Freescale Semiconductor
Application Note

Document Number: AN3942
Rev. 0, 12/2009

Flash Programming Routines for the HCS08 and the ColdFire (V1) Devices

by: Pavel Krenek
Application Engineering
Roznov CSC, Czech Republic

Contents

1 Introduction

This application note describes flash programming routines for the HC9S08 and the ColdFire V1 family MCUs. These programming routines can be used to program and erase the flash memory. Because of differences between the 8-bit and 32-bit MCUs (flash module and address space) two software versions were created:

- The first version is for 8-bit MCUs. The HC9S08 family
- The second version is for 32-bit MCUs. The ColdFire V1 (MCF51JM, MCF51QE, MCF51AC, MCF51EM, and MCF51CN)

This application note describes how to call each routine in the user software, performance, and return confirmation of the routine execution. The software files are available in the zip file AN3942SW, on the Freescale Semiconductor website, www.freescale.com.

There are basic structures of the flash memory on the HCS08 and ColdFire V1 MCUs. The flash memory is divided into several smaller memory blocks that can be erased. These blocks are the smallest possible erasable areas. The size of these blocks depend on the individual implementation of the MCU families. For example the HCS08JM60 has 512 bytes and the MCF51JM128 has 1024 bytes block size.

The most important part of the program is the correct location of the code sequence that executes the main flash programming.This code cannot run from the same flash memory because the flash module cannot write and read simultaneously.

For this reason the function doonstack that copies the relevant program sequence to the RAM memory was implemented. The code starts here and is executed.

2 API Functions

This section describes the structure and behavior of the API functions. These functions are available for the user and should be implemented in your main program. The API functions are described in detail in following chapter. Two different types of flash programming software was created for the API functions. These functions can be found in files doonstack.h.

2.1 HCS08 Version

- `FlashErase(const unsigned char * flash_destination);`
- `FlashProg(const unsigned char * flash_destination, unsigned char data);`
- `FlashProgBurst(const unsigned char * flash_destination, unsigned char * ram_source, unsigned char length);`

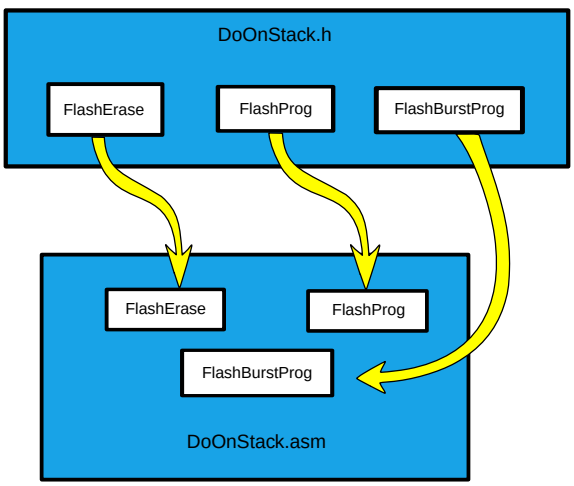

Figure 1. Architecture of API functions for HCS08 MCUs

2.1.1 FlashErase

Syntax:

- `void FlashErase(const unsigned char * flash_destination);`

Parameters:

- flash_destination—This pointer shows the block address that will be erased

Description:

- This function provides erasing the required memory block. The first address of the erasing block is shown by the pointer "flash destination."

2.1.2 FlashProg

Syntax:

- `void FlashProg(const unsigned char * flash_destination, unsigned char data);`

Parameters:

- flash_destination—This pointer shows the block address that will be programed
- data—Here, variable data is saved that can be programed to the flash memory. The maximum length of the data is 1 byte.

Description:

- This function provides programming only one byte of memory block.

