

32-bit microcontrollers

Embedded FLASH for HC 32F 460 Series

Applicable objects

F Series	HC 32F460
----------	-----------

Table of Contents Table of Contents

1	Abstract	3
2	FL ASH Introduction.....	3
3	FL ASH for HC32F460 Series	4
	3.1 Introduction	4
	3.2 Description.....	4
	3.2.1 Register Introduction	4
	3.2.2 Workflow Introduction.....	5
	3.2.3 One Time Programmable Byte (OTP)	11
	3.2.4 Guided exchange.....	11
4	Sample Code	13
	4.1 Code Introduction	13
	4.2 Code Run	14
5	Version Information & Contact	15

1 Abstract

This application note introduces how to read and write data using the embedded FLASH of HC32F460 series chip.

2 FLASH Introduction

What is FLASH?

The FLASH interface provides access to FLASH through the FLASH ICODE, DCODE, and MCODE buses, which provide access to

FLASH performs programming, erase and full-erase operations; accelerates code execution through a caching mechanism.

What are the important features of FLASH?

FLASH read, program, erase and full erase operations, boot swap support, security protection and data encryption.

3 FLASH of HC32F460 Series

3.1 Introduction

The FLASH interface provides access to FLASH via the FLASH ICODE, DCODE, and MCODE buses, which provide access to

FLASH performs programming, erase and full-erase operations; accelerates code execution through instruction prefetch and cache mechanisms.

3.2 Description

FLASH read, program, sector erase and full erase operations.

CODE bus 16Bytes prefetch, 64 caches (128bit wide) shared on I_CODE and D-CODE

buses. Supports FLASH low power reads.

Support for guided exchange.

Support security protection and data encryption.

The capacity is 512Kbytes (32bytes of which are functionally reserved), divided into 64 sectors of 8KBytes each, programmed in 4Bytes and erased in 8KBytes.

128bit wide data readout.

The OTP (One Time Program) area consists of 1020 bytes, divided into 960 bytes of data area with 60 bytes of latch area.

3.2.1 Register Introduction

- 1) EFM_FAPRT: Access the EFM register protection register.
- 2) EFM_FSTP: FLASH stop register.
- 3) EFM_FRMC: Read mode register. It can be configured to insert wait period, cache function, prefetch finger function, etc.
- 4) EFM_FWMC: Erase mode register. Configure the programming erase mode.
- 5) EFM_FSR: Status register. View FLASH status, end flags, error flags, etc.
- 6) EFM_FSCLR: Status clear register.
- 7) EFM_FITE: Interrupt permit register. Configure end-of-operation or error interrupt

permission.

- 8) EFM_FSWP: Boot swap status register. This register is used to determine whether the program is reset from sector 0 or sector 1.

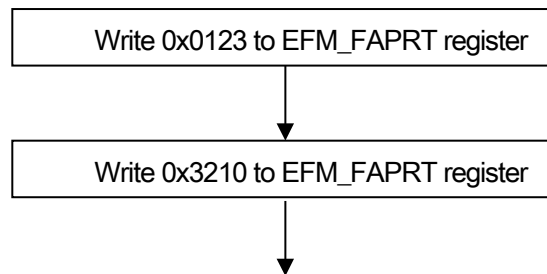
Start.

- 9) EFM_FPMTSW: FLASH window protection start address register.
- 10) EFM_FPMTEW: FLASH window protection end address register.
- 11) EFM_UQID1~3: Unique ID registers.

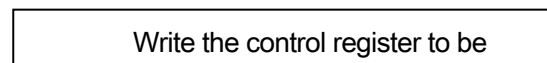
3.2.2 Workflow Introduction

3.2.2.1 Register unprotected and write protected

The registers of this module are protected by the EFM_FAPRT register, and when in the protected state, block the normal write operations. The steps to unprotect are as



follows:



In the unprotected state, write any value to the EFM_FAPRT register, and the EFM register will enter the protected state again. Caution:

- In the actual application, if you find that the register value is not written successfully, you should first check whether the EFM register access protection is valid, when the protection is valid, the EFM_FAPRT register value is read out as 0x00000000.

