Freescale Semiconductor
Application Note

Document Number: AN5074
Rev 0, 01/2015

EEPROM Driver for MC56F84xxx and MC56F82xxx DSC Family

by: Xuwei Zhou

Contents

1 Introduction

In the Freescale MC56F84xxx DSC family, part or all of FlexNVM together with a RAM block of 1K words called FlexRAM can be used to emulate the characteristics of an EEPROM using a built-in filing system. Once the EEPROM is properly configured, users can manipulate FlexRAM to write to or read from this EEPROM. The filing system does all the record backup work, which users can be blind to.

Refer to MC56F847xx Reference Manual or AN4689: Using EEPROM on MC56F84xxx DSC for more information.

In MC56F82xxx DSC family, FlexNVM or FlexRAM is not available. If EEPROM is desired, you have to emulate it on program Flash by firmware.

This application note describes an EEPROM driver for both MC56F84xxx and MC56F82xxx DSC family. You can use this driver directly through the guidelines in this application note. The application note also describes a method of reprogramming Flash without erasing EEPROM in CodeWarrior10.6.

As for MC56F84xxx DSC family, this driver is suitable for both Small Data Mode and Large Data Mode since it is written in assembly. AN4689 also provides a driver for EEPROM in MC56F84xxx DSC family, but it only applies to Large Data Mode. And as for MC56F82xxx DSC family, this driver uses Flash Driver Library described in AN4860: Flash Driver

Library for MC56F847xx and MC56F827xx DSC Family together with CRC feature to emulate EEPROM for higher reliability.

2 EEPROM driver description

This driver is developed in order to make the EEPROM in DSC easier and efficient to use. For MC56F84xxx family, there are APIs with byte string, word string, and longword string write&read functions and APIs with byte, word, and longword write&read functions. For MC56F82xxx family, the driver is developed using incremental writing feature in the Erase Sector mode. CRC is performed every time an entry is written into or read from the Flash in order to improve reliability.

2.1 Description of EEPROM driver for MC56F84xxx family

Set EEPROM_EMULATION to 0 in *EepromDrv_cfg.h* file to enable drivers for MC56F84xxx family. All functions are written in assembly to fit both small and large data model as well as to increase execution efficiency. Table 1 lists all the user-available functions for MC56F84xxx family. This driver is realized in *EepromDrv.c* and *EepromDrv.h*. The driver used in CodeWarrior is shown in Figure 1.

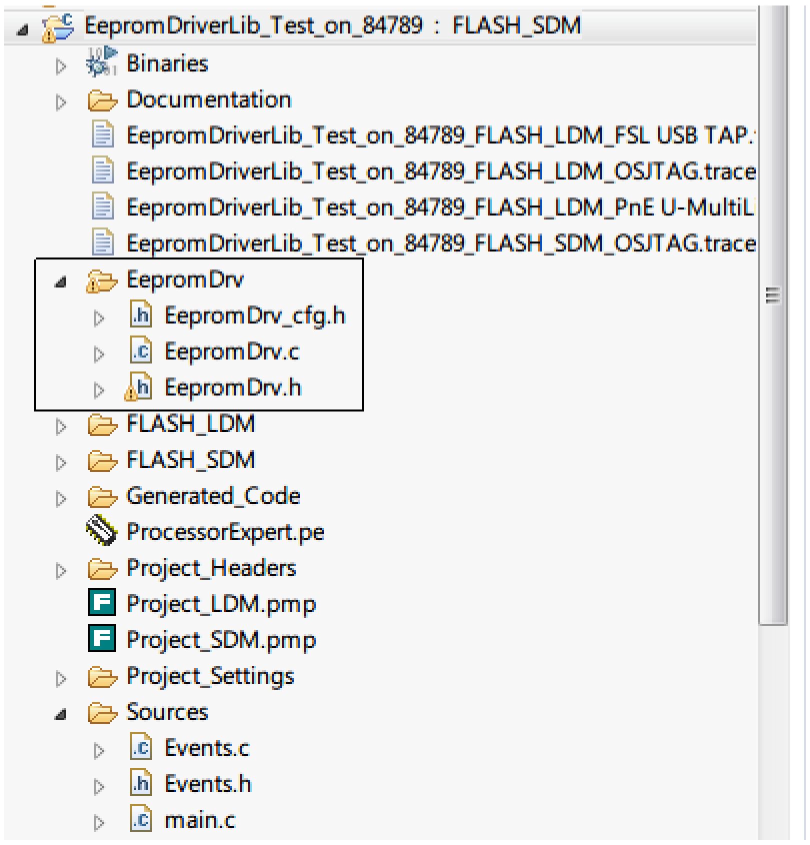

EepromDriverLib_Test_on_84789 : FLASH_SDM
Binaries
Documentation
EepromDriverLib_Test_on_84789_FLASH_LDM_FSL USB TAP.
EepromDriverLib_Test_on_84789_FLASH_LDM_OSJTAG.trace
EepromDriverLib_Test_on_84789_FLASH_LDM_PnE U-MultiL
EepromDriverLib_Test_on_84789_FLASH_SDM_OSJTAG.trace
EepromDrv
EepromDrv_cfg.h
EepromDrv.c
EepromDrv.h
FLASH_LDM
FLASH_SDM
Generated_Code
ProcessorExpert.pe
Project_Headers
Project_LDM.pmp
Project_SDM.pmp
Project_Settings
Sources
Events.c
Events.h
main.c

Figure 1. CodeWarrior projects view showing usage of EEPROM driver for MC56F84xxx family

Table 1. List of EEPROM drivers for MC56F84xxx family

Function Name	Short Description
GetEepromInfo()	Get the size of EEPROM and backup FlexNVM
DEFlashPartition()	Set the size of EEPROM and backup FlexNVM
SetEEEEnable()	Enable FlexRAM for EEPROM interface
SetEEEDisable()	Set FlexRAM as regular RAM, no EEPROM function
EepromWriteByte()	Write a byte into an address in EEPROM
EepromReadByte()	Read a byte from an address in EEPROM
EepromWriteWord()	Write a word into an address in EEPROM
EepromReadWord()	Read a word from an address in EEPROM
EepromWriteLongWord()	Write a longword into an address in EEPROM
EepromReadLongWord()	Read a longword from an address in EEPROM
EepromWriteByteString()	Write a string of bytes into EEPROM
EepromReadByteString()	Read a string of bytes from EEPROM
EepromWriteWordString()	Write a string of words into EEPROM
EepromReadWordString()	Read a string of words from EEPROM
EepromWriteLongWordString()	Write a string of longwords into EEPROM
EepromReadLongWordString()	Read a string of longwords from EEPROM

All functions use the following data types defined in *EepromDrv.h*:

- UWord8 – unsigned byte. Range: [0, 255]
- UWord16 – unsigned word (two bytes). Range: $[0, 2^{16})$
- UWord32 – unsigned longword (four bytes). Range: $[0, 2^{32})$

The return codes are also defined in *EepromDrv.h*. Only the first four functions in Table 1 have return codes. See Table 2 for the list.

Table 2. Return codes of EEPROM driver for MC56F84xxx family

Return Code	Defined Value
EEPROM_FLASHDRV_SUCCESS	0
EEPROM_FLASHDRV_FAIL	1
EEPROM_FLASHDRV_ACCESS_ERROR	2
EEPROM_FLASHDRV_PROT_VIOLATION	3

You can partition FlexNVM in two parts: EEPROM backup and Data Flash. Both Program Flash and Data Flash have a small non-volatile information registers called IFR, which are separate from the main memory array. The IFR of Data Flash has 256 bytes, two of which contain EEPROM related information:

- EEESIZE: The least significant four bits in this byte determines the amount of FlexRAM used in each of the available EEPROM subsystems. The available values are defined in *EepromDrv.h* as listed in Table 3.

Table 3. Available values for EEESIZE

Code	Defined Value	EEPROM Size	Access Word Address of FlexRAM in Data Memory Map
EEESIZE_2048B	0x33	2048 bytes	0x1e000 ~ 0x1e3ff
EEESIZE_1024B	0x34	1024 bytes	0x1e000 ~ 0x1e1ff
EEESIZE_512B	0x35	512 bytes	0x1e000 ~ 0x1e0ff
EEESIZE_256B	0x36	256 bytes	0x1e000 ~ 0x1e07f
EEESIZE_128B	0x37	128 bytes	0x1e000 ~ 0x1e03f
EEESIZE_64B	0x38	64 bytes	0x1e000 ~ 0x1e01f
EEESIZE_32B	0x39	32 bytes	0x1e000 ~ 0x1e00f
EEESIZE_0B	0x3F	0	N/A

- DEPART: The least significant four bits in this byte specifies the amount of FlexNVM that is used as EEPROM backup memory. The available values are defined in *EepromDrv.h* as listed in Table 4.

Table 4. Available values for DEPART

Code	Defined Value	Backup Size for EEPROM	Size of Data Flash
DEPART_0K	0x0	No EEPROM backup	32K bytes
DEPART_8K	0x1	8K bytes	24K bytes
DEPART_16K	0x2	16K bytes	16K bytes
DEPART_24K	0x9	24K bytes	8K bytes
DEPART_32K	0x3	32K bytes	No remainder

During the reset sequence, values of EEESIZE and DEPART determine whether FlexNVM is partitioned for EEPROM backup. If so, EEPROM backup data is copied to the configured FlexRAM and the EEERDY flag in FTFL_FCNFG register is set. Otherwise, FlexRAM serves as regular RAM and the RAMRDY flag in FTFL_FCNFG register is set.

EEESIZE and DEPART bytes in Data Flash IFR can be modified by Program Partition command in FTFL module only if Data Flash IFR is already in an erased state, where the value of EEESIZE and DEPART is 0xFF. An Erase All Blocks command or external request of triggering the Erase All command can erase IFR of both Data Flash and Program Flash. See MC56F847xx Reference Manual for more information on FTFL commands.

There are two global variables *uw16EEESize* and *uw16EEBackUpFlashSize* in the driver, which are used to store the value of EEESIZE and DEPART.

Refer to Table 24: Flash command timing specifications in MC56F847xx Advance Information Data Sheet for the performance of all the functions listed in this section.

2.1.1 GetEepromInfo()

This function reads the IFR of Data Flash to get the value of EEESIZE and DEPART. Read Resource command is executed to realize the reading and store the values to variables *uw16EEESize* and *uw16EEBackUpFlashSize*. This function should be invoked once before EEPROM is used. The values of *uw16EEESize* and *uw16EEBackUpFlashSize* tell whether EEPROM is configured:

- *uw16EEESize == 0xFF and uw16EEBackUpFlashSize == 0xFF*: No backup FlexNVM, EEPROM is not configured.
- *uw16EEESize != 0xFF or uw16EEBackUpFlashSize != 0xFF*: EEPROM has been configured. The least significant four bits of this two variables reflect the size of EEPROM and backup FlexNVM.

Prototype of this function is:

```
UWord32 GetEepromInfo(void);
```

Table 5. GetEepromInfo() function return codes

Return Code	Description
EEPROM_FLASHDRV_SUCCESS	Successfully get the EEPROM information
EEPROM_FLASHDRV_ACCESS_ERROR	Function internal error

Listing 1 on page 6 shows how to use *GetEepromInfo()* for EEPROM initialization.

2.1.2 DEFlashPartition()

This function configures the size of EEPROM and backup FlexNVM by programming EEESIZE and DEPART if the EEPROM has not been configured during initialization. The Program Partition command is executed, which prepares the FlexNVM block for use as Data Flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Once the Program Partition command is launched, EEESIZE and DEPART in Data Flash IFR are checked to see if they have been erased. If erased, this command erases the contents of the FlexNVM memory, and the FlexNVM is partitioned for EEPROM backup accordingly. The allocated EEPROM backup sectors are formatted for EEPROM use. Finally, the partition codes in Table 6 are programmed into the Data Flash IFR. This command also verifies that the partition codes read back correctly after programming. EEERDY flag in FTFL_FCNFG will set if FlexNVM is partitioned successfully for EEPROM backup.

Prototype of this function is:

```
UWord32 DEFlashPartition(UWord8 EEEDataSize, UWord8 EEBackUpFlashSize);
```

The function parameters and return codes are listed in following tables.

Table 6. DEFlashPartition() function parameters

Parameter Name	Parameter Type	Description
EEEDataSize	UWord8	Configure the size of EEPROM. Use the code in Table 3.
EEBackUpFlashSize	UWord8	Configure the backup Data Flash size. Use the code in Table 4.

Table 7. DEFlashPartition() function return codes

Return Code	Description
EEPROM_FLASHDRV_SUCCESS	Successfully configure EEPROM
EEPROM_FLASHDRV_ACCESS_ERROR	Function internal error
EEPROM_FLASHDRV_FAIL	MGSTAT0 bit of FTFL_FSTAT is set, meaning any errors have been encountered during the verify operation.

Avoid the following operations or else EEPROM_FLASHDRV_ACCESS_ERROR will occur:

- Invoke *DEFlashPartition()* when EEPROM has already been configured, namely, either EEESIZE != 0xFF or DEPART != 0xFF.
- Pass the code DEPART_0K to EEBackUpFlashSize, and pass a code to EEEDataSize that allocates FlexRAM for EEPROM.
- Pass the code EEESIZE_0B to EEEDataSize, and pass a code to EEBackUpFlashSize that allocates space for EEPROM backup.

Listing 1. Use of GetEepromInfo() and DEFlashPartition() functions for EEPROM initialization

```
#include "EepromDrv.h"
Word16   w16Stat;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();

  w16Stat = GetEepromInfo();

  if(((uw16EEESize&0x00ff) == 0xff) && ((uw16EEBackUpFlashSize&0x00ff) == 0xff)
      && (w16Stat == EEPROM_FLASHDRV_SUCCESS))
  {
     // Data Flash will be erased during partition
     // 256bytes of EEPROM, with 16K bytes FlexNVM as backup
     w16Stat = DEFlashPartition(EEESIZE_256B,DEPART_16K);
  }
}
```

2.1.3 SetEEEEnable()

This function enables FlexRAM as the interface to EEPROM. The Set FlexRAM Function command is executed and makes the FlexRAM available for EEPROM. The existing EEPROM data from the EEPROM backup record space is copied to the FlexRAM by flash module when the command completes, and EEERDY flag in FTFL_FCNFG is set, RAMRDY flag is cleared. In this scenario, normal read and write access to the FlexRAM is available, but writes to the FlexRAM also invoke EEPROM activity. Use EEPROM write and read functions provided in this application note to operate EEPROM.

Prototype of this function is:

```
UWord32 SetEEEEnable(void);
```

Table 8. SetEEEEnable() function return codes

Return Code	Description
EEPROM_FLASHDRV_SUCCESS	Successfully enable FlexRAM as interface to EEPROM
EEPROM_FLASHDRV_ACCESS_ERROR	Function internal error

NOTE

When *DEFlashPartition()* is successfully executed and FlexRAM is already configured as interface to EEPROM, it is unnecessary to invoke *SetEEEEnable()* right after *DEFlashPartition()*.

2.1.4 SetEEEDisable()

This function sets FlexRAM as traditional RAM, but not the interface to EEPROM. The Set FlexRAM Function command is executed with making the FlexRAM available as traditional RAM. The entire FlexRAM is written with ones by flash module when the command completes, and RAMRDY flag in FTFL_FCNFG is set, EEERDY flag is cleared. In this scenario, normal read and write access to the FlexRAM is available.

Prototype of this function is:

```
UWord32 SetEEEDisable(void);
```

Table 9. SetEEEDisable() function return codes

Return Code	Description
EEPROM_FLASHDRV_SUCCESS	Successfully set FlexRAM as traditional RAM
EEPROM_FLASHDRV_ACCESS_ERROR	Function internal error

NOTE

When FlexRAM is configured as traditional RAM, use LDM to access it because the start word address of FlexRAM is 0x1E000, which is beyond 16 bits. Or use the inline functions in *EepromDrv.h* to access FlexRAM, which are suitable for both SDM and LDM.

Listing 2 on page 7 shows how to use *SetEEEEnable()* and *SetEEEDisable()* to change the role of FlexRAM

Listing 2. Use of SetEEEEnable() and SetEEEDisable() to change the role of FlexRAM.

```
#include "EepromDrv.h"
UWord8   uw8Data;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*
  /* EEPROM initialization */
  // FlexNVM partition...

  EepromWriteByte(EEPROM_BASE_ADDR_BYTE, 0x12);//Write data 0x12 to the first byte cell of
                                               //EEPROM
  SetEEEDisable(); // Set FlexRAM as traditional RAM

  /*
      Inline functions below can be used:
      UWord32 FlexRAM_ReadLongword(register UWord32 dwAddress);
      void FlexRAM_WriteLongword(register UWord32 dwAddress, register UWord32 dwData);
      UWord16 FlexRAM_ReadWord(register UWord32 dwAddress);
      void FlexRAM_WriteWord(register UWord32 dwAddress, register UWord16 dwData);
      UWord8 FlexRAM_ReadByte(register UWord32 dwAddress);
      void FlexRAM_WriteByte(register UWord32 dwAddress, register UWord8 dwData);
  */
  // Use FlexRAM for other operations...

  SetEEEEnable(); // Set FlexRAM as interface to EEPROM.
  EepromReadByte(EEPROM_BASE_ADDR_BYTE,&uw8Data ); //Read the first byte cell of EEPROM and
                                                    //store the data to uw8Data,the value is
                                                    //still 0x12

}
```

2.1.5 EepromWriteByte()

When EEPROM has been properly configured, use this function to write a byte (8-bit) to the desired address in EEPROM.

Prototype of this function is:

```
void EepromWriteByte(UWord32 byteAddr,UWord8 data);
```

Table 10. EepromWriteByte() function parameters

Parameter Name	Parameter Type	Description
byteAddr	UWord32	The address of EEPROM
data	UWord8	The byte data that tends to be written into EEPROM

For instance, if EEPROM size is configured to be 64 bytes using constant *EEESIZE_64B*, the available EEPROM byte address range is 0x3C000~0x3C03F. There's a macro definition in *Eeprom.h*:

```
#define EEPROM_BASE_ADDR_BYTE 0x3c000
```

You can use EEPROM_BASE_ADDR_BYTE as the base address when EEPROM is accessed in bytes.

2.1.6 EepromReadByte()

This function is to read a byte from a specified byte address in EEPROM.

Prototype of this function is:

```
void EepromReadByte(UWord32 byteAddr,UWord8 *data);
```

Table 11. EepromReadByte() function parameters

Parameter Name	Parameter Type	Description
byteAddr	UWord32	The address of EEPROM
data	UWord8*	A byte pointer. The read out byte is stored to the place where this pointer points

Listing 2 on page 7 also shows how to use *EepromWriteByte()* and *EepromReadByte()* to write a byte to and read a byte from EEPROM.

2.1.7 EepromWriteByteString()

This function writes a string of bytes data to EEPROM.

Prototype of this function is:

```
void EepromWriteByteString(UWord32 byteAddr,UWord8* data, UWord16 length);
```

Table 12. EepromWriteByteString() function parameters

Parameter Name	Parameter Type	Description
byteAddr	UWord32	The start address of EEPROM that data string is written to

Table continues on the next page...

Table 12. EepromWriteByteString() function parameters (continued)

Parameter Name	Parameter Type	Description
data	UWord8*	The byte pointer that points to a byte data string which is to be written into EEPROM
length	UWord16	Length of the string in unit of bytes

2.1.8 EepromReadByteString()

This function reads a string of bytes data out of EEPROM from a specified start byte address.

Prototype of this function is:

```
void EepromReadByteString(UWord32 byteAddr,UWord8* data, UWord16 length);
```

Table 13. EepromReadByteString() function parameters

Parameter Name	Parameter Type	Description
byteAddr	UWord32	The start address of EEPROM that data string is read out of
data	UWord8*	The byte pointer that points to a byte data string, to which the read out byte string is stored
length	UWord16	Length of the string in unit of bytes

Listing 3 on page 9 shows how to use *EepromWriteByteString()* and *EepromReadByteString()* to access EEPROM.

Listing 3. Use of EepromWriteByteString() and EepromReadByteString() to access EEPROM

```
#include "EepromDrv.h"
UWord8   uw8Num[32];
UWord8   uw8NumRd[32];
Word16   w16Stat;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  Word8 i;

  /* EEPROM initialization, 32 bytes of EEPROM with 16K bytes of FlexNVM backup */
  w16Stat = GetEepromInfo();

  if(((uw16EEESize&0x00ff) == 0xff) && ((uw16EEBackUpFlashSize&0x00ff) == 0xff)
      && (w16Stat == EEPROM_FLASHDRV_SUCCESS))
  {
     // Data Flash will be erased during partition
     // 32 bytes of EEPROM, with 16K bytes FlexNVM as backup
     w16Stat = DEFlashPartition(EEESIZE_32B,DEPART_16K);
  }

  for(i=0;i<32;i++)
  {
     uw8Num[i] += i;
  }
  /* 32 bytes in uw8Num[0]~uw8Num[31] are written into EEPROM sequentially */
  EepromWriteByteString(EEPROM_BASE_ADDR_BYTE,uw8Num,32);
  /* The data residing in EEPROM_BASE_ADDR_BYTE to (EEPROM_BASE_ADDR_BYTE+31) of EEPROM
     Are read out and stored in uw8NumRd[0]~uw8NumRd[31] */
  EepromReadByteString(EEPROM_BASE_ADDR_BYTE,uw8NumRd,32);

}
```

2.1.9 EepromWriteWord()

In contrast to *EepromWriteByte()* function, a 16-bit word data is written into EEPROM. The difference lies in the address of EEPROM. For instance, if EEPROM size is configured to be 64 bytes using constant *EEESIZE_64B*, the available EEPROM word address range is 0x1E000~0x1E01F.

There is a macro definition in *Eeprom.h*:

```
#define EEPROM_BASE_ADDR_WORD 0x1e000
```

You can use EEPROM_BASE_ADDR_WORD as the base address when EEPROM is accessed in words.

Prototype of this function is:

```
void EepromWriteWord(UWord32 wordAddr,UWord16 data);
```

Table 14. EepromWriteByte() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The address of EEPROM
data	UWord16	The word data that tends to be written into EEPROM

NOTE

Byte address EEPROM_BASE_ADDR_BYTE and (EEPROM_BASE_ADDR_BYTE +1) actually refer to the least significant byte and most significant byte of word address EEPROM_BASE_ADDR_WORD. The rest can be done in the same manner to understand the relationship between byte address and word address.

2.1.10 EepromReadWord()

This function is to read a word from a specified word address in EEPROM.

Prototype of this function is:

```
void EepromReadWord(UWord32 wordAddr,UWord16 *data);
```

Table 15. EepromReadByte() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The address of EEPROM
data	UWord16*	A word pointer. The read out word is stored to the place where this pointer points

Listing 4 on page 11 shows how to use *EepromWriteWord()* and *EepromReadWord()* to write a word to and read a word from EEPROM.

Listing 4. Use of EepromWriteWord() and EepromReadWord() to access EEPROM

```
#include "EepromDrv.h"
UWord16   uw16Num;
UWord16   uw16NumRd;
Word16    w16Stat;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();

  /* EEPROM initialization, 32 bytes of EEPROM with 16K bytes of FlexNVM backup */
  w16Stat = GetEepromInfo();

  if(((uw16EEESize&0x00ff) == 0xff) && ((uw16EEBackUpFlashSize&0x00ff) == 0xff)
      && (w16Stat == EEPROM_FLASHDRV_SUCCESS))
  {
     // Data Flash will be erased during partition
     // 32 bytes of EEPROM, with 16K bytes FlexNVM as backup
     w16Stat = DEFlashPartition(EEESIZE_32B,DEPART_16K);
  }

  uw16Num = 0x4567;

  EepromWriteWord(EEPROM_BASE_ADDR_WORD+2,uw16Num);// write 0x4567 to address of 0x1e002
  EepromReadWord(EEPROM_BASE_ADDR_WORD+2,&uw16NumRd);// read the word data in 0x1e002 out to
                                                     // variable uw16NumRd

}
```

2.1.11 EepromWriteWordString()

This function writes a string of words data to EEPROM.

Prototype of this function is:

```
void EepromWriteWordString(UWord32 wordAddr,UWord16* data, UWord16 length);
```

Table 16. EepromWriteWordString() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The start word address of EEPROM that data string is written to
data	UWord16*	The word pointer that points to a word data string which is to be written into EEPROM
length	UWord16	Length of the string in unit of words

2.1.12 EepromReadWordString()

This function reads a string of words data out of EEPROM from a specified start word address.

Prototype of this function is:

```
void EepromReadWordString(UWord32 wordAddr,UWord16* data, UWord16 length);
```

Table 17. EepromReadWordString() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The start word address of EEPROM that data string is read out of
data	UWord16*	The word pointer that points to a word data string, to which the read out word string is stored
length	UWord16	Length of the string in unit of words

Listing 5 on page 12 shows how to use *EepromWriteWordString()* and *EepromReadWordString()* to access EEPROM.

Listing 5. Use of EepromWriteWordString() and EepromReadWordString() to access EEPROM

```
#include "EepromDrv.h"
UWord16   uw16Num[32];
UWord16   uw16NumRd[32];
Word16    w16Stat;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  Word8 i;

  /* EEPROM initialization, 64 bytes of EEPROM with 16K bytes of FlexNVM backup */
  w16Stat = GetEepromInfo();

  if(((uw16EEESize&0x00ff) == 0xff) && ((uw16EEBackUpFlashSize&0x00ff) == 0xff)
      && (w16Stat == EEPROM_FLASHDRV_SUCCESS))
  {
      // Data Flash will be erased during partition
      // 64 bytes of EEPROM, with 16K bytes FlexNVM as backup
     w16Stat = DEFlashPartition(EEESIZE_64B,DEPART_16K);
  }

  for(i=0;i<32;i++)
  {
  uw16Num[i] += i;
  }
  /* 32 words in uw16Num[0]~uw16Num[31] are written into EEPROM sequentially */
  EepromWriteWordString(EEPROM_BASE_ADDR_WORD,uw16Num,32);
  /* The data residing in EEPROM_BASE_ADDR_WORD to (EEPROM_BASE_ADDR_WORD+31) of EEPROM
     Are read out and stored in uw16NumRd[0]~uw16NumRd[31] */
  EepromReadWordString(EEPROM_BASE_ADDR_WORD,uw16NumRd,32);

}
```

2.1.13 EepromWriteLongWord()

In contrast to *EepromWriteByte()* function, a 32-bit long word data is written into EEPROM. The difference lies in the address of EEPROM. For instance, if EEPROM size is configured to be 64 bytes using constant *EEESIZE_64B*, which means the space can hold up to 16 long words. The available EEPROM long word addresses in sequence is:

```
0x1E000, 0x1E002, 0x1E004, 0x1E006, 0x1E008, 0x1E00A, 0x1E00C, 0x1E00E

0x1E010, 0x1E012, 0x1E014, 0x1E016, 0x1E018, 0x1E01A, 0x1E01C, 0x1E01E
```

Notice, the address for long word access in EEPROM should be an even number.

Prototype of this function is:

```
void EepromWriteLongWord(UWord32 wordAddr,UWord32 data);
```

Table 18. EepromWriteLongWord() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The long word address of EEPROM. It must be even.
data	UWord32	The long word data that tends to be written into EEPROM

2.1.14 EepromReadLongWord()

This function is to read a long word from a specified long word address in EEPROM.

Prototype of this function is:

```
EepromReadLongWord(UWord32 wordAddr,UWord32 *data);
```

Table 19. EepromReadByte() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The long word address of EEPROM. It must be even.
data	UWord32*	A long word pointer. The read out long word is stored to the place where this pointer points

Listing 6 on page 13 shows how to use *EepromWriteLongWord()* and *EepromReadLongWord()* to write a long word to and read a long word from EEPROM.

Listing 6. Use of EepromWriteLongWord() and EepromReadLongWord() to access EEPROM

```
#include "EepromDrv.h"
UWord32   uw32Num,uw32Num1;
UWord32   uw32NumRd,uw32NumRd1;
Word16    w16Stat;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();

  /* EEPROM initialization, 32 bytes of EEPROM with 16K bytes of FlexNVM backup */
  w16Stat = GetEepromInfo();

  if(((uw16EEESize&0x00ff) == 0xff) && ((uw16EEBackUpFlashSize&0x00ff) == 0xff)
      && (w16Stat == EEPROM_FLASHDRV_SUCCESS))
  {
     // Data Flash will be erased during partition
     // 32 bytes of EEPROM, with 16K bytes FlexNVM as backup
     w16Stat = DEFlashPartition(EEESIZE_32B,DEPART_16K);
  }

  uw32Num = 0x11223344;
  uw32Num1 = 0x55667788;

  EepromWriteLongWord(EEPROM_BASE_ADDR_WORD,uw32Num);// write 0x11223344 to address of
                                                   // 0x1e000
  EepromWriteLongWord(EEPROM_BASE_ADDR_WORD+2,uw32Num1);// write 0x55667788 to address of
                                                        // 0x1e002

  EepromReadLongWord(EEPROM_BASE_ADDR_WORD,&uw32NumRd);// read the word data in 0x1e000 out
                                                       // to variable uw32NumRd

  EepromReadLongWord(EEPROM_BASE_ADDR_WORD+2,&uw32NumRd1);// read the word data in 0x1e002
                                                          // out to variable uw32NumRd1

}
```

2.1.15 EepromWriteLongWordString()

This function writes a string of long words data to EEPROM. Be sure the start address is an even address.

Prototype of this function is:

```
void EepromWriteLongWordString(UWord32 wordAddr,UWord32* data, UWord16 length);
```

Table 20. EepromWriteLongWordString() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The start long word address of EEPROM that data string is written to. It must be even.
data	UWord32*	The long word pointer that points to a long word data string which is to be written into EEPROM
length	UWord16	Length of the string in unit of long words

2.1.16 EepromReadLongWordString()

This function reads a string of long words data out of EEPROM from a specified start long word address, which should be an even address.

Prototype of this function is:

```
void EepromReadLongWordString(UWord32 wordAddr,UWord32* data, UWord16 length);
```

Table 21. EepromReadLongWordString() function parameters

Parameter Name	Parameter Type	Description
wordAddr	UWord32	The start long word address of EEPROM that data string is read out of. It must be even.
data	UWord32*	The long word pointer that points to a long word data string, to which the read out long word string is stored
length	UWord16	Length of the string in unit of long words

Listing 7 on page 15 shows how to use *EepromWriteLongWordString()* and *EepromReadLongWordString()* to access EEPROM.

Listing 7. Use of EepromWriteLongWordString() and EepromReadLongWordString() to access EEPROM

```
#include "EepromDrv.h"
UWord32   uw32Num[16];
UWord32   uw32NumRd[16];
Word16    w16Stat;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  Word8 i;

  /* EEPROM initialization, 64 bytes of EEPROM with 16K bytes of FlexNVM backup */
  w16Stat = GetEepromInfo();

  if(((uw16EEESize&0x00ff) == 0xff) && ((uw16EEBackUpFlashSize&0x00ff) == 0xff)
       && (w16Stat == EEPROM_FLASHDRV_SUCCESS))
    {
     // Data Flash will be erased during partition
     // 64 bytes of EEPROM, with 16K bytes FlexNVM as backup
     w16Stat = DEFlashPartition(EEESIZE_64B,DEPART_16K);
  }

  for(i=0;i<16;i++)
  {
     uw16Num[i] += i;
  }
  /* 16 long words in uw32Num[0]~uw32Num[15] are written into EEPROM sequentially */
  EepromWriteLongWordString(EEPROM_BASE_ADDR_WORD,uw32Num,16);
  /* The long word data residing in EEPROM_BASE_ADDR_WORD to (EEPROM_BASE_ADDR_WORD+30) of
     EEPROM are read out and stored in uw32NumRd[0]~uw32NumRd[15] */
  EepromReadLongWordString(EEPROM_BASE_ADDR_WORD,uw32NumRd,16);

}
```

2.2 Description of EEPROM emulation driver for MC56F82xxx family

The Flash Driver Library described in AN4860 is used here to emulate EEPROM, so the related source files should be integrated in the IDE as well as *EepromDrv.c*, *EepromDrv.h* and *EepromDrv_cfg.h*. The driver used in CodeWarrior is shown in Figure 2. Several settings are necessary to use this driver:

In *EepromDrv_cfg.h*:

- Set EEPROM_EMULATION to 1 to enable drivers for MC56F82xxx family.

In *FlashDrv_cfg.h*:

- Set *FLASHDRV_FLSHCNT* to 1.
- Configure the size of Flash by setting *FLASHDRV_PRIMARY_START*, *FLASHDRV_PRIMARY_END* and *FLASHDRV_PRIMARY_SECTOR_SIZE* properly. *FLASHDRV_PRIMARY_SECTOR_SIZE* is always 0x200 for MC56F82xxx family, but *FLASHDRV_PRIMARY_END* may be different for different parts.
- Set *FLASHDRV_IWRT_ENABLE* to 1 to enable incremental flash writing feature.
- Set *FLASHDRV_IWRT_ERASE_ALL*” to 0 to make sure only a sector is erased once the memory that emulates EEPROM is full.

- Set *FLASHDRV_COPY2RAM* to 1 to make sure flash command executing functions are running in RAM.
- Set a reasonable number to *FLASHDRV_IWRT_SECT_CNT*, which decides how many sectors are used to emulate EEPROM. Make sure at least two sectors are used, or else there will be no backup, which is not quite safe in case of sudden power off. Three sectors are used in the example project.

Modify the linker file according to AN4860.

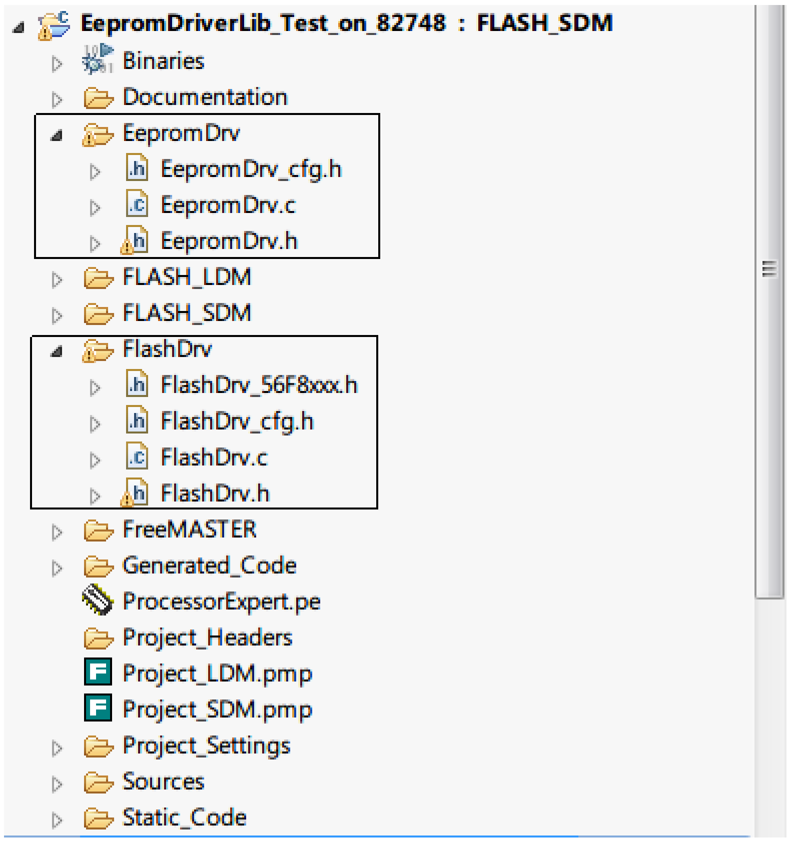

```
EepromDriverLib_Test_on_82748 : FLASH_SDM
    Binaries
    Documentation
    EepromDrv
        EepromDrv_cfg.h
        EepromDrv.c
        EepromDrv.h
    FLASH_LDM
    FLASH_SDM
    FlashDrv
        FlashDrv_56F8xxx.h
        FlashDrv_cfg.h
        FlashDrv.c
        FlashDrv.h
    FreeMASTER
    Generated_Code
    ProcessorExpert.pe
    Project_Headers
    Project_LDM.pmp
    Project_SDM.pmp
    Project_Settings
    Sources
    Static_Code
```

Figure 2. CodeWarrior projects view showing usage of EEPROM driver for MC56F82xxx family

In *FlashDrv_Cfg.h* file, there is a variable type definition:

```
typedef struct
{
    unsigned int dwEntryNum[200]; // user defined variables
    int dwCrcSum;
} FLASHDRV_IWRT_DATA_T, *LPFLASHDRV_IWRT_DATA_T;
```

The *unsigned int dwEntryNum [200]* can be substituted with any other variables according to your specific applications, any variable type will do, but make sure the size does not exceed 512 words. *" intdwCrcSum;"* must be kept unchanged because the driver uses this variable to store the CRC-16 code.

The EEPROM emulation is based on entry incremental writing feature with Erase Sector mode, refer to AN4860 for details. The structure of an entry is defined in *FlashDrv*.c:

```
typedef struct
{
    unsigned long int dwMark;    // Identificator of entry
    FLASHDRV_IWRT_DATA_T entry;    // User-defined incremental writing data structure
} FLASHDRV_IWRT_ENTRY_T, *LPFLASHDRV_IWRT_ENTRY_T;
```

A global entry variable is also defined in “FlashDrv.c”:

```
FLASHDRV_IWRT_ENTRY_T FLASHDRV_IWRT_ENTRY = {0, FLASHDRV_IWRT_DATA_INIT};
```

A pointer is defined to point to this entry variable:

```
FLASHDRV_IWRT_DATA_T *FLASHDRV_IWRT_DATA = &(FLASHDRV_IWRT_ENTRY.entry);
```

From user's perspective, there are two ways to access the user defined data in the entry. For example:

- *FLASHDRV_IWRT_DATA*->dwEntryNum [2]
- *FLASHDRV_IWRT_ENTRY*.entry.dwEntryNum [2]

Table 22. List of EEPROM emulation drivers for MC56F82xxx family

Function Name	Short Description
Crc_Init()	Enable the clock of CRC module. Inline function.
EepromDrv_Init()	Alias of FlashDrv_Init().
Crc_Calculation()	Calculate the crc-16 code of string of bytes
EepromDrv_Write()	Write an entry into EEPROM. Crc-16 code is calculated and written into EEPROM as part of the entry.
EepromDrv_Read()	Read the old entry out of EEPROM and store it to FLASHDRV_IWRT_ENTRY.entry. Crc-16 is checked.

2.2.1 Crc_Init()

This is an inline function that enables clock of CRC module in MC56F82xxx family.

```
#define Crc_Init() (UD_SIM_PCE2|=0x0020)
```

The CRC generator module uses the 16-bit CRC-CCITT polynomial, $x^{16}+x^{12}+x^5+1$ to generate a CRC code for error detection.

2.2.2 Crc_Calculation()

This function calculates the CRC code for a string of bytes using the CRC module described in Section 2.2.1. *Crc_Init()* must be invoked before calculation.

Prototype of this function is:

```
UWord16 Crc_Calculation(UWord8 *pbData, UWord16 w16Cnt);
```

The returned 16-bit data is the CRC code.

Table 23. Crc_Calculation() function parameters

Parameter Name	Parameter Type	Description
pbData	UWord8*	The byte pointer that points to a string of bytes which need CRC
w16Cnt	UWord16	Length of the string in unit of bytes

This function is invoked in *EepromDrv_Write()* and *EepromDrv_Read()*. You can use this function to calculate the CRC of other data because it is a general purpose function. Listing 8 on page 18 shows how to use it.

Listing 8. Use of Crc_Calculation() and to calculate CRC code for a string of bytes

```
#include "EepromDrv.h"
UWord8   uw8Data[12];
UWord16  uw16Crc;
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();

  Word16 i;

  Crc_Init(); // Enable CRC clock

  for(i=0; i<12; i++)
  {
      uw8Data[i] = i;
  }

  uw16Crc = Crc_Calculation(uw8Data,12);

}
```

2.2.3 EepromDrv_Write()

As described in the section Description of EEPROM emulation driver for MC56F82xxx family, a pointer called *FLASHDRV_IWRT_DATA* points to a structure variable *FLASHDRV_IWRT_ENTRY.entry*, which is composed of two parts: user defined data structure variables and a 16-bit CRC code *dwCrcSum*. In the example given in Description of EEPROM emulation driver for MC56F82xxx family, user-defined variable is an array of 200 words.

In the function of *EepromDrv_Write()*, firstly, a CRC code is calculated based on user defined variables in *FLASHDRV_IWRT_ENTRY.entry*, and stored in *FLASHDRV_IWRT_ENTRY.entry.dwCrcSum*. Secondly, write the whole *FLASHDRV_IWRT_ENTRY.entry* including CRC code to flash.

Prototype of this function is:

```
UWord8 EepromDrv_Write(void);
```

Table 24. EepromDrv_Write() function return codes

Return Code	Description
EEPROM_FLASHDRV_SUCCESS	Successfully write the entry to flash
EEPROM_FLASHDRV_ACCESS_ERROR	Function internal error
EEPROM_FLASHDRV_FAIL	MGSTAT0 bit of FTFL_FSTAT is set, meaning any errors have been encountered during the verify operation.
EEPROM_FLASHDRV_PROT_VIOLATION	Protection violation

The source code of this function is as below:

```
UWord8 EepromDrv_Write(void)
{
    Word16 w16Tmp;
    UWord8*  pbData;
    UWord8 ucResult;
    w16Tmp = sizeof(FLASHDRV_IWRT_DATA_T); // in unit of bytes
    w16Tmp -= 2;  // get the length of data string in unit of bytes
    pbData = (UWord8*)FLASHDRV_IWRT_DATA;
    FLASHDRV_IWRT_DATA->dwCrcSum = Crc_Calculation(pbData, w16Tmp); // get the crc
                                                                     // code
    ucResult = FlashDrv_IncWrite();     // write the data string and crc code into flash

    return ucResult;
}
```

2.2.4 EepromDrv_Read()

This function reads out the defined backup entry from flash and store it to *FLASHDRV_IWRT_ENTRY.entry*, including the CRC code. Then it calculates the CRC code of all data in *FLASHDRV_IWRT_ENTRY.entry*. The CRC code should be zero if the data is not corrupted.

Prototype of this function is:

```
UWord8 EepromDrv_Read(UWord16 uw16EntryAge);
```

Table 25. EepromDrv_Read() function parameters

Parameter Name	Parameter Type	Description
uw16EntryAge	UWord16	A number indicating which entry is to be read out. 0 means the latest entry.

Table 26. EepromDrv_Read() function return codes

Return Code	Description
EEPROM_FLASHDRV_SUCCESS	Successfully read the entry
EEPROM_FLASHDRV_ACCESS_ERROR	Invalid parameter. E.g. uw16EntryAge is too big and there's no valid entry.
EEPROM_CRC_ERROR	The CRC code of the read out entry is not zero.

The source code of the *EepromDrv_Read()* function is as below:

```
UWord8 EepromDrv_Read(UWord16 uw16EntryAge)
{
    Word16 w16Tmp,w16Crc;
    UWord8*  pbData;
    UWord8 ucResult;
    ucResult = FlashDrv_GetEntry(uw16EntryAge); // read the latest entry

    if(ucResult == FLASHDRV_ACCESS_ERROR)
    {
        return EEPROM_FLASHDRV_ACCESS_ERROR;
    }
    else
    {
        w16Tmp = sizeof(FLASHDRV_IWRT_DATA_T); // in unit of bytes
        w16Tmp -= 2;    // get the length of data string in unit of bytes
        pbData = (UWord8*)FLASHDRV_IWRT_DATA;

        // get the crc check code of data string and the stored crc result
        Crc_Calculation(pbData, w16Tmp);
        UD_CRC_CRCL = (FLASHDRV_IWRT_DATA->dwCrcSum >> 8) & 0x00ff;
        UD_CRC_CRCL = (FLASHDRV_IWRT_DATA->dwCrcSum) & 0x00ff;
        w16Crc = ((UD_CRC_CRCH<<8)|UD_CRC_CRCL);

        // crc check should be zero
        if(w16Crc == 0)
        {
            return EEPROM_FLASHDRV_SUCCESS;
        }
        else
        {
            return EEPROM_CRC_ERROR;
        }
    }
}
```

Listing 9 on page 20 shows how to use this driver to emulate EEPROM on MC56F82748. For example, if the user has 4 words, 3 long words and 2 bytes to be stored in EEPROM, the configuration in header file *FlashDrv_cfg.h* is as below:

```
/* Number of flash memories
 * - This is either one (MC56F827xx devices) or two (MC56F847xx devices) */

#define FLASHDRV_FLSHCNT                1
/* Primary flash parameters - program address space */
#define FLASHDRV_PRIMARY_START          0x00000000 UL // Word addresses
#define FLASHDRV_PRIMARY_END            0x00007FFFUL

#define FLASHDRV_PRIMARY_SECTOR_SIZE    0x0200UL     // Sector size (1kB)

#define      FLASHDRV_COPY2RAM                 1
/* Incremental flash writing
 * - This option enables incremental writing of fix-sized entries into
 * flash memory area, designated by user. */
#define      FLASHDRV_IWRT_ENABLE             1

/* -Number of dedicated sectors for incremental writing */
#define      FLASHDRV_IWRT_SECT_CNT           3
/* Size of memory to delete when memory is full
 * -This option determines whether erase an entire memory area (option is
 * enabled) or single sector (option is disabled) once the memory is full. */
#define      FLASHDRV_IWRT_ERASE_ALL          0

/* This structure contains the data, that will be stored, using incremental
 * writing */
typedef struct
{
       // user defined variables
       unsigned int uw16Num1;
       unsigned int uw16Num2;
       unsigned int uw16Num3;
       unsigned int uw16Num4;
       unsigned long uw32Num1;
       unsigned long uw32Num2;
       unsigned long uw32Num3;
       unsigned char uw8Num1;
       unsigned char uw8Num2;

       int dwCrcSum; // this variable is used by the driver, keep it.
} FLASHDRV_IWRT_DATA_T, *LPFLASHDRV_IWRT_DATA_T;
```

In this configuration, three sectors ranging from 0x7A00~0x7FFF are used as EEPROM backup.

Listing 9. Use of EEPROM emulation driver on MC56F82748

```
#include "EepromDrv.h"
UWord8   uw8Status;
UWord16  uw16Data[4];
UWord32  uw32Data[3];
UWord8   uw8Data[2];
void main(void)
{
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();

  Crc_Init(); // Enable CRC clock
```

```
    EepromDrv_Init(); // Flash increasing write initialization

    /* Update variable uw16Data[0]~ uw16Data[3],uw32Data[0]~uw32Data[2]
    and uw8Data[0]~ uw8Data[1]   */
    // ...

    /* Put the data into entry variable */
    FLASHDRV_IWRT_DATA->uw16Num1 = uw16Data[0];
    FLASHDRV_IWRT_DATA->uw16Num2 = uw16Data[1];
    FLASHDRV_IWRT_DATA->uw16Num3 = uw16Data[2];
    FLASHDRV_IWRT_DATA->uw16Num4 = uw16Data[3];
    FLASHDRV_IWRT_DATA->uw32Num1 = uw32Data[0];
    FLASHDRV_IWRT_DATA->uw32Num2 = uw32Data[1];
    FLASHDRV_IWRT_DATA->uw32Num3 = uw32Data[2];
    FLASHDRV_IWRT_DATA->uw8Num1 = uw8Data[0];
    FLASHDRV_IWRT_DATA->uw8Num2 = uw8Data[1];

    /* Store the data to EEPROM */
    uw8Status = EepromDrv_Write();

    /* Read out the data from EEPROM */
    /*
        FLASHDRV_IWRT_DATA->uw16Num1,
        FLASHDRV_IWRT_DATA->uw16Num2,
        FLASHDRV_IWRT_DATA->uw16Num3,
        FLASHDRV_IWRT_DATA->uw16Num4,
        FLASHDRV_IWRT_DATA->uw32Num1,
        FLASHDRV_IWRT_DATA->uw32Num2,
        FLASHDRV_IWRT_DATA->uw32Num3,
        FLASHDRV_IWRT_DATA->uw8Num1,
        FLASHDRV_IWRT_DATA->uw8Num2 are updated after reading

    */
    uw8Status = EepromDrv_Read(0);

    /* Use the saved data */
    uw16Data[0] = FLASHDRV_IWRT_DATA->uw16Num1;
    uw16Data[1] = FLASHDRV_IWRT_DATA->uw16Num2;
    uw16Data[2] = FLASHDRV_IWRT_DATA->uw16Num3;
    uw16Data[3] = FLASHDRV_IWRT_DATA->uw16Num4;
    uw32Data[0] = FLASHDRV_IWRT_DATA->uw32Num1;
    uw32Data[1] = FLASHDRV_IWRT_DATA->uw32Num2;
    uw32Data[2] = FLASHDRV_IWRT_DATA->uw32Num3;
    uw8Data[0] = FLASHDRV_IWRT_DATA->uw8Num1;
    uw8Data[1] = FLASHDRV_IWRT_DATA->uw8Num2;

}
```

3 Updating firmware without erasing EEPROM

It is available to update firmware without erasing EEPROM using CodeWarrior10.x.

- For MC56F84xxx family, EEPROM backup is stored in FlexNVM which ranges from 0x68000 to 0x6BFFF in program memory map. Avoid erasing this part of flash during programming.
- For MC56F82xxx family, EEPROM backup is stored in the top several sectors of program flash. Three sectors are used in Listing 9 on page 20 , which ranges from 0x7A00 to 0x7FFF in program memory map. Avoid erasing these sectors during programming.

A restricted range flash programming method in CodeWarrior 10.6 is introduced as below. Take MC56F84789 for example:

1. From the CodeWarrior IDE menu bar, select **Window > Show View > Other**. The **Show View** dialog box appears.
2. Expand the **Debug** tree control and select **Target Tasks**.

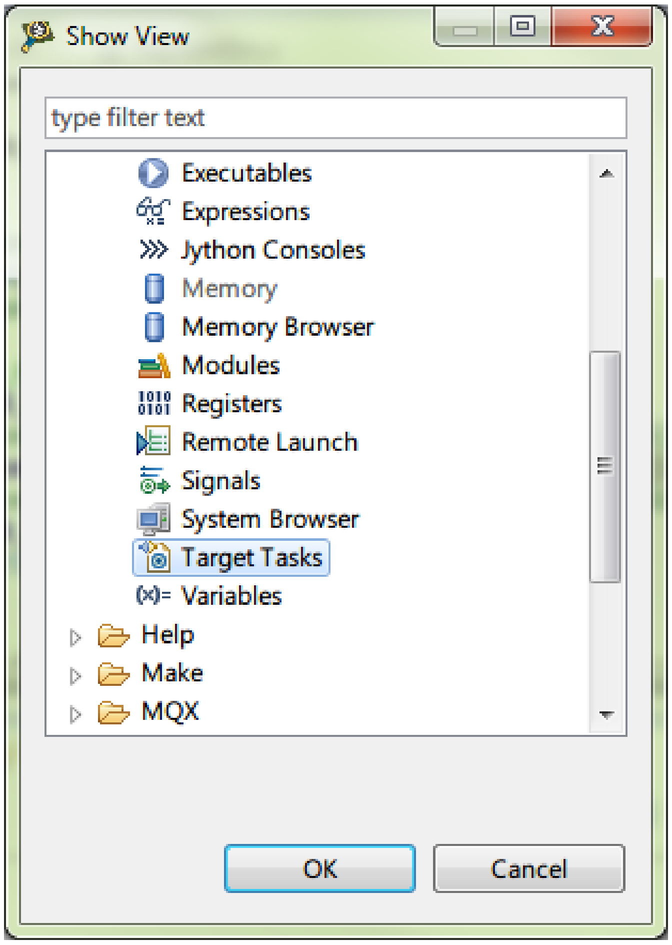


Figure 3. Show View dialog

3. Click **OK**.

   The **Target Tasks** view appears

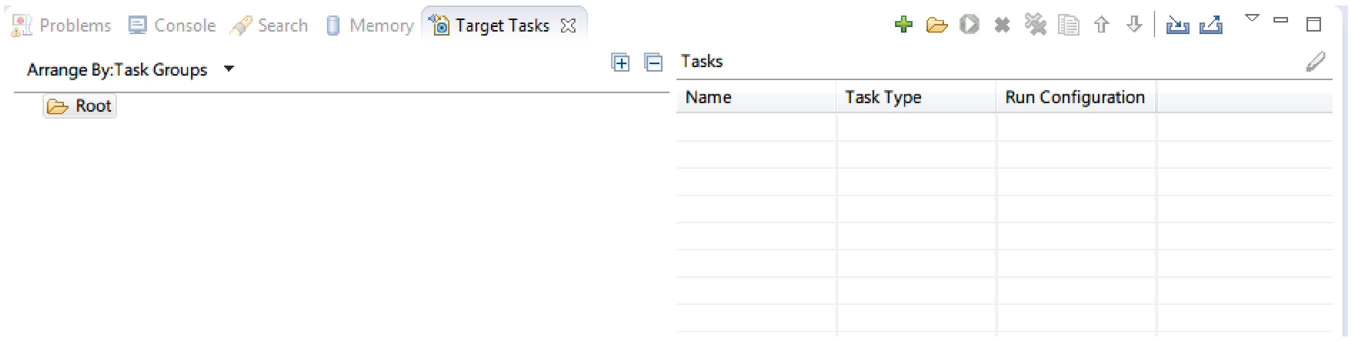


Figure 4. Target Tasks view

4. Right-click on **Root**, select **Import** from the context menu.

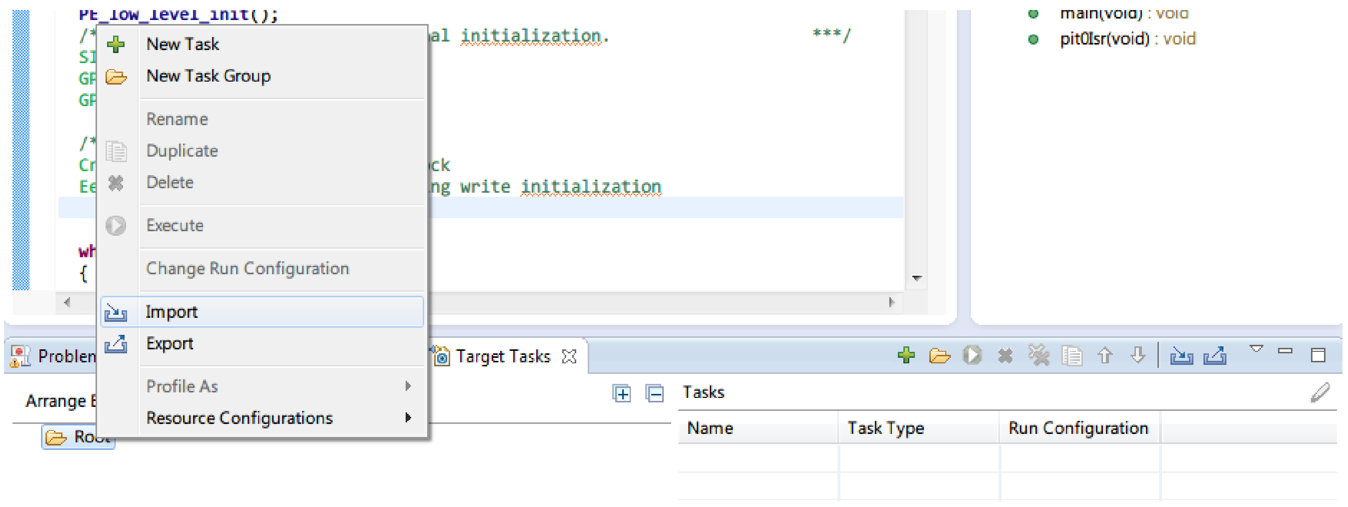
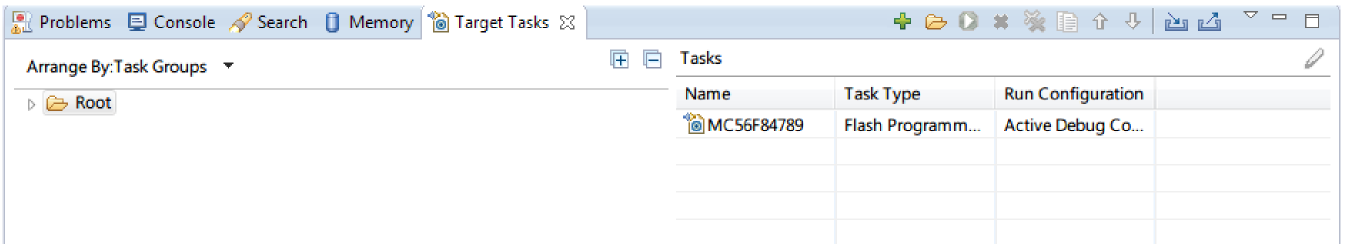


Figure 5. Context menu

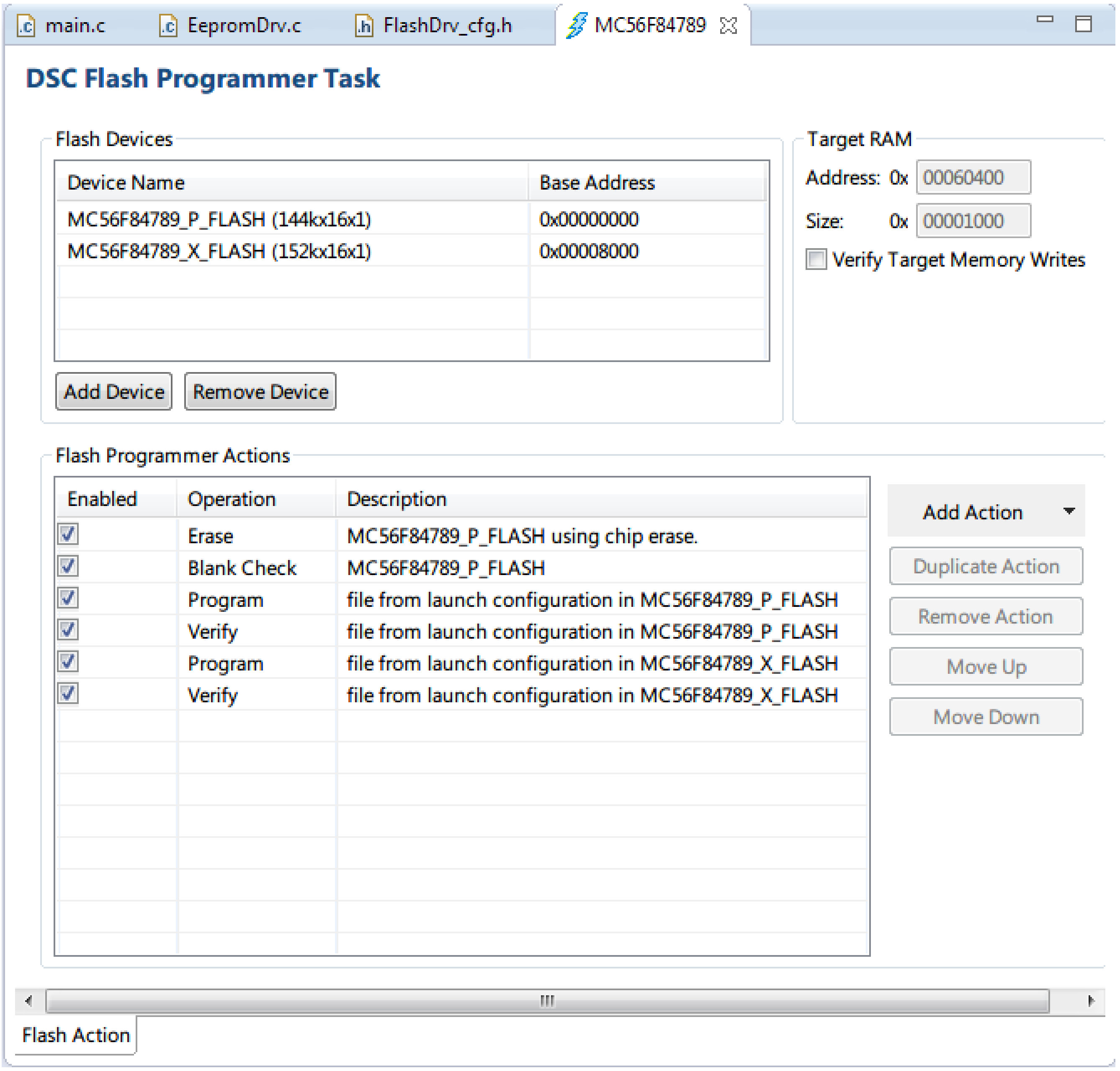
5. Navigate to the pre-defined tasks folder at `<CW MCU install>\MCU\bin\plugins\support\TargetTask\Flash_Programmer\` and select the desired .xml file. In this case, MC56F84789.xml is chosen.



Name	Task Type	Run Configuration
MC56F84789	Flash Programm...	Active Debug Co...

Figure 6. Choose MC56F84789.xml file

6. Double-click on the task's name. A tab of **DSC Flash Programmer Task** appears.



DSC Flash Programmer Task

Flash Devices

Device Name	Base Address
MC56F84789_P_FLASH (144kx16x1)	0x00000000
MC56F84789_X_FLASH (152kx16x1)	0x00008000

Target RAM

Address: 0x 00060400

Size: 0x 00001000

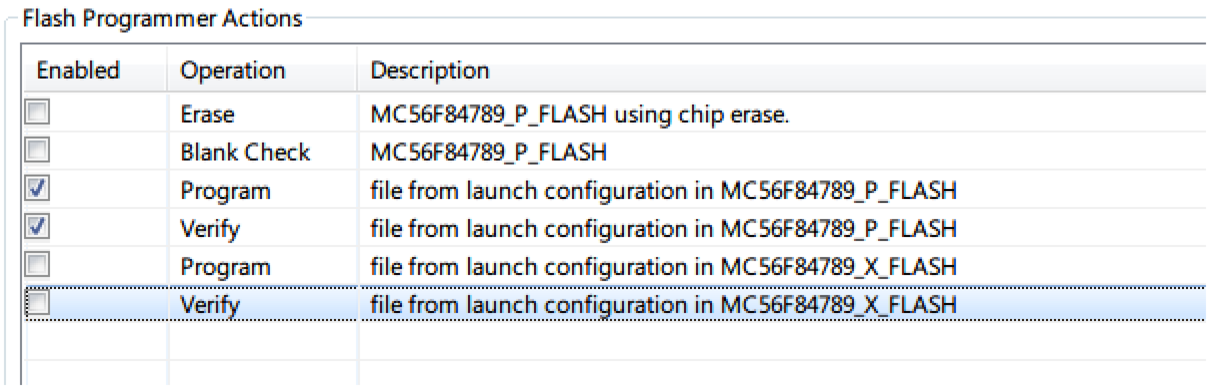
☐ Verify Target Memory Writes

Flash Programmer Actions

Enabled	Operation	Description
✓	Erase	MC56F84789_P_FLASH using chip erase.
✓	Blank Check	MC56F84789_P_FLASH
✓	Program	file from launch configuration in MC56F84789_P_FLASH
✓	Verify	file from launch configuration in MC56F84789_P_FLASH
✓	Program	file from launch configuration in MC56F84789_X_FLASH
✓	Verify	file from launch configuration in MC56F84789_X_FLASH

Figure 7. Flash programmer task editor window displaying stored actions

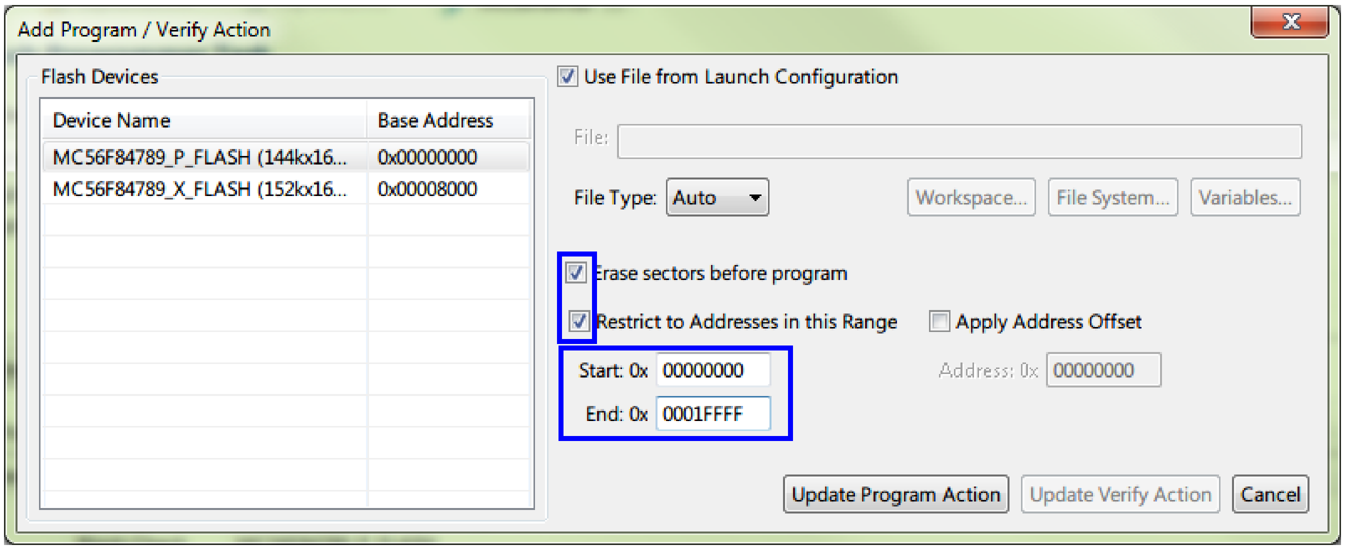
7. Uncheck the **Erase** and **Blank Check** actions. Also, Uncheck the **Program** and **Verify** actions from the launch configuration in MC56F84789_X_FLASH.


Flash Programmer Actions

Enabled	Operation	Description
☐	Erase	MC56F84789_P_FLASH using chip erase.
☐	Blank Check	MC56F84789_P_FLASH
✓	Program	file from launch configuration in MC56F84789_P_FLASH
✓	Verify	file from launch configuration in MC56F84789_P_FLASH
☐	Program	file from launch configuration in MC56F84789_X_FLASH
☐	Verify	file from launch configuration in MC56F84789_X_FLASH

Figure 8. Only keep program and verify actions for MC56F84789_P_FLASH

8. Double-click on the checked **Program** action. In the pop-up dialog, check **Erase sectors before program** option. Check **Restrict to Addresses in this Range** option. Specify the address range. The memory out of this range will not change. Click **Update Program Action** button to update the settings of this action.


Add Program / Verify Action

Flash Devices

Device Name	Base Address
MC56F84789_P_FLASH (144kx16...	0x00000000
MC56F84789_X_FLASH (152kx16...	0x00008000

✓ Use File from Launch Configuration

File:

File Type: Auto

✓ Erase sectors before program

✓ Restrict to Addresses in this Range ☐ Apply Address Offset

Start: 0x 00000000

End: 0x 0001FFFF

Address: 0x 00000000

Figure 9. Add Program/Verify Action dialog for Program Action

9. Double-click on the checked **Verify** action. In the pop-up dialog, check **Restrict to Addresses in this Range** option and specify the same address range as used in **Program** action.
10. Click **Update Verify Action** button to update the settings of this action.

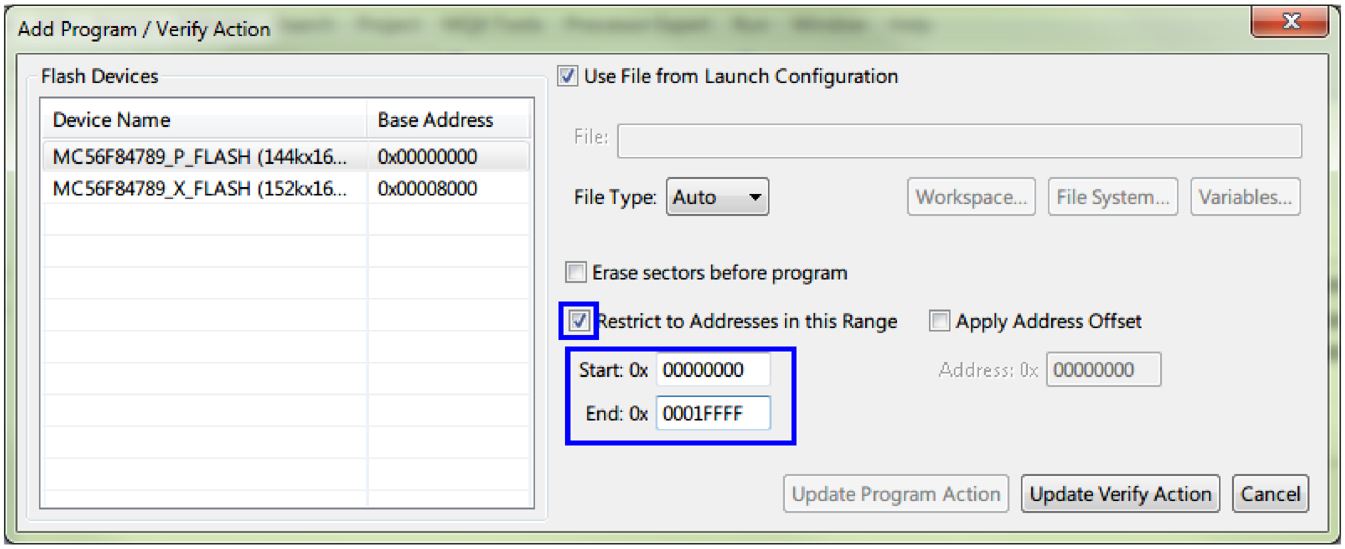


Figure 10. Add Program/Verify Action dialog for Verify Action

11. Right-click on the task name in **Target Tasks** view, and select **Change Run Configuration**.

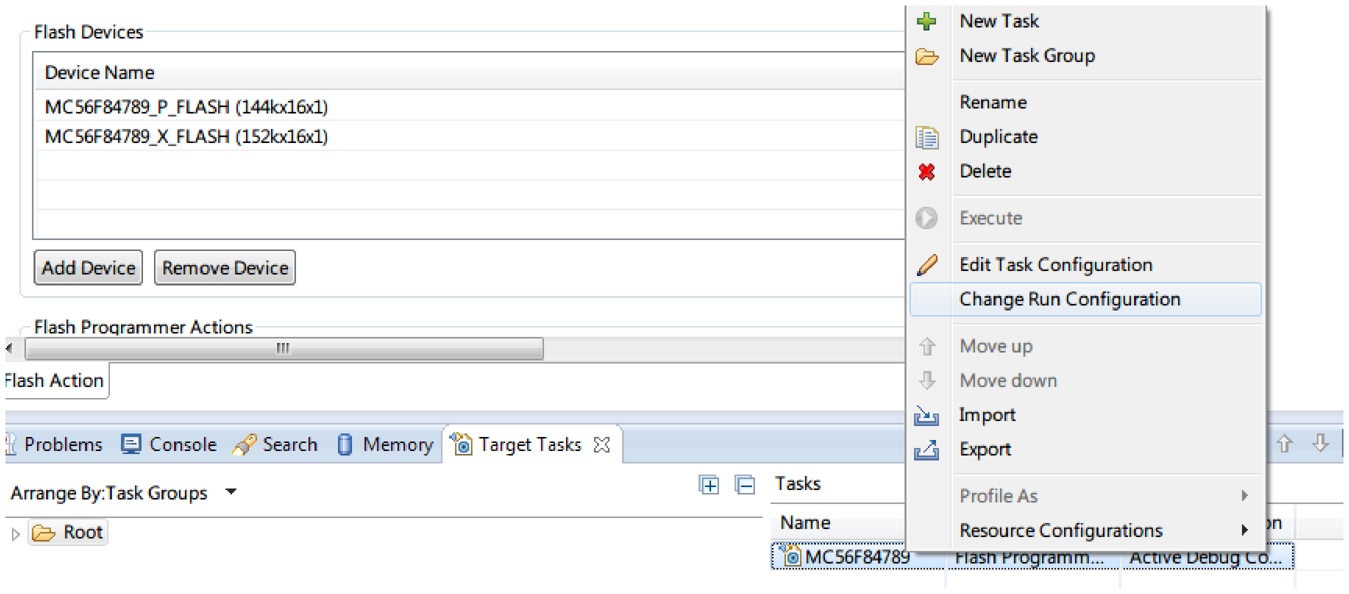
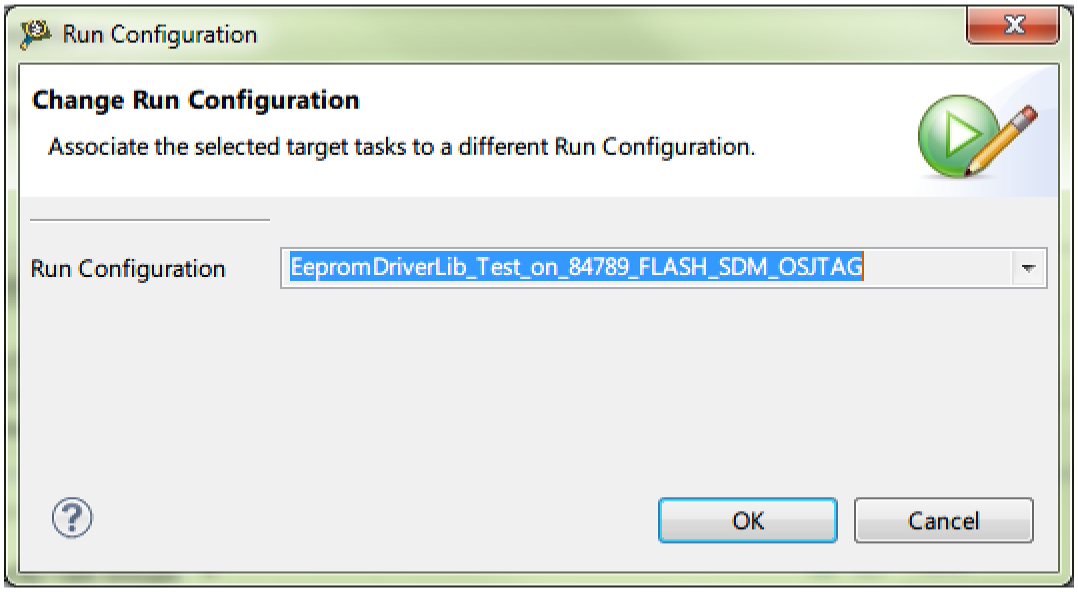


Figure 11. Change run configuration

12. The **Run Configuration** dialog appears. Select a run Configuration from the available configurations of the opened projects. Click **OK**



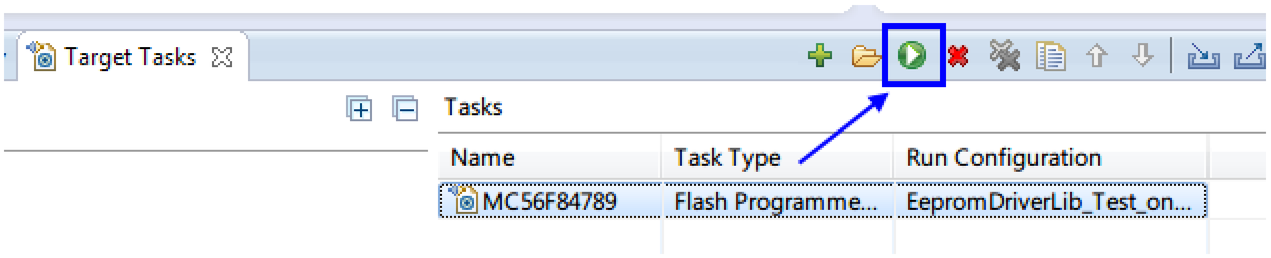
Change Run Configuration

Associate the selected target tasks to a different Run Configuration.

Run Configuration: EepromDriverLib_Test_on_84789_FLASH_SDM_OSJTAG

Figure 12. Run configuration dialog

13. Click **Execute** button to execute the actions.



Name	Task Type	Run Configuration
MC56F84789	Flash Programme...	EepromDriverLib_Test_on...

Figure 13. Execute the actions

Through the steps 1-13 above, only Program Flash is programmed, while FlexNVM remains the same. EEPROM is not affected in this way.

For MC56F82xxx family, since the last few sectors of Program Flash are used as backup for emulated EEPROM, change the addresses range in Figure 9 and Figure 10 accordingly. In Listing 9 on page 20 , since 0x7A00~0x7FFF are used as EEPROM backup, the addresses range in Figure 9 and Figure 10 should be changed to 0x0000~0x79FF to avoid erasing the contents in EEPROM.

4 Conclusion

There are three files in the EEPROM driver: *EepromDrv.c*, *EepromDrv.h* and *EepromDrv_cfg.h*. There's only one macro in *EepromDrv_cfg.h* which is used to define whether the driver is for MC56F84xxx family or MC56F82xxx family.

- Set EEPROM_EMULATION to 0 in *EepromDrv_cfg.h* file to enable drivers for MC56F84xxx family. Description of EEPROM driver for MC56F84xxx family shows the configuration of the driver.
- Set EEPROM_EMULATION to 1 in *EepromDrv_cfg.h* file to enable drivers for MC56F82xxx family. Meanwhile, the FDL driver described in AN4860 should also be included in the project, as shown in Figure 2. The configuration of FDL is described in Description of EEPROM emulation driver for MC56F82xxx family. Remember to modify the linker file as indicated in AN4860.

For MC56F84xxx family, a build-in filing system performs the EEPROM characteristics automatically, so the performance is more sophisticated. For MC56F82xxx family, there's no such system, so FDL is used to emulate EEPROM, together with CRC-16 function and incremental entry writing feature, the reliability and flash cycling endurance is also improved.

Document Number AN5074
Revision 0, 01/2015