2.1.3 FlashBurstProg

Syntax:

- `unsigned char FlashProgBurst(const unsigned char * flash_destination, unsigned char * ram_source, unsigned char length);`

Parameters:

- flash_destination—This pointer shows the first address of the memory block that can be programmed
- ram_source—The source of the data array that is programmed to the flash memory
- length—Length of the programed data array

Description:

- The burst program function can be used to program a block of flash memory while crossing row boundaries within the flash array. This command has a 50% faster programming time than the basic program command.
- The burst command saves the flash memory because the flash module starts only at the beginning of each cycle and is switched off at the end of this cycle.

2.2 ColdFire Version

These functions can be found in the files doonstack.h for the HCS08, and doonstack_CFV1.h for the ColdFire MCUs.

- `PageErase(unsigned int * flash_destination);`
- `BurstProg(unsigned int * flash_destination, unsigned int * ram_source, unsigned char length);`

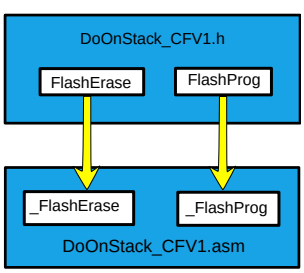

Figure 2. Architecture of API functions for ColdFire MCUs

2.2.1 FlashErase

Syntax:

- `void FlashErase(unsigned int * flash_destination);`

Parameters:

- flash_destination—This pointer shows the block address that can be erased

2.2.2 FlashProg

Syntax:

- `void FlashProg(unsigned int * flash_destination, unsigned int * ram_source, unsigned char length);`

Parameters:

- flash_destination—This pointer shows the block address that can be programed
- ram_source—The source of the data array that can be programmed to the flash memory
- length—Length of the programed data array

3 Adding the Flash Driver to the Application

This section shows implementation of the driver files to the main application.

3.1 HCS08 Implementation

- Copy the files doonstack.asm, doonstack.h, and doonstack.inc to the Sources directory for the project you are using.
- Add the doonstack.asm, doonstack.h, and doonstack.inc driver files to the project.
- Add the line #include "doonstack.h" to the main application program file.
- Add the relevant function (FlashProgBurst, FlashErase, or FlashProg) to the main application.

```
char source_data[]={0x54, 0x45, 0x53, 0x54, 0x49, 0x4E, 0x47, 0x20, 0x58, 0x44}

void Clock_Init(void)
{      // this is example of clock initialization for JM60
  MCGC2 = 0x00;
  MCGC1 = 0x06;
  MCGC3 = 0x01;

  /* bus clock 10 MHz */
 while(!MCGSC_LOCK){} /* wait until FLL is locked*/
}

void Flash_Clock_Init(void)
{
 FCDIV = 0x49;        // flash clock must be set up in the range (150-200 kHz)
}
void main(void)
{
 Clock_Init();
 Flash_Clock_Init();

  length_data = 10;    // the length of the data is 10 bytes
  adress = 0x2000;     // the source data will program to this address

 DisableInterrupts;
  FlashErase(address);  // erase the flash block
  FlashProgBurst(address, source_data, length_data)
 EnableInterrupts;
}
```

3.2 ColdFire Implementation

- Copy the files doonstack_CFV1.asm, doonstack_CFV1.h, doonstack_CFV1.inc, and doonstack_CFG.inc to the Sources directory for the project you are using.
- Add the doonstack_CFV1.asm, doonstack_CFV1.h, doonstack_CFV1.inc, doonstack_CFG.inc, and doonstack_CFG.inc driver files to the project.
- Add the line #include "doonstack_CFV1.h" to the main application program file.
- Define the target MCU and parameter size in the file doonstack_CFG.inc. You can choose between two types of MCU groups. The first group includes the MCF51(JM, QE, AC, and CN) and the second group the MCF51(128 and 256). These two groups were created for different internal flash structures. For example, the MCF51EM256 MCU has two separate flash blocks (2 x 128 Kbytes) with two groups of flash module registers. This problem was solved by using a conditional compilation.

```
; MCU = 1 -- MCF51(JM,QE,AC,CN)
; MCU = 2 -- MCF51(128(2x64kB),256(2x128kB))

MCF51_JM_QE_CN_AC .EQU 1
MCF51EM .EQU  2
MCU   .EQU   MCF51_JM_QE_CN_AC
SIZE  .EQU   256
```

- Add the relevant function (FlashErase, FlashProg) to the main application.

```
int source_data[]={0x54454554, 0x45454545, 0x54545312, 0x54545454, 0x45445459, 0x4745457E,
 0x54545447, 0x64545620, 0x24545158, 0x74545544};

void Clock_Init(void)
{      // this is example of clock initialization for JM60
  MCGC2 = 0x00;
  MCGC1 = 0x06;
  MCGC3 = 0x01;
  MCGC4 = 0x02;
  /* bus clock 24 MHz */
 while(!MCGSC_LOCK){} /* wait until FLL is locked*/
}
void Flash_Clock_Init(void)
{
 FCDIV = 0x4E;        // flash clock must be set up in the range (150-200 kHz)
}

void main(void)
 {
  Clock_init();        // initialization of  clock source on JM128
  Flash_Clock_Init();  // initialization of  flash clock frequency

  length_data = 10;    // the length of the data is 10 bytes
  adress = 0x2000;     // the source data will program to this address

  DisableInterrupts;
   FlashErase(address);  // erase function
   FlashProg(address, source_data, length_data); //program function
  EnableInterrupts;

 }
```

4 Application Example

This section discusses several examples that demonstrate how programming and erasing operations are performed on the HCS08 flash and ColdFire MCUs. All source code is written in assembler for minimum flash occupation. The source code for the HCS08 is about 350 bytes and for ColdFire about 900 bytes.

4.1 HCS08 Flash Routines and ColdFire Flash Routines

In the following code blocks the flash programming routines for erase and burst programming are shown. These short functions are situated permanently in the flash memory and serve for reading the address and setting flash parameters. These functions are executed before programing and erasing.

```
FlashErase:
        ; see Errata: SE133-FLASH : Unexpected Flash Block Protection Errors
STA     ,X                  ;latch the unprotected address from H:X
NOP           ;brief delay to allow the command state machine to start
STA     ,X    ;intentionally cause an access error to abort this command

        psha                          ;adjust sp for DoOnStack entry
lda     #(mFPVIOL+mFACCERR) ;mask
sta     FSTAT               ;abort any command and clear errors
lda     #mPageErase         ;mask pattern for page erase command
bsr     DoOnStack           ;finish command from stack-based sub
ais     #1                  ;deallocate data location from stack
rts
```

Before every program cycle there must be a flash block that can be programmed to completely erase the flash memory.

```
FlashProgBurst:
        pshx                ;save source address - low byte
        pshh                ;save source address - high byte
        psha                ;save length of data
lda     #(mFPVIOL+mFACCERR) ;mask
sta     FSTAT               ;abort any command and clear errors
ldhx    #SpSubEndBurst      ;point at last byte to move to stack;
SpMoveLoopBurst:
        lda     ,x          ;read from flash
psha                        ;move onto stack
aix     #-1                 ;next byte to move
cphx    #SpSubBurst-1       ;past end?
bne     SpMoveLoopBurst     ;loop till whole sub on stack
tsx                         ;point to sub on stack
tpa                         ;move CCR to A for testing
and     #$08                ;check the I mask
bne     I_setBurst          ;skip if I already set
sei                         ;block interrupts while FLASH busy
jsr     ,x                  ;execute the sub on the stack
cli                         ;ok to clear I mask now
bra     I_contBurst         ;continue to stack de-allocation
I_setBurst:
        jsr     ,x                  ;execute the sub on the stack
I_contBurst:
       ais     #SpSubSizeBurst+3   ;deallocate sub body + H:X + command ;H:X flash pointer OK
 from SpSub
rts                         ;to flash where DoOnStack was called
```

These functions are copied to the RAM memory before every programing and erasing cycle. These functions serve for starting the flash module and mainly flashing procedures.

```
SpSub:
        ldhx    LOW(SpSubSize+4),sp ;get flash address from stack
sta     0,x                 ;write to flash; latch addr and data
lda     SpSubSize+3,sp      ;get flash command
sta     FCMD                ;write the flash command
lda     #mFCBEF             ;mask to initiate command
sta     FSTAT               ;[pwpp] register command
nop                         ;[p] want min 4~ from w cycle to r
ChkDone:
        lda     FSTAT               ;[prpp] so FCCF is valid
lsla                        ;FCCF now in MSB
bpl     ChkDone             ;loop if FCCF = 0
SpSubEnd:
```

```
        rts                         ;back into DoOnStack in flash
SpSubSize: equ (*-SpSub)


SpSubBurst:
lda     FSTAT       ;check FCBEF
and     #mFCBEF     ;mask it
beq     SpSubBurst          ;loop if not empty
ldhx    LOW(SpSubSizeBurst+4),sp ;get source address from stack
lda     0,x                 ;load source data byte
aix     #1                  ;increment source address
sthx    (SpSubSizeBurst+4),sp ;save new source address to stack
ldhx    LOW(SpSubSizeBurst+8),sp ;get destination address from stack
sta     0,x                 ;write to flash Latch
aix     #1                  ;increment destination address
sthx    (SpSubSizeBurst+8),sp ;save new destination address to stack
lda     #mBurstProg         ;load Burst program command
sta     FCMD                ;write the flash command
lda     #mFCBEF             ;mask to initiate command
sta     FSTAT               ;[pwpp] register command
nop                         ;[p] want min 4~ from w cycle to r.
lda     FSTAT               ;load FSTAT to check ERRORs
and     #$30                ;check only FPVIOL and FACCERR
beq     FlashWriteOk
        lda     #255                ;set up error flag
        rts                         ;back into FlashProgBurst in flash
FlashWriteOk:
dbnz    SpSubSizeBurst+3,sp,SpSubBurst
ChkDoneBurst:
lda     FSTAT               ;[prpp] so FCCF is valid
lsla                        ;FCCF now in MSB
bpl     ChkDoneBurst        ;loop if FCCF = 0
clra
SpSubEndBurst:
        rts                         ;back into DoOnStack in flash
SpSubSizeBurst: equ (*-SpSubBurst)
```

4.2 Examples of Memory Allocation in Linker Files

In the following code block the flash memory allocation in the linker command file LCF in the MCF51JM128 is shown.

```
// EXAMPLE OF FLASH ALLOCATION IN lcf FILE ON MCF51JM128
#Memory ranges

 MEMORY{
   vectors  (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000200
   code     (RX)  : ORIGIN = 0x00000410, LENGTH = 0x0003FBEF
   userram  (RWM) : ORIGIN = 0x00800000, LENGTH = 0x00003FFF
 }
```

In the following code block the flash memory allocation in the linker file PRM in the HCS08JM60 is shown.

```
// EXAMPLE OF FLASH ALLOCATION IN PRM FILE ON HCS08JM60

SECTION
        ROM = READ_ONLY 0x1960 TO 0xFFAD
END
```

5 References

For more information, see the devices Reference Manual and the documentation lists in the following table.

Table 1. References

Document	Title
HCS08RM	M68HCS08 Microcontrollers Reference Manual
CFPRM	ColdFire® Family Programmer's Reference Manual
MCF51QE128RM	MCF51QE128 ColdFire® Integrated Microcontroller Reference Manual
MCF51CN128RM	MCF51CN128 ColdFire® Integrated Microcontroller Reference Manual
MCF51EM256RM	MCF51EM256 ColdFire® Integrated Microcontroller Reference Manual
MCF51JM128RM	MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3942
Rev. 0, 12/2009