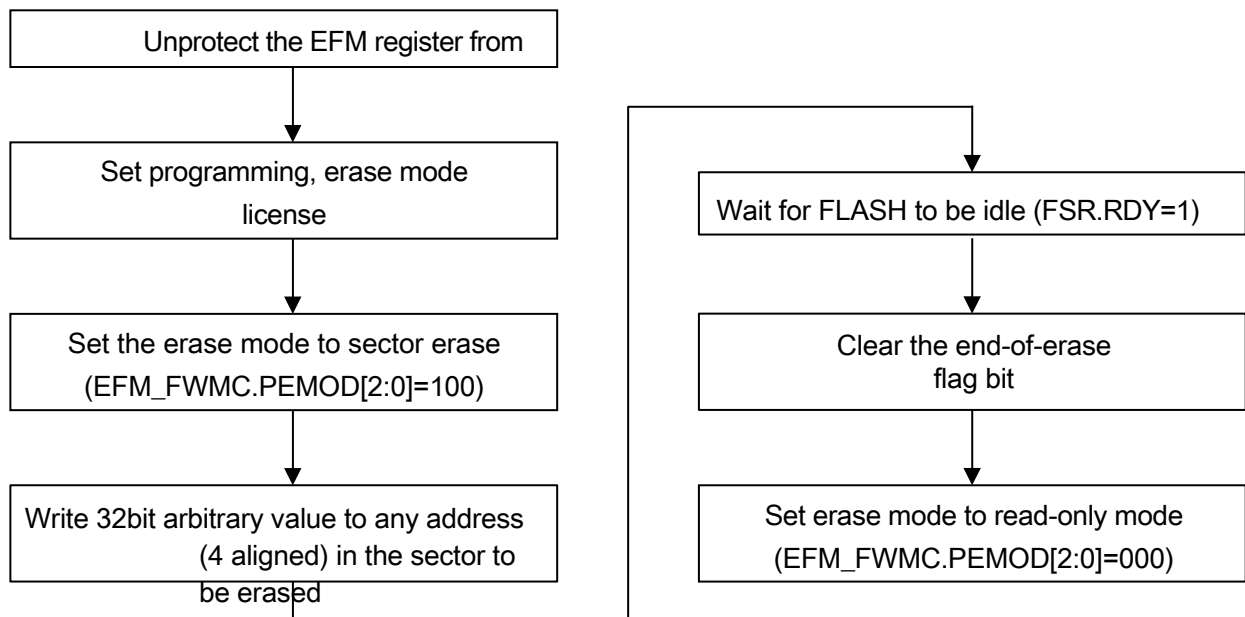
3.2.2.2 Sector Erase

EFM provides two types of erasure methods, sector erase and full erase. After a sector erase operation is performed on FLASH, the address of the sector

(8Kbytes of space) Data is flushed

to full 1. The sector erasure is set

up as follows:



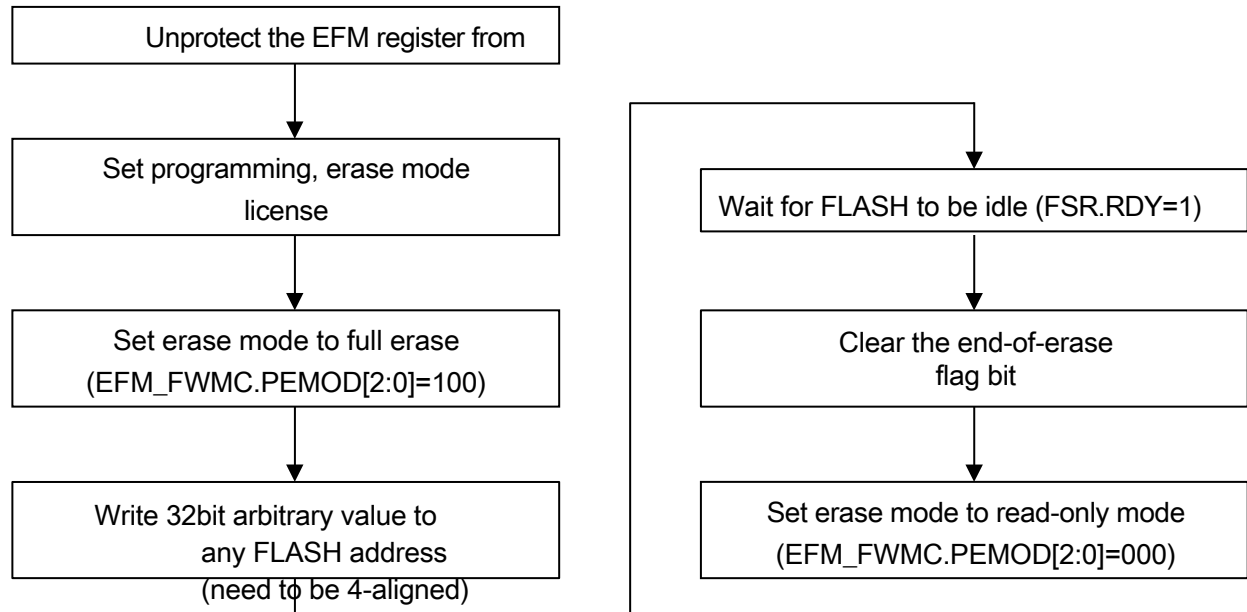
Caution:

- EFM_FWMC.PEMODE is a permission for the EFM_FWMC.PEMOD[2:0] setting;
- PEMOD[2:0] should be set to read-only mode after the erase is completed to prevent misuse of FLASH, which may cause the whole sector to be erased.

3.2.2.3 Full Erase

EFM provides two types of erasure methods, sector erase and full erase. After full erase operation of FLASH, the entire FLASH area is refreshed to full 1 with address data.

The setting steps for full erase are as follows:

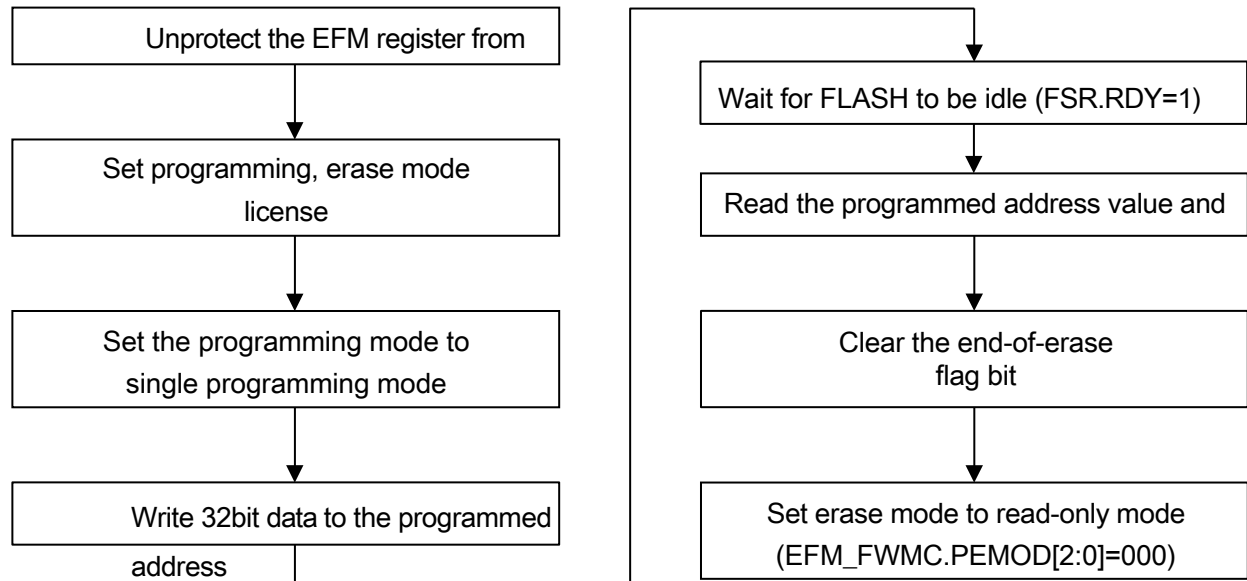


Caution:

- PEMOD[2:0] should be set to read-only mode after the erase is completed to prevent misuse of FLASH, which may cause the whole sector to be erased.

3.2.2.4 Single programming without readback

The steps for setting the single-programmed no-readback mode are as follows:

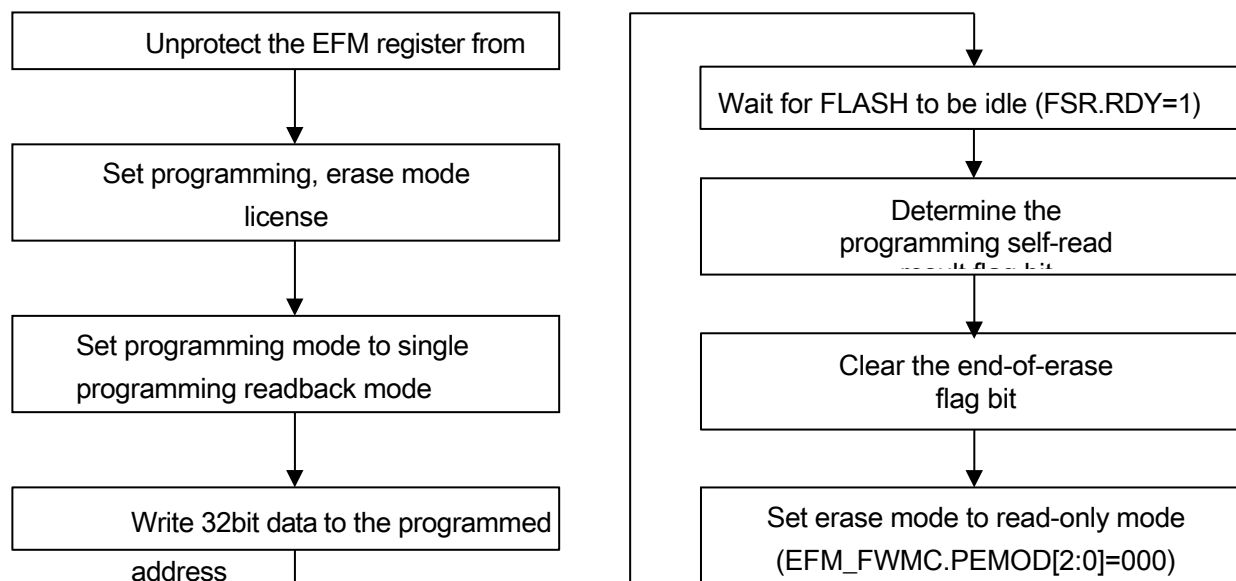


3.2.2.5 Single programming readback

Single programmed readback mode means that the programmed address is automatically read and compared with the written data after the programming is finished, and the consistency flag is output.

EFM_FSR.PGMISMTCH.

The single programming readback mode is set as follows:



Caution:

- EFM_FSR.PGMISMTCH is 0, indicating successful programming, and 1, indicating that the FLASH address has been corrupted and is permanently discarded.

3.2.2.6 Continuous Programming

Continuous programming mode is recommended when programming FLASH addresses continuously. Continuous programming mode can save more than 50% of time compared to single programming mode. The frequency should not be lower than 12MHz for continuous programming mode.

A comparison of continuous programming mode and single programming mode events is shown in the following table:

Sym bols	Par ame ters	Con diti ons	Minimum value	Typical values	Maximum value	Unit
-------------	--------------------	--------------------	------------------	-------------------	------------------	------

T _{prog}	Word programming time	Single Programming Mode	43+2* Thclk	48+4* Thclk	53+6* Thclk	μs
	Word programming time	Continuous programming mode	12+2* Thclk	14+4* Thclk	16+6* Thclk	μs
T _{erase}	Block Erase Time	-	16+2* Thclk	18+4* Thclk	20+6* Thclk	ms
T _{mas}	Full Erase Time	-	16+2* Thclk	18+4* Thclk	20+6* Thclk	ms

Note: Thclk is 1 cycle of the CPU clock

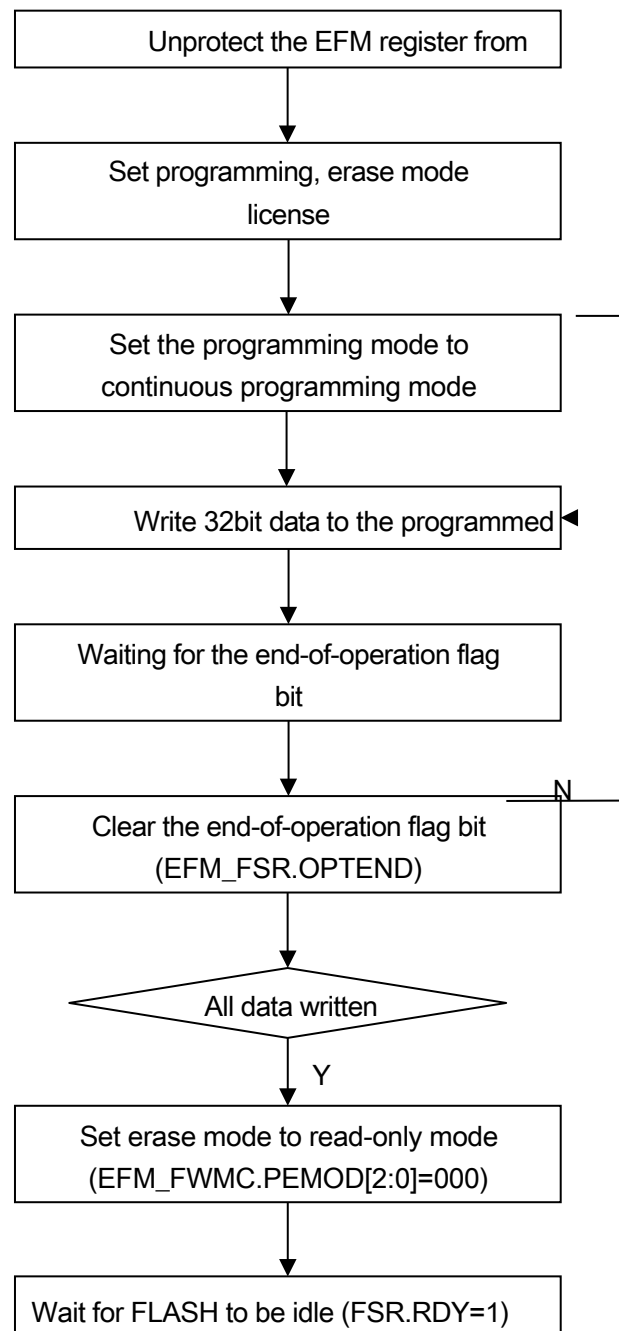
The continuous

programming setting

procedure is as follows:

Note

- During FLASH continuous programming, if FLASH is read, it will read an indefinite value.
- Before initiating a dummy read to any FLASH address in sequential programming mode, the cache enable function should be disabled to prevent the erroneous data from entering the cache pool; at the same time, the dummy read will generate a read conflict bit (i.e., EFM_FSR.RDCOLERR bit).



3.2.2.7 Bus hold/release function

BUSHLDCTL bit in the EFM_FWMC register allows you to set whether the bus is held or released during FLASH programming and erasing.

This control bit must be set to 0 when FLASH programming and erase instructions are executed on FLASH, and can be freely set as required when the erase instruction is executed in an unintended space of FLASH (e.g. RAM).

BUSHLDCTL is 1 (i.e., during FLASH programming and erase, bus release state), when programming (except for continuous programming), read/write access to FLASH will be protected before the end of erase (EFM_FSR.RDY=1), flag bit EFM_FSR.BUSCOLERR position bit.

3.2.3 One Time Programmable Byte (OTP)

The OTP (One Time Program) area is divided into 15 64-byte blocks of data, each corresponding to a 4Bytes latch address. See Table 9-3 in the user manual for details.

The latch address is used to latch the corresponding data block. When the latch address data is all 1, the corresponding OTP area data block is programmable; when the latch address data is all 0, the corresponding OTP area data is not programmable.

All OTP data blocks and latch addresses are not erasable.

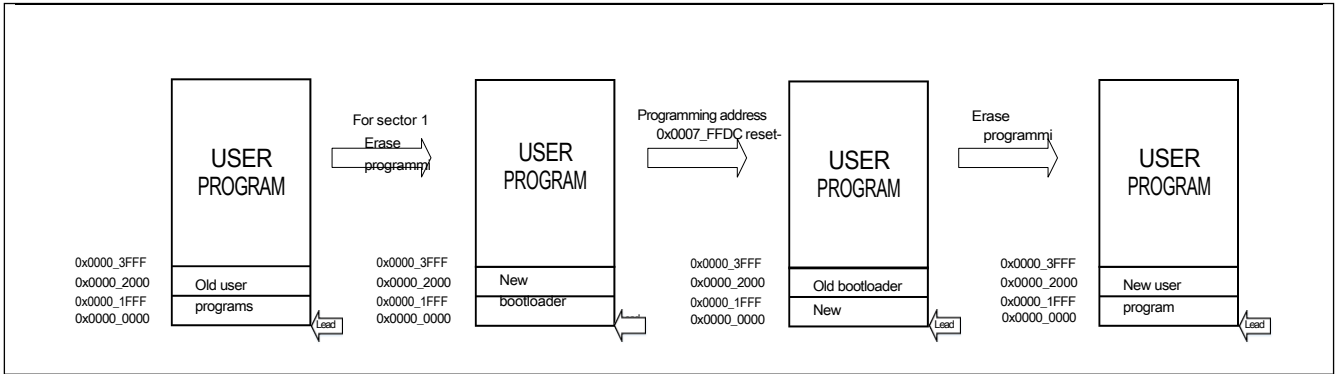
The programming of the OTP area is encapsulated in the flashloader, so the user can directly manipulate the otp address. For a sample, please refer to Sample code of efm_otp under EFM module.

3.2.4 Guided exchange

EFM provides the boot-swap function. When users upgrade the boot program, they erase sector 0 (0x00000000~0x00001FFF), and if an unscheduled accident (power down, reset) is encountered during the erase, it may cause the whole chip to fail to boot normally. The boot swap function can avoid this situation.

Before erasing sector 0, write the new boot program to sector 1 (0x00002000~0x00003FFF), then program the data 0xFFFF4321 to EFM address 0x0007FFDC and reset the terminal to enable the CPU to start the new boot program from sector 1, then erase sector 0 and reprogram the user program.

The flow of the bootstrap exchange is as follows:



When upgrading the bootloader again, the address 0x0007FFDC, where the boot sector swap information is stored, has already been programmed

(The user can determine if the boot swap function has been used by reading the FLASH address or the EFM_FSWP register.

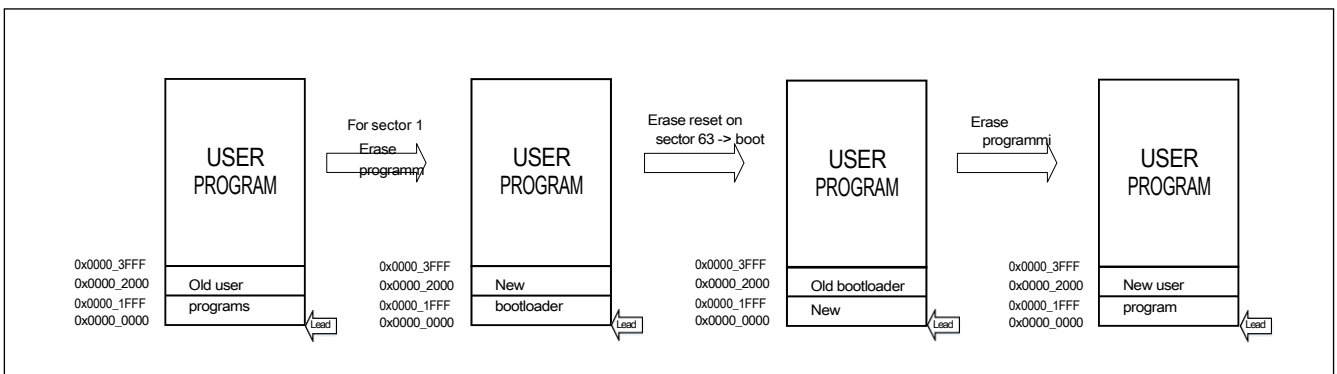
EFM_FSWP.FSWP = 0, indicating that sector 0 and sector 1 have been swapped and will be started from sector 1 after the reset), sector 63 needs to be reset.

(0x0007E000~0x0007FFFF) to perform sector erasure before upgrading the bootloader.

Before erasing sector 0, the new bootloader is pre-written to sector 1, and then sector 63 is erased by end

The CPU starts a new boot program from sector 1, and then erases sector 0 to reprogram a new boot program.

The operation process is as follows:



The sample bootstrap switch can be found in the efm_switch sample code under the EFM module.

4 Sample Code

4.1 Code Introduction

Users can write their own code to learn to verify the module according to the above workflow, or download the sample code of Device Driver Library (DDL) directly from the website of UW Semiconductors and use the sample of FLASH to verify.

The following section briefly describes the configuration involved in the efm_simple code of this sample DDL-based FLASH module of AN.

- 1) To unprotect the FLASH registers:

```
/* Unlock EFM. */  
EFM_Unlock().
```

- 2) Enabling FLASH:

```
/* Enable flash. */  
EFM_FlashCmd(Enable).
```

- 3) Waiting for FLASH ready:

```
/* Wait flash ready. */  
while(Set != EFM_GetFlagStatus(EFM_FLAG_RDY)).
```

- 4) Sector erasure:

```
/* Erase sector 62. */  
EFM_SectorErase(FLASH_SECTOR62_ADRR)
```

- 5) Programming FLASH:

```
u32Addr = FLASH_SECTOR62_ADR R.  
for(i = 0; i < 10; i++)  
{  
    EFM_SingleProgram(u32Addr,u32TestData);  
    u32Addr += 4.  
}
```

- 6) Lock FLASH registers:

```
/* Lock EFM.  
*/ EFM_Lock().
```

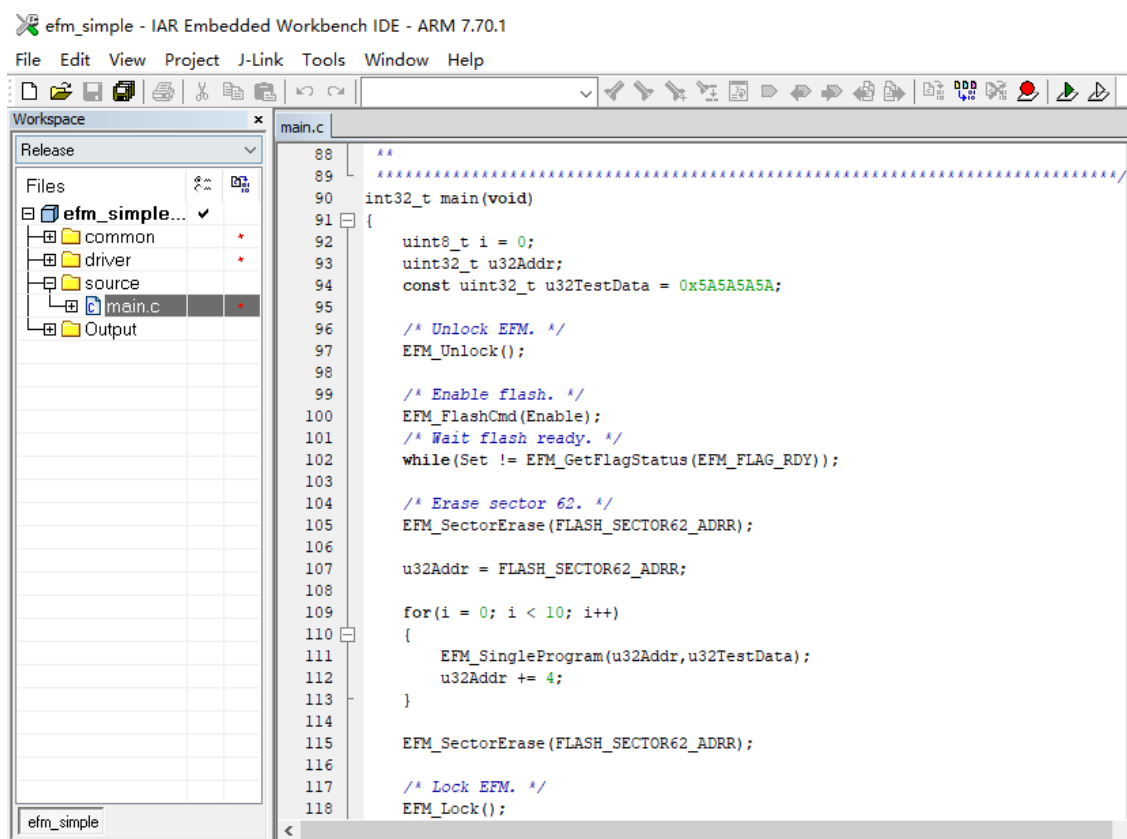

4.2 Code Run

Users can download the sample code (efm_simple) of the HC32F460 DDL from the UW website and run the code with the evaluation board (EV-HC32F460-LQFP100-050-V1.1) to learn how to use the FLASH module.


The following section describes how to run the FLASH sample code on the evaluation board and observe the results:

- Verify that the correct IAR EWARM v7.7 tool is installed (please download the appropriate installation package from the official IAR website and refer to the user manual for installation).
- Download the HC32F460 DDL code from the UW Semiconductors website.
- Download and run the project file in efm\ efm_simple\ at

1) Open the efm_simple \ project and open the 'main.c' view as follows:



2) Click  to recompile the entire project.

3) Click  Download the code to the evaluation board, break the point at line 105 and 115 code lines respectively, and run at full speed.

- 4) Run the first breakpoint, single-step, check memory, and observe that sector 62 is all 1.
Continue to run at full speed to the next breakpoint, where the address is written to the expected value. The base single step is run and the written value is erased.

5 Version Information & Contact

Date	Versions	Modify records
2019/3/15	Rev1.0	Initial Release
2020/8/26	Rev1.1	Update supported models



If you have any comments or suggestions in the process of purchase and use, please feel free to contact us.

Email: mcu@hdsc.com.cn

Website: <http://www.hdsc.com.cn/mcu.htm>

Address: 10/F, Block A, 1867 Zhongke Road, Pudong

New Area, Shanghai, 201203, P.R. China

