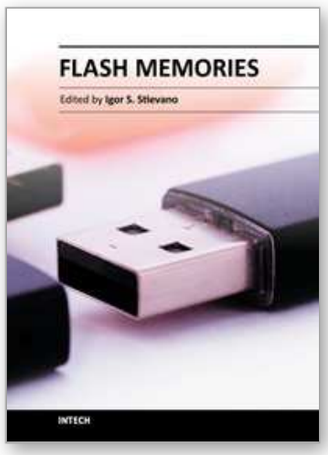
[1] Freescale: MC9S08AW60 Data Sheet ,Rev.2,2006

[2] Freescale: MC9S12XS256 Reference Manual, Rev. 1.09, 2009

[3]Freescale: MCF52235 ColdFire integrated Microcontroller Reference Manual,Rev.4, 2007

[4] WANG Yi-huai, LIU Xiao-sheng, Embedded systems-design and application on HCS12 MCUs, Beihang, University Press, 2008

[5] Yihuai Wang ,Zhigui Lin. Stable In circuit Programming of Flash Memory in Freescale's MC9S12 MCU family. Proceedings–ICMTMA2010. Volume III:477-480 . IEEE Computer Society,2010



Flash Memories

Edited by Prof. Igor Stievano

ISBN 978-953-307-272-2
Hard cover, 262 pages
Publisher InTech
Published online 06, September, 2011
Published in print edition September, 2011

Flash memories and memory systems are key resources for the development of electronic products implementing converging technologies or exploiting solid-state memory disks. This book illustrates state-of-the-art technologies and research studies on Flash memories. Topics in modeling, design, programming, and materials for memories are covered along with real application examples.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yihuai Wang and Jin Wu (2011). Programming Flash Memory in Freescale S08/S12/CordFire MCUs Family, Flash Memories, Prof. Igor Stievano (Ed.), ISBN: 978-953-307-272-2, InTech, Available from: http://www.intechopen.com/books/flash-memories/programming-flash-memory-in-freescale-s08-s12-cordfire-mcus-family

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen