Freescale Semiconductor
应用笔记

Document Number: AN5074
Rev 0, 01/2015

MC56F84xxx 和 MC56F82xxx DSC 系列的 EEPROM 驱动程序

作者: Xuwei Zhou

内容

1 简介

在 Freescale MC56F84xxx DSC 系列中，部分或全部 FlexNVM 以及 FlexRAM（1K 字的 RAM 数据块）可用于利用内置式文件归档系统（build-in filing system）来模拟 EEPROM 的特性。正确配置 EEPROM 后，用户可以使用 FlexRAM 来写入此 EEPROM 或从此 EEPROM 进行读取。文件归档系统负责所有记录备份工作，用户可以不用理会这些工作。

请参阅《MC56F847xx 参考手册》或“AN4689：在 MC56F84xxx DSC 上使用 EEPROM”了解更多信息。

在 MC56F82xxx DSC 系列中，FlexNVM 或 FlexRAM 不可用。如果需要 EEPROM，则必须通过固件在程序 Flash（Program Flash）上进行模拟。

该应用笔记介绍了 MC56F84xxx 和 MC56F82xxx DSC 系列的 EEPROM 驱动程序。您可以按照此应用笔记中的指南来直接使用该驱动程序。该应用笔记还介绍了一种在 CodeWarrior10.6 中编程的方法，借助该方法，无需擦除 EEPROM 即可对 Flash 重新编程。

对于 MC56F84xxx DSC 系列，该驱动程序适用于小数据模式（Small Data Mode）也适用于大数据模式（Large Data Mode），因为它是用汇编语言编写的。AN4689 也为 MC56F84xxx DSC 系列中的 EEPROM 提供了驱动程序，但它只适用于大数据模式。至于 MC56F82xxx DSC 系列，该驱动程序采用 AN4860“MC56F847xx 和 MC56F827xx DSC 系列的 Flash 驱动程序库”一文中所述的 Flash 驱动程序库与 CRC 功能来模拟 EEPROM，以增加可靠性。

2 EEPROM 驱动程序说明

为了更简单有效地使用 DSC 中的 EEPROM，我们开发了该驱动程序。对于 MC56F84xxx 系列，提供了具有字节串、字串和长字串读写功能的 API，以及字节、字、长字读写功能的 API。对于 MC56F82xxx 系列，该驱动程序在擦除扇区模式使用递增写入功能进行开发。每次向 Flash 中写入条目或从 Flash 读取条目时，都会执行 CRC，以便提高可靠性。

2.1 MC56F84xxx 系列的 EEPROM 驱动程序说明

在 *EepromDrv_cfg.h* 文件中将 EEPROM_EMULATION 设为 0 时，可使能用于 MC56F84xxx 系列的驱动程序。所有函数都采用汇编语言编写，以适应小数据模式和大数据模式，并提高执行效率。表 1 列出了 MC56F84xxx 系列中可供用户使用的所有函数。此驱动程序在 *EepromDrv.c* 和 *EepromDrv.h* 中实现。CodeWarrior 中使用的驱动程序如图 1 所示。

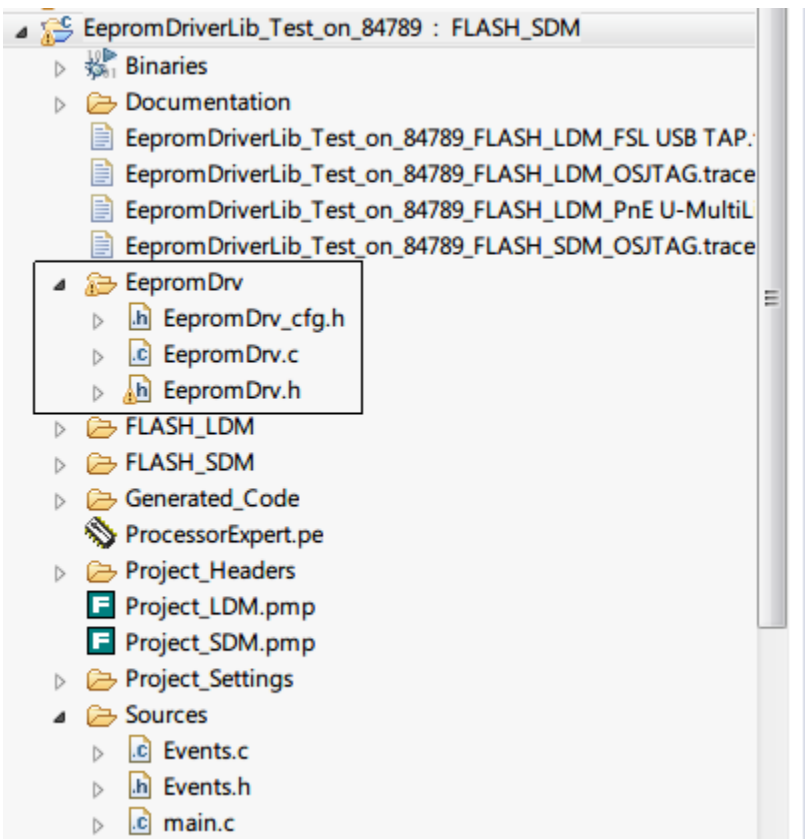

图 1. CodeWarrior 项目视图，显示了 MC56F84xxx 系列的 EEPROM 驱动程序的用法

表 1. MC56F84xxx 系列的 EEPROM 驱动程序列表

函数名称	简短说明
GetEepromInfo()	获取 EEPROM 和备份 FlexNVM 的大小
DEFlashPartition()	设置 EEPROM 和备份 FlexNVM 的大小

下一页继续介绍此表...

表 1. MC56F84xxx 系列的 EEPROM 驱动程序列表 (继续)

函数名称	简短说明
SetEEEEnable()	将 FlexRAM 配置为 EEPROM 的接口
SetEEEDisable()	将 FlexRAM 设为常规 RAM，无 EEPROM 功能
EepromWriteByte()	向 EEPROM 中的某个地址写入一个字节
EepromReadByte()	从 EEPROM 中的某个地址读取一个字节
EepromWriteWord()	向 EEPROM 中的某个地址写入一个字
EepromReadWord()	从 EEPROM 中的某个地址读取一个字
EepromWriteLongWord()	向 EEPROM 中的某个地址写入一个长字
EepromReadLongWord()	从 EEPROM 中的某个地址读取一个长字
EepromWriteByteString()	向 EEPROM 中写入一串字节
EepromReadByteString()	从 EEPROM 中读取一串字节
EepromWriteWordString()	向 EEPROM 中写入一串字
EepromReadWordString()	从 EEPROM 中读取一串字
EepromWriteLongWordString()	向 EEPROM 中写入一串长字
EepromReadLongWordString()	从 EEPROM 中读取一串长字

所有函数都使用 *EepromDrv.h* 中定义的以下数据类型:

- UWord8 – 无符号字节。范围: [0, 255]
- UWord16 – 无符号字（双字节）。范围: [0, 2^{16})
- UWord32 – 无符号长字（四字节）。范围: [0, 2^{32})

EepromDrv.h 中也定义了返回代码。只有表 1 中的前 4 个函数具有返回代码。相关列表请参见表 2 。

表 2. MC56F84xxx 系列的 EEPROM 驱动程序返回代码

返回代码	定义值
EEPROM_FLASHDRV_SUCCESS	0
EEPROM_FLASHDRV_FAIL	1
EEPROM_FLASHDRV_ACCESS_ERROR	2
EEPROM_FLASHDRV_PROT_VIOLATION	3

您可以将 FlexNVM 分区成两个部分：用于 EEPROM 的备份和数据 Flash（Data Flash）。程序 Flash（Program Flash）和数据 Flash（Data Flash）都各自有一小块非易失性信息寄存器（non-volatile information registers），称为 IFR，该寄存器与主存储器阵列分离。数据 Flash 的 IFR 具有 256 字节，其中 2 字节包含 EEPROM 的相关信息:

- EEESIZE: 该字节中的四个最低有效位确定每个可用的 EEPROM 子系统中使用的 FlexRAM 大小。可用值在 *EepromDrv.h* 中定义，如表 3 所列。

表 3. EEESIZE 的可用值

代码	定义值	EEPROM 大小	在数据存储器映像中 FlexRAM 的字访问地址
EEESIZE_2048B	0x33	2048 字节	0x1e000 ~ 0x1e3ff
EEESIZE_1024B	0x34	1024 字节	0x1e000 ~ 0x1e1ff

下一页继续介绍此表...

表 3. EEESIZE 的可用值 (继续)

代码	定义值	EEPROM 大小	在数据存储器映像中 FlexRAM 的字访问地址
EEESIZE_512B	0x35	512 字节	0x1e000 ~ 0x1e0ff
EEESIZE_256B	0x36	256 字节	0x1e000 ~ 0x1e07f
EEESIZE_128B	0x37	128 字节	0x1e000 ~ 0x1e03f
EEESIZE_64B	0x38	64 字节	0x1e000 ~ 0x1e01f
EEESIZE_32B	0x39	32 字节	0x1e000 ~ 0x1e00f
EEESIZE_0B	0x3F	0	N/A

- DEPART：该字节中的四个最低有效位指定用作 EEPROM 备份存储器的 FlexNVM 大小。可用值在 *EepromDrv.h* 中定义，如表 4 所列。

表 4. DEPART 的可用值

代码	定义值	EEPROM 的备份大小	数据 Flash 的大小
DEPART_0K	0x0	无 EEPROM 备份	32K 字节
DEPART_8K	0x1	8K 字节	24K 字节
DEPART_16K	0x2	16K 字节	16K 字节
DEPART_24K	0x9	24K 字节	8K 字节
DEPART_32K	0x3	32K 字节	无剩余

在芯片的复位过程中，EEESIZE 和 DEPART 的值决定了是否对 FlexNVM 分区以提供 EEPROM 的备份。如果是，则会将 EEPROM 的备份数据复制到配置的 FlexRAM，并且 FTFL_FCNFG 寄存器中的 EEERDY 标志位将置位。否则，FlexRAM 用作常规 RAM，并且 FTFL_FCNFG 寄存器中的 RAMRDY 标志位将置位。

只有当数据 Flash IFR 已处于擦除状态，并且 EEESIZE 和 DEPART 的值为 0xFF 时，数据 Flash IFR 中的 EEESIZE 和 DEPART 字节才可被 FTFL 模块中的 Program Partition 命令修改。Erase All Blocks 命令和触发 Erase All 命令的外部请求（这个请求主要从与 JTAG 口连接的下载器而来）都可以清除数据 Flash 和程序 Flash 的 IFR。关于 FTFL 命令的更多信息，请参见《MC56F847xx 参考手册》。

该驱动程序里有两个全局变量 *uw16EEESize* 和 *uw16EEBackUpFlashSize*，用于存储 EEESIZE 和 DEPART 的值。

关于本节所列所有函数的性能，请参见《MC56F847xx 高级信息数据手册》中的表 24“Flash 命令时序特性”。

2.1.1 GetEepromInfo()

该函数会读取数据 Flash 的 IFR，以获得 EEESIZE 和 DEPART 的值。读取通过执行 Read Resource 命令来实现，并且值会存储到变量 *uw16EEESize* 和 *uw16EEBackUpFlashSize* 中。应该在使用 EEPROM 之前调用一次该函数。通过 *uw16EEESize* 和 *uw16EEBackUpFlashSize* 的值可以判断 EEPROM 是否已配置：

- *uw16EEESize == 0xFF* 且 *uw16EEBackUpFlashSize == 0xFF*：无备份 FlexNVM，EEPROM 未配置。
- *uw16EEESize != 0xFF* 或 *uw16EEBackUpFlashSize != 0xFF*：EEPROM 已配置。这两个变量的四个最低有效位反映 EEPROM 和备份 FlexNVM 的大小。

该函数的原型为：

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

表 5. GetEepromInfo()函数返回代码

返回代码	说明
EEPROM_FLASHDRV_SUCCESS	成功获取 EEPROM 信息
EEPROM_FLASHDRV_ACCESS_ERROR	函数内部错误

清单 1 显示了如何使用 *GetEepromInfo()*进行 EEPROM 初始化。

2.1.2 DEFlashPartition()

如果未在初始化时配置 EEPROM，该函数会通过对 EEESIZE 和 DEPART 进行编程，以配置 EEPROM 和备份 FlexNVM 的大小。此时会执行 Program Partition 命令，该命令会将 FlexNVM 配置为数据 Flash、EEPROM 备份，或两者的组合，并初始化 FlexRAM。

Program Partition 命令启动后，便会检查数据 Flash IFR 中的 EEESIZE 和 DEPART，确认它们是否已擦除。如果已擦除，该命令会擦除 FlexNVM 存储器中的内容，并相应地对 FlexNVM 进行分区，以提供 EEPROM 的备份。分配的 EEPROM 备份扇区会进行格式化，以供 EEPROM 使用。最后，会将表 6 中的分区代码编入数据 Flash IFR。此命令还会在编程后验证是否可以正确回读分区代码。如果 FlexNVM 成功地进行了 EEPROM 备份分区，FTFL_FCNFG 中的 EEERDY 标志位会置位。

该函数的原型为:

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

下表列出了函数参数和返回代码。

表 6. DEFlashPartition()函数参数

参数名称	参数类型	说明
EEEDataSize	UWord8	配置 EEPROM 的大小。使用表 3 中的代码。
EEBackUpFlashSize	UWord8	配置备份数据 Flash 的大小。使用表 4 中的代码。

表 7. DEFlashPartition()函数返回代码

返回代码	说明
EEPROM_FLASHDRV_SUCCESS	成功配置 EEPROM
EEPROM_FLASHDRV_ACCESS_ERROR	函数内部错误
EEPROM_FLASHDRV_FAIL	FTFL_FSTAT 的 MGSTAT0 位置位，表明验证操作期间发生了错误。

避免以下操作，否则会出现 EEPROM_FLASHDRV_ACCESS_ERROR:

- 在 EEPROM 已配置时调用 *DEFlashPartition()*，即 EEESIZE != 0xFF 或 DEPART != 0xFF。

- 将代码 DEPART_0K 传递到 EEBackUpFlashSize，并将一个分配 FlexRAM 以供 EEPROM 使用的代码传递到 EEEDataSize。
- 将代码 EEESIZE_0B 传递到 EEEDataSize，并将一个为 EEPROM 备份分配空间的代码传递到 EEBackUpFlashSize。

清单 1. 使用 GetEepromInfo()和 DEFlashPartition()函数进行 EEPROM 初始化

```
#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()

该函数使能 FlexRAM 作为 EEPROM 的接口。此时会执行 Set FlexRAM Function 命令，使得 FlexRAM 可用于 EEPROM。该命令完成时，EEPROM 备份空间中的现有 EEPROM 数据会通过 flash 模块复制到 FlexRAM， 并且 FTFL_FCNFG 中的 EEERDY 标志位会置位，RAMRDY 标志位则会清零。这种情况下，可以正常地读写 FlexRAM，但写入 FlexRAM 还会引起 EEPROM 的相关活动。使用此应用笔记提供的 EEPROM 读写函数来操作 EEPROM。

该函数的原型为:

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

表 8. SetEEEEnable()函数返回代码

返回代码	说明
EEPROM_FLASHDRV_SUCCESS	成功使能 FlexRAM 作为到 EEPROM 的接口
EEPROM_FLASHDRV_ACCESS_ERROR	函数内部错误

注

*DEFlashPartition()*成功执行并且 FlexRAM 已配置为 EEPROM 的接口时，就不必在 *DEFlashPartition()*之后立即调用 *SetEEEEnable()* 。

2.1.4 SetEEEDisable()

该函数将 FlexRAM 设置为传统 RAM，而不是 EEPROM 的接口。此时会执行 Set FlexRAM Function 命令，使 FlexRAM 用作传统 RAM。该命令完成时，会通过 flash 模块向整个 FlexRAM 写入 1，并且 FTFL_FCNFG 中的 RAMRDY 标志位置位，EEERDY 标志位则清零。这种情况下，可以正常地读写 FlexRAM。

该函数的原型为:

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

表 9. SetEEEDisable()函数返回代码

返回代码	说明
EEPROM_FLASHDRV_SUCCESS	成功将 FlexRAM 设置为传统 RAM
EEPROM_FLASHDRV_ACCESS_ERROR	函数内部错误

注

当 FlexRAM 配置为传统 RAM 时，可使用 LDM 来访问它，因为 FlexRAM 的初始字地址是 0x1E000，已超过 16 位。或者使用 *EepromDrv.h* 中的内联函数来访问 FlexRAM，这对 SDM 和 LDM 均适用。

清单 2 第 页上 7 显示了如何使用 *SetEEEEnable()*和 *SetEEEDisable()*来更改 FlexRAM 的角色

清单 2. 使用 SetEEEEnable()和 SetEEEDisable()来更改 FlexRAM 的角色。

```
#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()

正确配置 EEPROM 后，可使用该函数将一个字节（8 位）写入 EEPROM 中的目标地址。

该函数的原型为:

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

表 10. EepromWriteByte()函数参数

参数名称	参数类型	说明
byteAddr	UWord32	EEPROM 的地址
data	UWord8	写入 EEPROM 的字节数据

例如，如果采用常数 *EEESIZE_64B* 将 EEPROM 大小配置为 64 字节， 可用的 EEPROM 字节地址范围为 0x3C000~0x3C03F。*Eeprom.h* 中有宏定义：

```
#define EEPROM_BASE_ADDR_BYTE 0x3c000
```

当按字节访问 EEPROM 时，你可以使用 EEPROM_BASE_ADDR_BYTE 作为基地址。

2.1.6 EepromReadByte()

该函数用于从 EEPROM 中指定的字节地址读取一个字节。

该函数的原型为：

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

表 11. EepromReadByte()函数参数

参数名称	参数类型	说明
byteAddr	UWord32	EEPROM 的地址
data	UWord8*	字节指针。读取的字节会存储到该指针指向的地方

清单 2 也显示了如何使用 *EepromWriteByte()*和 *EepromReadByte()*写入一个字节到 EEPROM，以及从 EEPROM 读取一个字节。

2.1.7 EepromWriteByteString()

该函数会将一串字节数据写入 EEPROM。

该函数的原型为：

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

表 12. EepromWriteByteString()函数参数

参数名称	参数类型	说明
byteAddr	UWord32	数据串写入 EEPROM 的起始地址
data	UWord8*	指向要写入 EEPROM 的字节数据串的字节指针
length	UWord16	数据串长度，以字节为单位

2.1.8 EepromReadByteString()

该函数会从 EEPROM 中指定的起始字节地址读取一串字节数据。

该函数的原型为:

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

表 13. EepromReadByteString()函数参数

参数名称	参数类型	说明
byteAddr	UWord32	从 EEPROM 读取数据串的起始地址
data	UWord8*	指向字节数据串的字节指针，读出的字节数据存储在此数据串中
length	UWord16	字符串长度，以字节为单位

清单 3 第 页上 9 显示了如何使用 *EepromWriteByteString()*和 *EepromReadByteString()*访问 EEPROM。

清单 3. 使用 EepromWriteByteString()和 EepromReadByteString()访问 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()

与 *EepromWriteByte()*函数不同，此函数会向 EEPROM 中写入一个 16 位的字数据。区别在于 EEPROM 的地址。例如，如果采用常数 *EEESIZE_64B* 将 EEPROM 大小配置为 64 字节， 可用的 EEPROM 字地址范围为 0x1E000~0x1E01F。

在 *Eeprom.h* 中有宏定义:

```
#define EEPROM_BASE_ADDR_WORD 0x1e000
```

当按字访问 EEPROM 时，你可以使用 EEPROM_BASE_ADDR_WORD 作为基地址。

该函数的原型为:

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

表 14. EepromWriteByte()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	EEPROM 的地址
data	UWord16	写入 EEPROM 的字数据

注

字节地址 EEPROM_BASE_ADDR_BYTE 和（EEPROM_BASE_ADDR_BYTE+1）实际上是指字地址 EEPROM_BASE_ADDR_WORD 的最低有效字节和最高有效字节。其余可以用相同的方式来理解字节地址和字地址之间的关系。

2.1.10 EepromReadWord()

该函数用于从 EEPROM 中指定的字地址读取一个字。

该函数的原型为:

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

表 15. EepromReadByte()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	EEPROM 的地址
data	UWord16*	字指针。读取的字会存储到该指针指向的地方

清单 4 第 页上 10 显示了如何使用 *EepromWriteWord()*和 *EepromReadWord()*写入一个字到 EEPROM，以及从 EEPROM 读取一个字。

清单 4. 使用 EepromWriteWord()和 EepromReadWord()访问 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()

该函数会将一串字数据写入 EEPROM。

该函数的原型为:

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

表 16. EepromWriteWordString()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	数据串写入 EEPROM 的起始字地址
data	UWord16*	指向要写入 EEPROM 的字数据串的字指针
length	UWord16	数据串长度，以字为单位

2.1.12 EepromReadWordString()

该函数会从 EEPROM 中指定的起始字地址读取一串字数据。

该函数的原型为:

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

表 17. EepromReadWordString()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	从 EEPROM 读取数据串的起始字地址
data	UWord16*	指向字数据串的字指针，指明读取的字串的存储地址
length	UWord16	字符串长度，以字为单位

清单 5 显示了如何使用 *EepromWriteWordString()*和 *EepromReadWordString()*访问 EEPROM。

清单 5. 使用 EepromWriteWordString()和 EepromReadWordString()访问 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()

与 *EepromWriteByte()*函数不同，此时会向 EEPROM 中写入一个 32 位的长字数据。区别在于 EEPROM 的地址。例如，如果采用常数 *EEESIZE_64B* 将 EEPROM 大小配置为 64 字节，则意味着该空间可以保存最多 16 个长字。可用的 EEPROM 长字地址序列如下:

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

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

请注意，EEPROM 中的长字访问地址应该是偶数。

该函数的原型为:

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

表 18. EepromWriteLongWord()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	EEPROM 的长字地址。它必须是偶数。
data	UWord32	写入 EEPROM 的长字数据

2.1.14 EepromReadLongWord()

该函数用于从 EEPROM 中指定的长字地址读取一个长字。

该函数的原型为:

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

表 19. EepromReadByte()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	EEPROM 的长字地址。它必须是偶数。
data	UWord32*	长字指针。读取的长字会存储到该指针指向的地方

清单 6 显示了如何使用 *EepromWriteLongWord()*和 *EepromReadLongWord()*写入一个长字到 EEPROM，以及从 EEPROM 读取一个长字。

清单 6. 使用 EepromWriteLongWord()和 EepromReadLongWord()访问 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()

该函数会将一串长字数据写入 EEPROM。务必确保起始地址为偶数地址。

该函数的原型为:

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

表 20. EepromWriteLongWordString()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	数据串写入 EEPROM 的起始长字地址。它必须是偶数。
data	UWord32*	指向要写入 EEPROM 的长字数据串的长字指针
length	UWord16	数据串长度，以长字为单位

2.1.16 EepromReadLongWordString()

此函数会从 EEPROM 中指定的起始长字地址读取一串长字数据，该地址应为偶数地址。

该函数的原型为:

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

表 21. EepromReadLongWordString()函数参数

参数名称	参数类型	说明
wordAddr	UWord32	从 EEPROM 读取数据串的起始长字地址。它必须是偶数。
data	UWord32*	指向长字数据串的长字指针，指明读取的长字串的存储地址
length	UWord16	字符串长度，以长字为单位

清单 7 第 页上 14 显示了如何使用 *EepromWriteLongWordString()*和 *EepromReadLongWordString()*访问 EEPROM。

清单 7. 使用 EepromWriteLongWordString()和 EepromReadLongWordString()访问 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 MC56F82xxx 系列的 EEPROM 仿真驱动程序说明

此处使用 AN4860 中介绍的 Flash 驱动程序库来模拟 EEPROM, 因此相关的源文件应该和 *EepromDrv.c*、*EepromDrv.h* 以及 *EepromDrv_cfg.h* 一起集成在 IDE 中。CodeWarrior 中使用的驱动程序如图 2 所示。使用该驱动程序必须进行几项设置:

在 *EepromDrv_cfg.h* 中:

- 将 EEPROM_EMULATION 设为 1，以使能用于 MC56F82xxx 系列的驱动程序。

在 *FlashDrv_cfg.h* 中:

- 将 *FLASHDRV_FLSHCNT* 设为 1。
- 通过合理设置 *FLASHDRV_PRIMARY_START*、*FLASHDRV_PRIMARY_END* 和 *FLASHDRV_PRIMARY_SECTOR_SIZE*，可配置 Flash 的大小。对于 MC56F82xxx 系列，*FLASHDRV_PRIMARY_SECTOR_SIZE* 始终为 0x200，但是不同部件的 *FLASHDRV_PRIMARY_END* 可能有所不同。
- 将 *FLASHDRV_IWRT_ENABLE* 设为 1，以使能 flash 递增写功能。
- 将 *FLASHDRV_IWRT_ERASE_ALL* 设为 0，以确保当模拟 EEPROM 的存储器已满时只会擦除一个扇区。
- 将 *FLASHDRV_COPY2RAM* 设为 1，以确保 flash 命令执行函数在 RAM 中运行。
- 为 *FLASHDRV_IWRT_SECT_CNT* 设置合理的值，这将决定用于模拟 EEPROM 的扇区数量。确保至少有两个扇区在使用，否则将不会备份，如果突然断电就不太安全。示例项目中使用了三个扇区。

根据 AN4860 的说明修改链接器文件。

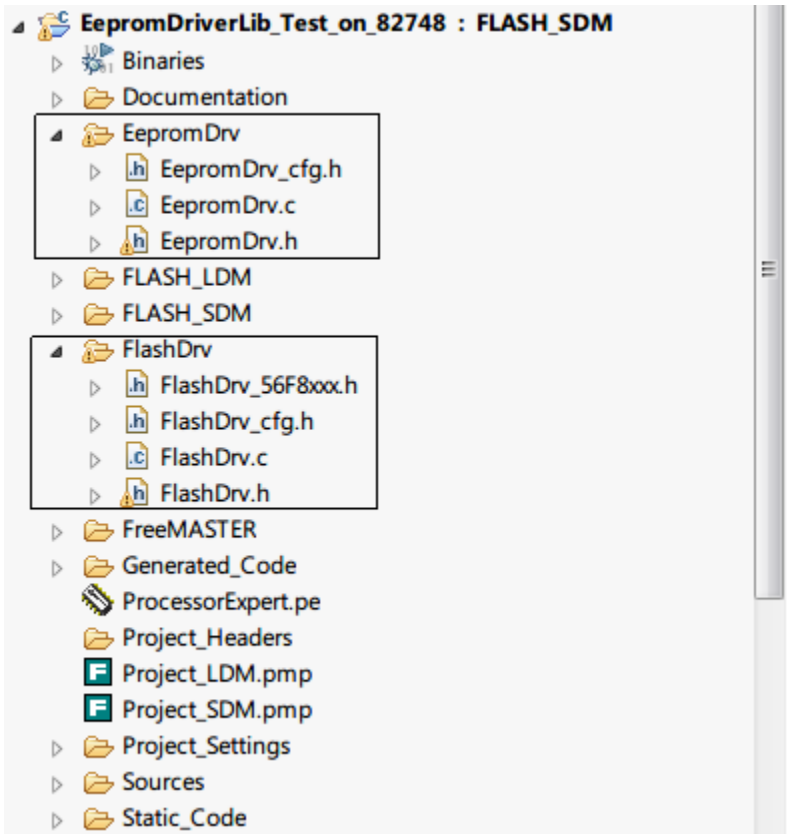

图 2. CodeWarrior 项目视图，显示了 MC56F82xxx 系列的 EEPROM 驱动程序的用法

FlashDrv_Cfg.h 文件中有变量类型定义：

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

根据您的具体应用，可使用任何其他变量来替代 *unsigned int dwEntryNum [200]*，任何变量类型都可以，但是应确保大小不超过 512 个字。*" intdwCrcSum;"*必须保持不变，因为驱动程序使用该变量来存储 CRC-16 代码。

EEPROM 仿真基于擦除扇区模式下的输入条目写功能，详情请参见 AN4860。条目的结构在 *FlashDrv*.c 中定义：

```
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;
```

“FlashDrv.c”中定义了一个全局条目变量：

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

代码中还定义了一个指向该条目的指针：

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

从用户的角度来看，有 2 种方式可以访问条目中用户定义的数据。例如：

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

表 22. MC56F82xxx 系列的 EEPROM 仿真驱动程序列表

函数名称	简短说明
Crc_Init()	使能 CRC 模块的时钟。内联函数。
EepromDrv_Init()	FlashDrv_Init()的别名。
Crc_Calculation()	计算字节串的 CRC-16 代码
EepromDrv_Write()	向 EEPROM 中写入一个条目。计算 CRC-16 代码并作为条目的一部分写入 EEPROM。
EepromDrv_Read()	从 EEPROM 读取旧的条目并存储到 FLASHDRV_IWRT_ENTRY.entry 中。进行 Crc-16 校验。

2.2.1 Crc_Init()

这是一个内联函数，可以使能 MC56F82xxx 系列中 CRC 模块的时钟。

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

CRC 生成器模块采用 16 位 CRC-CCITT 多项式（$x^{16}+x^{12}+x^5+1$）来生成 CRC 代码以便进行错误检测。

2.2.2 Crc_Calculation()

该函数采用 2.2.1 节 中所述的 CRC 模块来计算字节串的 CRC 代码。计算前必须先调用 *Crc_Init()* 。

该函数的原型为:

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

返回的 16 位数据就是 CRC 代码。

表 23. Crc_Calculation()函数参数

参数名称	参数类型	说明
pbData	UWord8*	字节指针，指向需要 CRC 的字节串
w16Cnt	UWord16	数据串长度，以字节为单位

该函数在 *EepromDrv_Write()*和 *EepromDrv_Read()*中调用。可以使用此函数来计算其它数据的 CRC, 因为它是一个通用函数。清单 8 显示了如何使用它。

清单 8. 使用 Crc_Calculation()并计算字节串的 CRC 代码

```
#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()

如 MC56F82xxx 系列的 EEPROM 仿真驱动程序说明 一节中所述，一个称为 *FLASHDRV_IWRT_DATA* 的指针指向结构体变量 *FLASHDRV_IWRT_ENTRY.entry*，该变量由两个部分组成：用户定义的数据变量和一个 16 位的 CRC 代码 *dwCrcSum*。在 MC56F82xxx 系列的 EEPROM 仿真驱动程序说明 给出的例子中，用户定义的变量为包含 200 个字的数组。

在函数 *EepromDrv_Write()*中，首先，基于 *FLASHDRV_IWRT_ENTRY.entry* 中用户定义的变量计算 CRC 代码，并存储在 *FLASHDRV_IWRT_ENTRY.entry.dwCrcSum* 中。接着，将整个 *FLASHDRV_IWRT_ENTRY.entry*（包括 CRC 代码）写入 flash。

该函数的原型为：

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

表 24. EepromDrv_Write()函数返回代码

返回代码	说明
EEPROM_FLASHDRV_SUCCESS	成功将条目写入 flash
EEPROM_FLASHDRV_ACCESS_ERROR	函数内部错误
EEPROM_FLASHDRV_FAIL	FTFL_FSTAT 的 MGSTAT0 位置位，表明验证操作期间发生了错误。
EEPROM_FLASHDRV_PROT_VIOLATION	误写入保护区域

以下是该函数的源代码：

```
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()

该函数会从 flash 读取定义的备份条目并存储到 *FLASHDRV_IWRT_ENTRY.entry* 中，包括 CRC 代码。然后，它会计算 *FLASHDRV_IWRT_ENTRY.entry* 中所有数据的 CRC 代码。如果数据没有损坏，CRC 代码应为 0。

该函数的原型为：

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

表 25. EepromDrv_Read()函数参数

参数名称	参数类型	说明
uw16EntryAge	UWord16	编号表示待读取的条目。0 表示最新的条目。

表 26. EepromDrv_Read()函数返回代码

返回代码	说明
EEPROM_FLASHDRV_SUCCESS	成功读取条目
EEPROM_FLASHDRV_ACCESS_ERROR	参数无效。例如，uw16EntryAge 值太大，无有效条目。
EEPROM_CRC_ERROR	读取条目的 CRC 代码不为 0。

以下是 *EepromDrv_Read()*函数的源代码:

```
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;
        }
    }
}
```

清单 9 显示了如何在 MC56F82748 上使用该驱动程序来模拟 EEPROM。例如，如果用户有 4 个字、3 个长字和 2 个字节需要存储到 EEPROM 中，头文件 *FlashDrv_cfg.h* 中的配置如下所示:

```
/* 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;
```

在该配置下，从 0x7A00 到 0x7FFF 的三个扇区用作 EEPROM 备份。

清单 9. 在 MC56F82748 上使用 EEPROM 仿真驱动程序

```
#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 更新固件而不擦除 EEPROM

在 CodeWarrior10.x 中，可以做到更新 Flash 中的应用代码而不擦除 EEPROM。

- 对于 MC56F84xxx 系列，EEPROM 备份存储在 FlexNVM 中，在程序存储器映像中的地址范围为 0x68000 到 0x6BFFF。避免在编程过程中擦除 flash 的这个部分。
- 对于 MC56F82xxx 系列，EEPROM 备份存储在程序 flash 的顶部几个扇区中。清单 9 第 页上 20 中使用了三个扇区， 在程序存储器映像中的地址范围为 0x7A00 到 0x7FFF。避免在编程过程中擦除这些扇区。

以下介绍了 CodeWarrior 10.6 中限制范围的 flash 编程方法。以 MC56F84789 为例:

1. 从 CodeWarrior IDE 菜单栏中，选择 **Window > Show View > Other**。此时将出现 **Show View** 对话框。
2. 展开 **Debug**，并选择 **Target Tasks**。

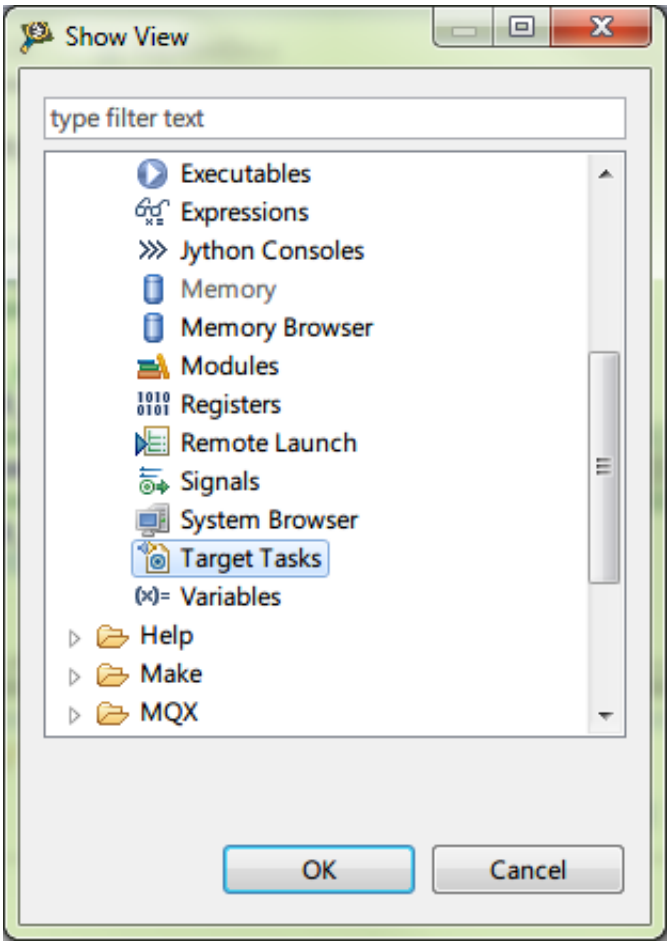

图 3. “Show View”对话框

3. 单击 **OK**。

   此时将出现 **Target Tasks** 视图

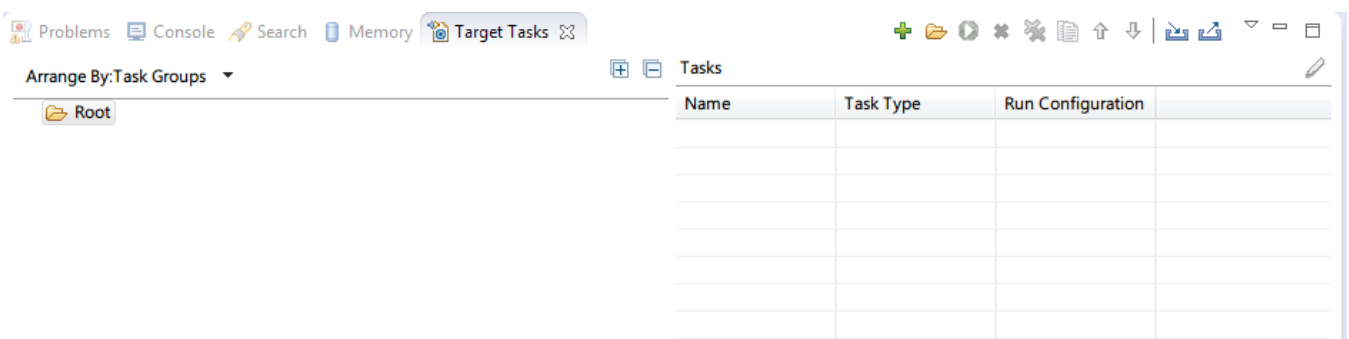

图 4. “Target Tasks”视图

4. 右键单击 **Root** 并从上下文菜单中选择 **Import**。

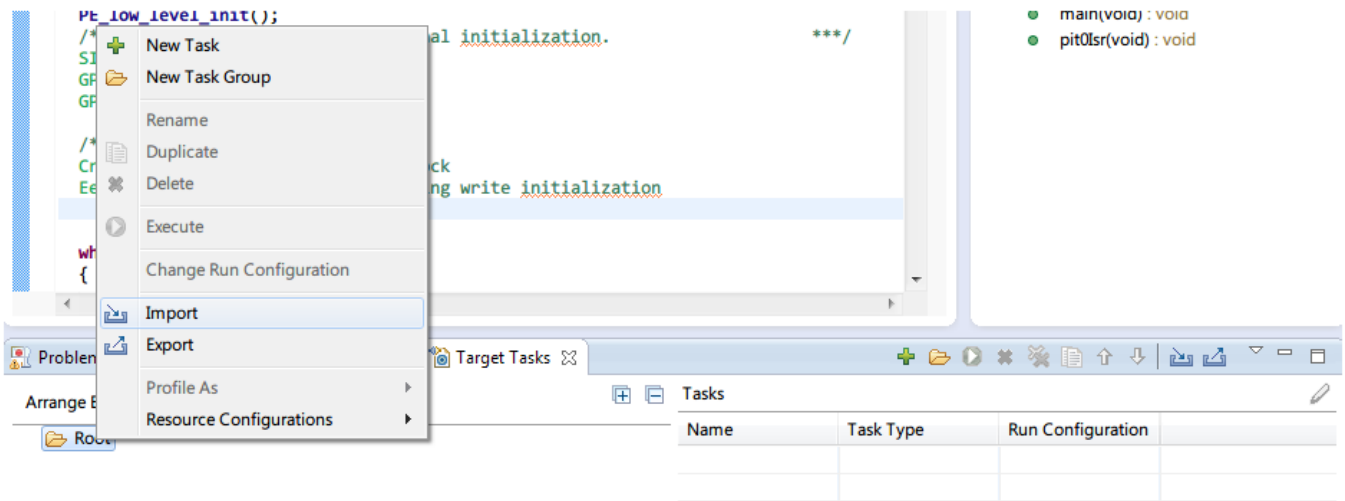

图 5. 上下文菜单

5. 导航到预定义的任务文件夹（`<CW MCU install>\MCU\bin\plugins\support\TargetTask\Flash_Programmer\`），并选择所需的.xml 文件。这个例子中选择了 MC56F84789.xml。

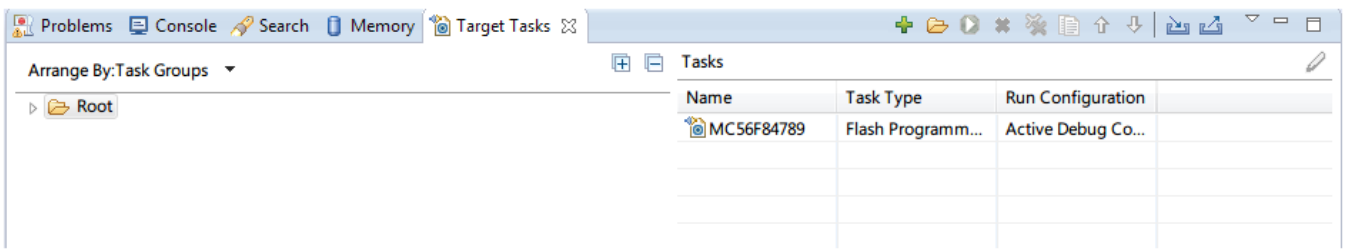

图 6. 选择 MC56F84789.xml 文件

6. 双击任务名称。此时将出现 **DSC Flash Programmer Task** 选项卡。

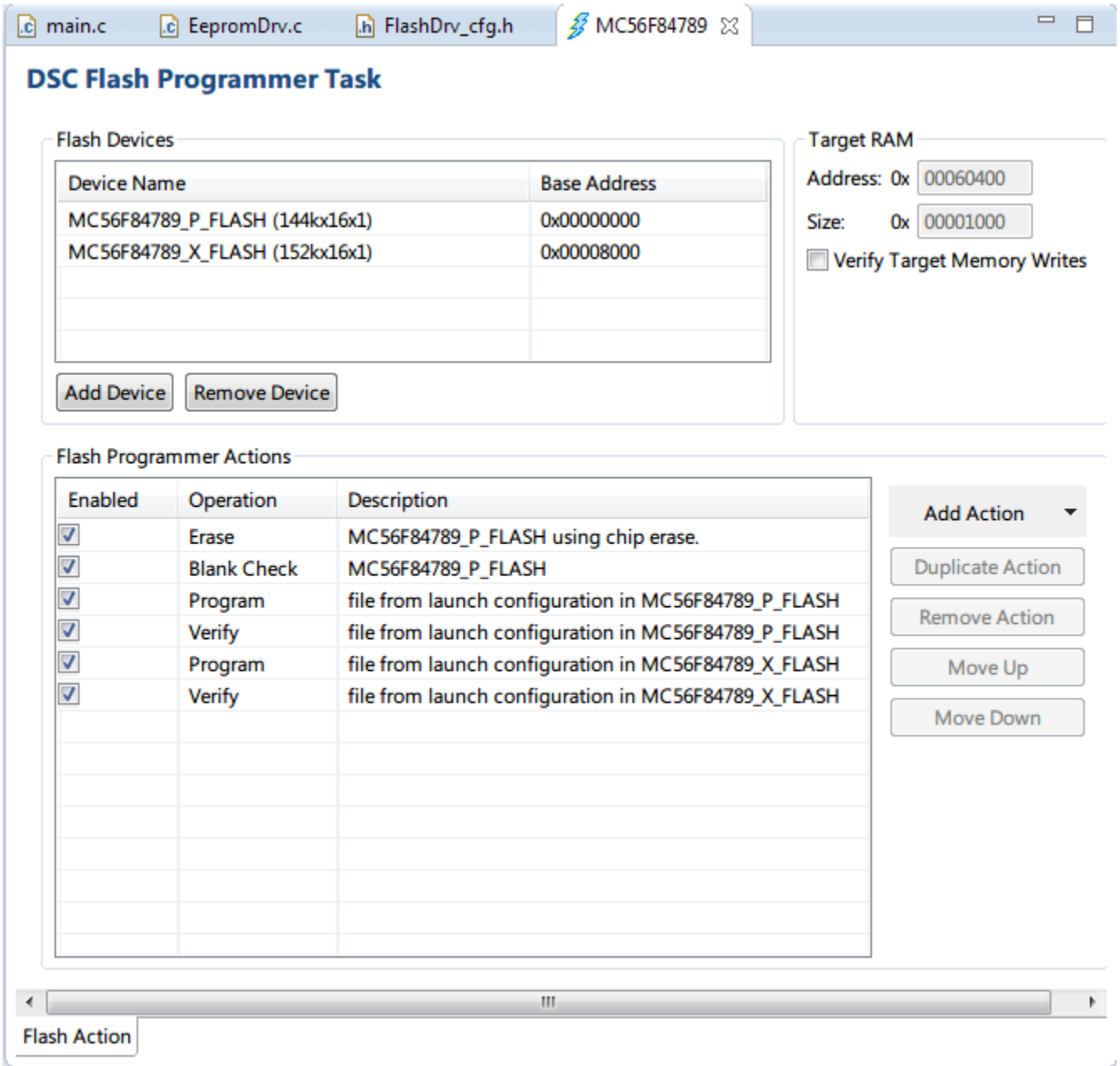

图 7. Flash 编程器任务编辑器窗口，显示已存储的操作

7. 取消选中 **Erase** 和 **Blank Check** 操作。同时，在 MC56F84789_X_FLASH 的启动配置中，取消选择 **Program** 和 **Verify** 操作。

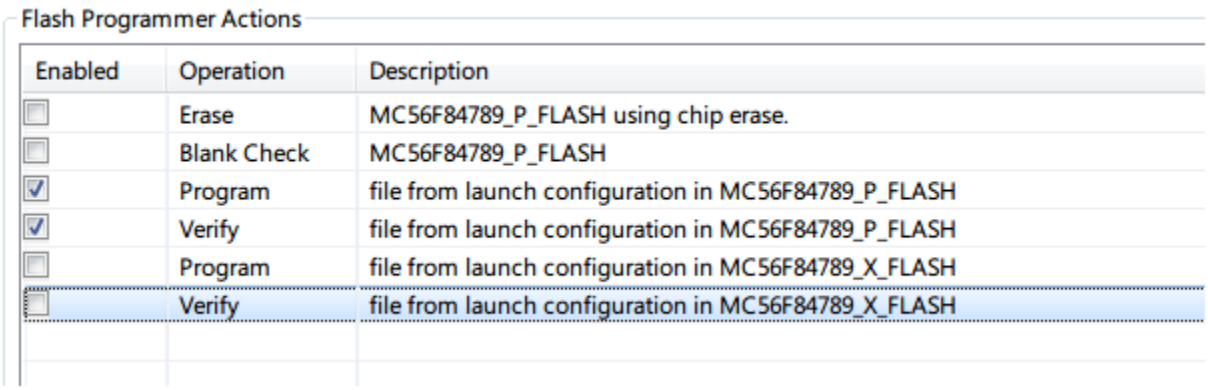

图 8. 只为 MC56F84789_P_FLASH 保留 program 和 verify 操作

8. 双击选中的 **Program** 操作。在弹出对话框中，选中 **Erase sectors before program** 选项。选中 **Restrict to Addresses in this Range** 选项。指定地址范围。此范围外的存储器内容不会改变。单击 **Update Program Action** 按钮，以更新该操作的设置。

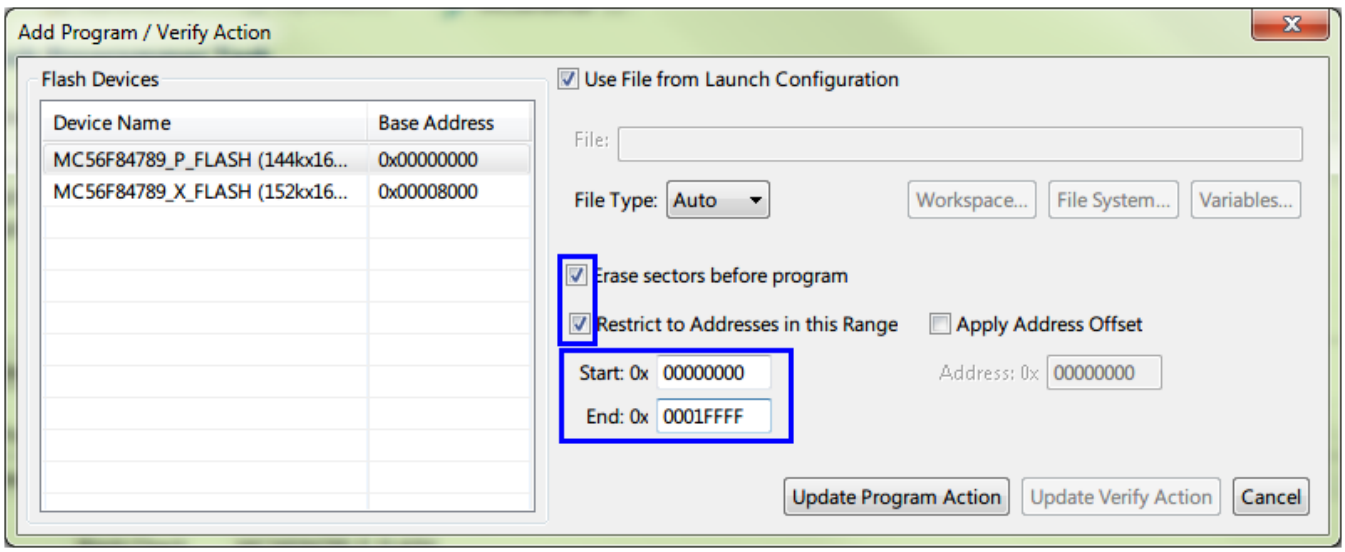

图 9. “Program” 操作的“Add Program/Verify Action”对话框

9. 双击选中的 **Verify** 操作。在弹出对话框中，选中 **Restrict to Addresses in this Range** 选项，并指定与 **Program** 操作同样的地址范围。
10. 单击 **Update Verify Action** 按钮，以更新该操作的设置。

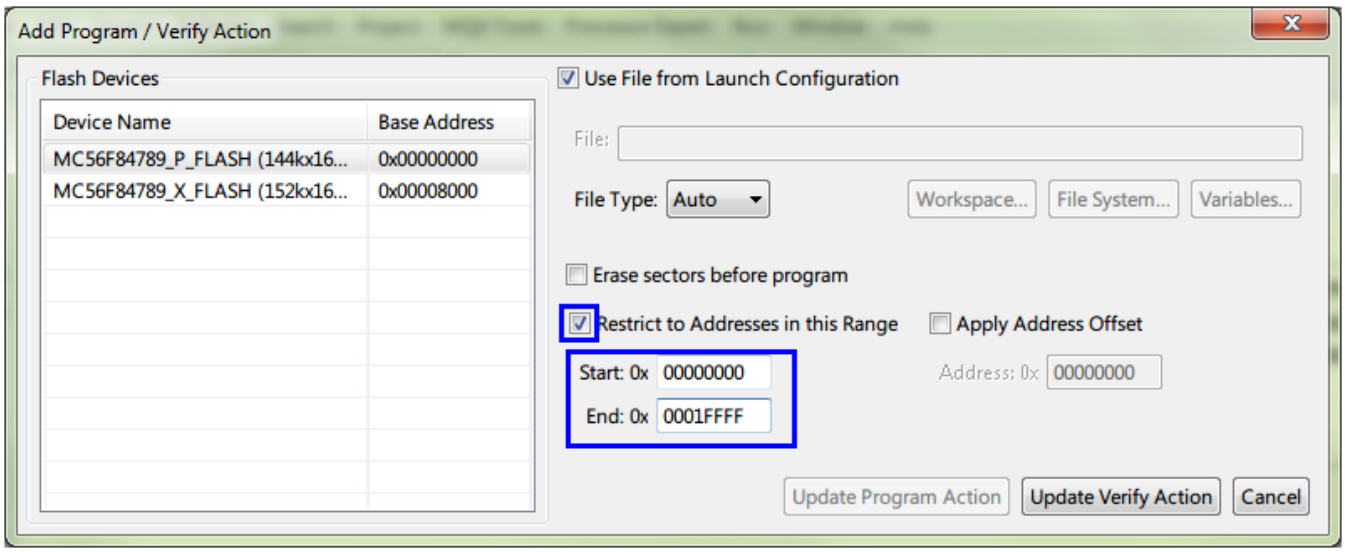

图 10. “Verify”操作的“Add Program/Verify Action”对话框

11. 在 **Target Tasks** 视图中，右键单击任务名称，并选择 **Change Run Configuration**。

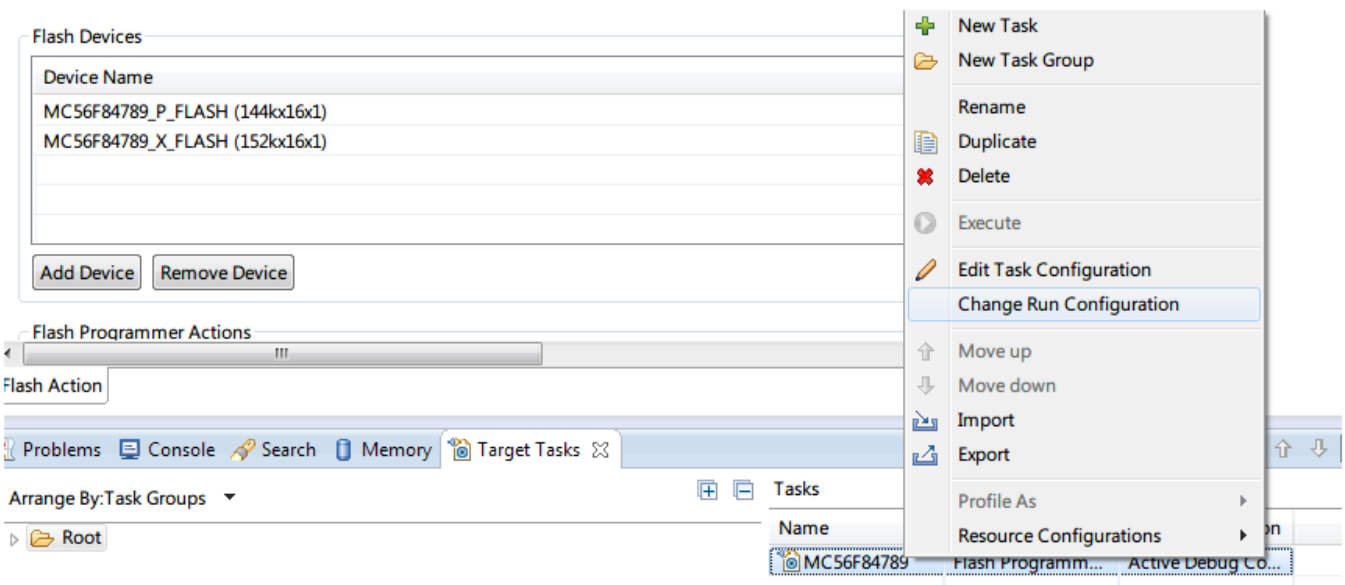

图 11. 更改运行配置

12. 此时将出现 **Run Configuration** 对话框。从已打开项目的可用配置中，选择运行配置。单击 **OK**。

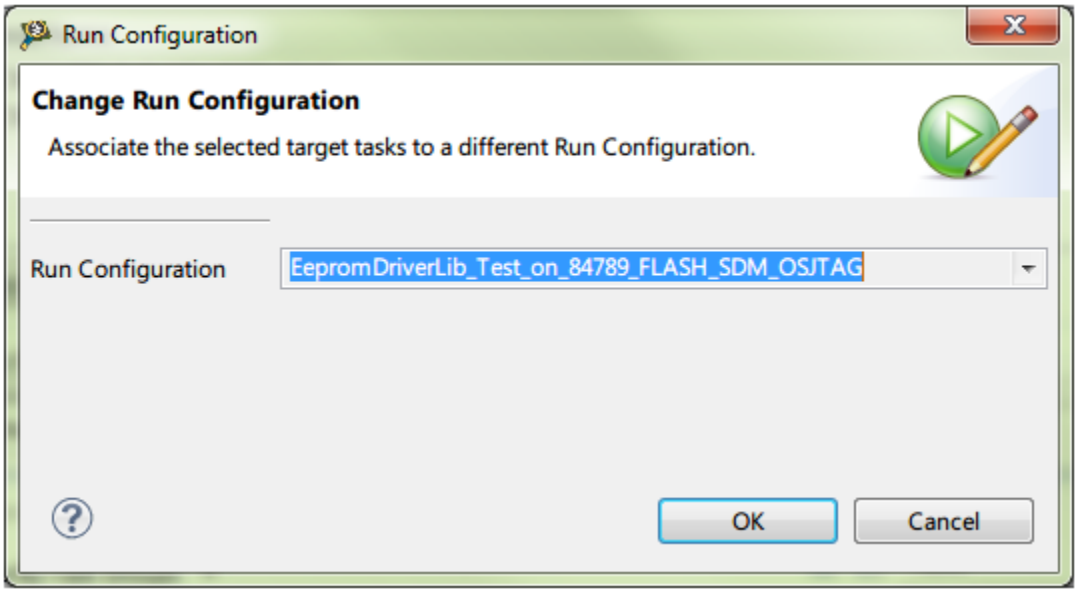

图 12. 运行配置对话框

13. 单击 **Execute** 按钮以执行操作。

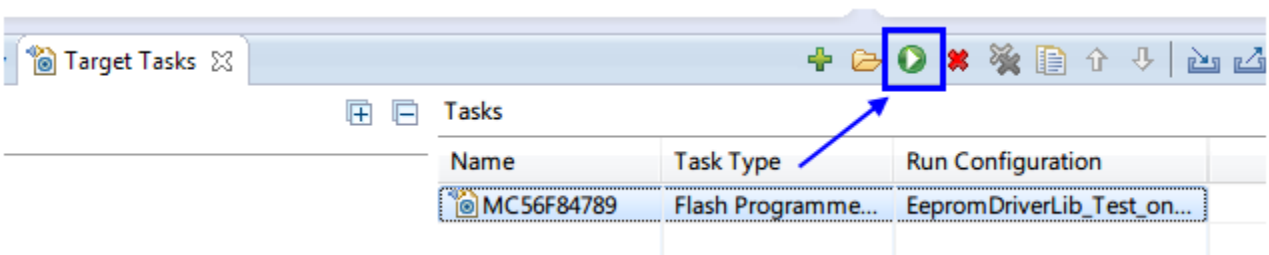

图 13. 执行操作

通过上述步骤 1 至 13，只对程序 Flash 进行了编程，FlexNVM 则保持原样。在这种情况下，EEPROM 不受影响。

对于 MC56F82xxx 系列，由于程序 Flash 的最后几个扇区用作仿真 EEPROM 的备份，因此应相应地更改图 9 和图 10 中的地址范围。在清单 9 第 页上 20，由于地址 0x7A00~0x7FFF 用作 EEPROM 备份，为避免擦除 EEPROM 的内容，图 9 和图 10 中的地址范围应更改为 0x0000~0x79FF。

4 结论

EEPROM 驱动程序中有三个文件: *EepromDrv.c*、*EepromDrv.h* 和 *EepromDrv_cfg.h*。*EepromDrv_cfg.h* 中只有一个宏，用于定义驱动程序是针对 MC56F84xxx 系列还是 MC56F82xxx 系列。

- 在 *EepromDrv_cfg.h* 文件中将 EEPROM_EMULATION 设为 0 时，则针对 MC56F84xxx 系列使能驱动程序。MC56F84xxx 系列的 EEPROM 驱动程序说明 显示了驱动程序的配置。
- 在 *EepromDrv_cfg.h* 文件中将 EEPROM_EMULATION 设为 1 时，则针对 MC56F82xxx 系列使能驱动程序。同时，如图 2 所示，AN4860 中描述的 FDL 驱动程序也应包括在该项目中。FDL 的配置在 MC56F82xxx 系列的 EEPROM 仿真驱动程序说明 中有描述。请注意按 AN4860 中所述修改链接器文件。

对于 MC56F84xxx 系列，一个内置式文件归档系统会自动执行 EEPROM 特性，因此性能更为强大可靠。MC56F82xxx 系列则没有这样的系统，因此利用 FDL 来模拟 EEPROM，连同 CRC-16 函数和递增写特性，改善了稳定性和 flash 擦写耐受能力。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5074
Revision 0, 01/2015