

M16C/62P Group

Virtual EEPROM management

Introduction

This application note describes the procedure for a virtual EEPROM management in the CPU rewrite mode EW1 and shows some software examples, how to increase the erase/write cycle for Block A and Block 1.

The M16C/62P is a 16-bit MCU with up to 384 Kbytes (512K bytes under planning). By activating the CPU rewrite mode, it is possible to enable an additional 4K byte flash block, also named the virtual EEPROM Block A. It makes sense to use this area as extended memory space for further data variables like e.g. calibration or parameter data. This block can be enabled and disabled via software and preserve important data, which are not cleared after a MCU reset.

It is possible to erase and program the on-chip flash memory without any external programming devices. For this, two different CPU rewrite modes are available: The erase/write mode 0 (EW0) and the mode 1 (EW1). Compared to the EW0 mode, where the flash operation is executed in the e.g. RAM area, the EW1 mode executed its software in the flash memory. In this document an example for the EW1 mode is shown.

Besides the Block A, the M16C62P consist of a second 4Kbyte flash block (Block 1), which is also specified for 10.000 E/W. It is possible to increase the erase/ write cycles – for both blocks - via an expedient software function. This application note also gives you an idea how to realize such a flash memory management.

Target Device

Applicable MCU: M16C62P Group (such as M30626FHPFP, M30626FHGP, M30627FHGP)

Oscillation frequency: 16 MHz (internal 24MHz)

Tools: The software project supports IAR V2.11 compiler.

For debugging the Renesas debugger PD30F or KD30 can be used.

Contents

1. Flash Memory	3
1.1 Flash Memory Map.....	3
1.2 Virtual EEPROM blocks	4
2. CPU Rewrite Modes	5
2.1 Important information to flash data.....	6
2.2 Some important registers	6
2.2.1 Processor Mode Register (PM1)	6
2.2.2 Flash Memory Control Register 0 (FMR0).....	6
2.2.3 Flash Memory Control Register 1 (FMR 1).....	8
2.3 Command for Flash rewrite mode	8
2.4 The Virtual EEPROM management Software	9
2.5 EW1 Mode.....	10
2.6 Implementing the Virtual EEPROM management software example	10
2.6.1 Software project available.....	12
3. Software Code for virtual EEPROM management	13
3.1 Header file VEEProm.h	13
3.2 Program file VEEProm.c	16
3.3 Program file Main.c	27
Reference	29
Revision Record	29

1. Flash Memory

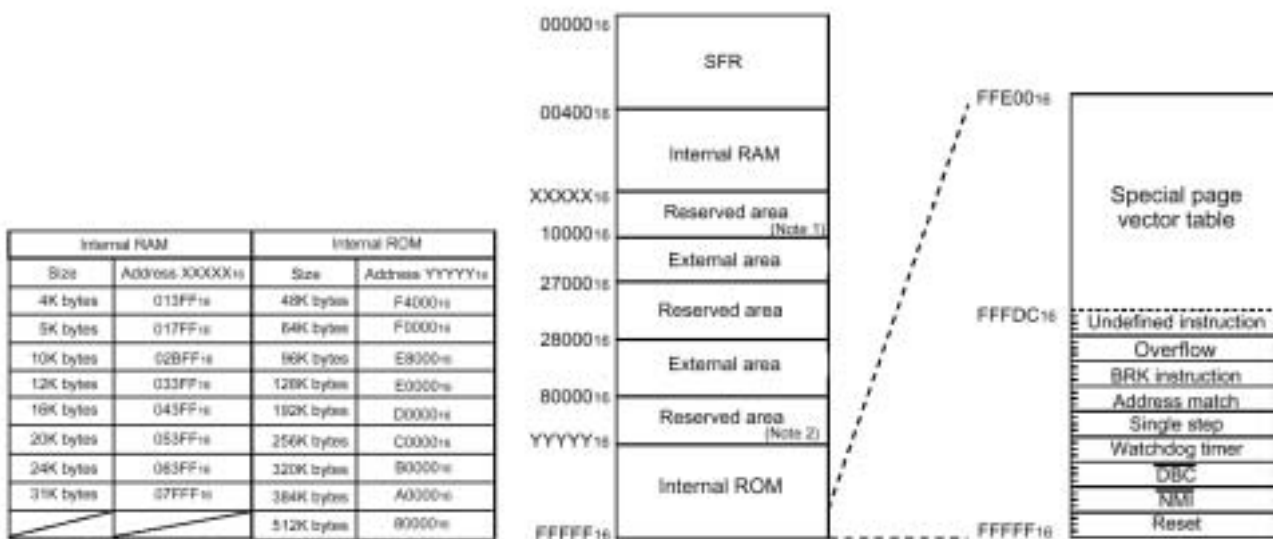
Figure 1 shows a memory map about the M16C/62P group. The address space extends the 1M bytes from address 00000hex to FFFFF hex. The internal Flash is allocated in a lower address area, beginning with address 0xFFFFF.

The fixed interrupt vector table is allocated to the addresses from 0xFFFFDC to 0xFFFFF. The start address of each interrupt routine has to be stored here.

The internal RAM is allocated in an upper address direction beginning with address 0x00400. For example, a 10-Kbytes internal RAM is allocated to the addresses from 0x0040016 to 0x02BFF. In addition to storing data, the internal RAM also stores the stack used when calling subroutines and when interrupts are generated.

The special function registers (SFR) are allocated to the addresses from 0x00000 to 0x003FF. The peripheral function control registers are located here.

The special page vector table is allocated to the addresses from FFE0016 to FFFDB16. This vector is used by the JMPS or JSRS instruction. For more details, refer to the “M16C62P, M16C/60 and M16C/20 Series Software Manual.” In memory expansion and microprocessor modes, some areas are reserved for future use and cannot be used by users.

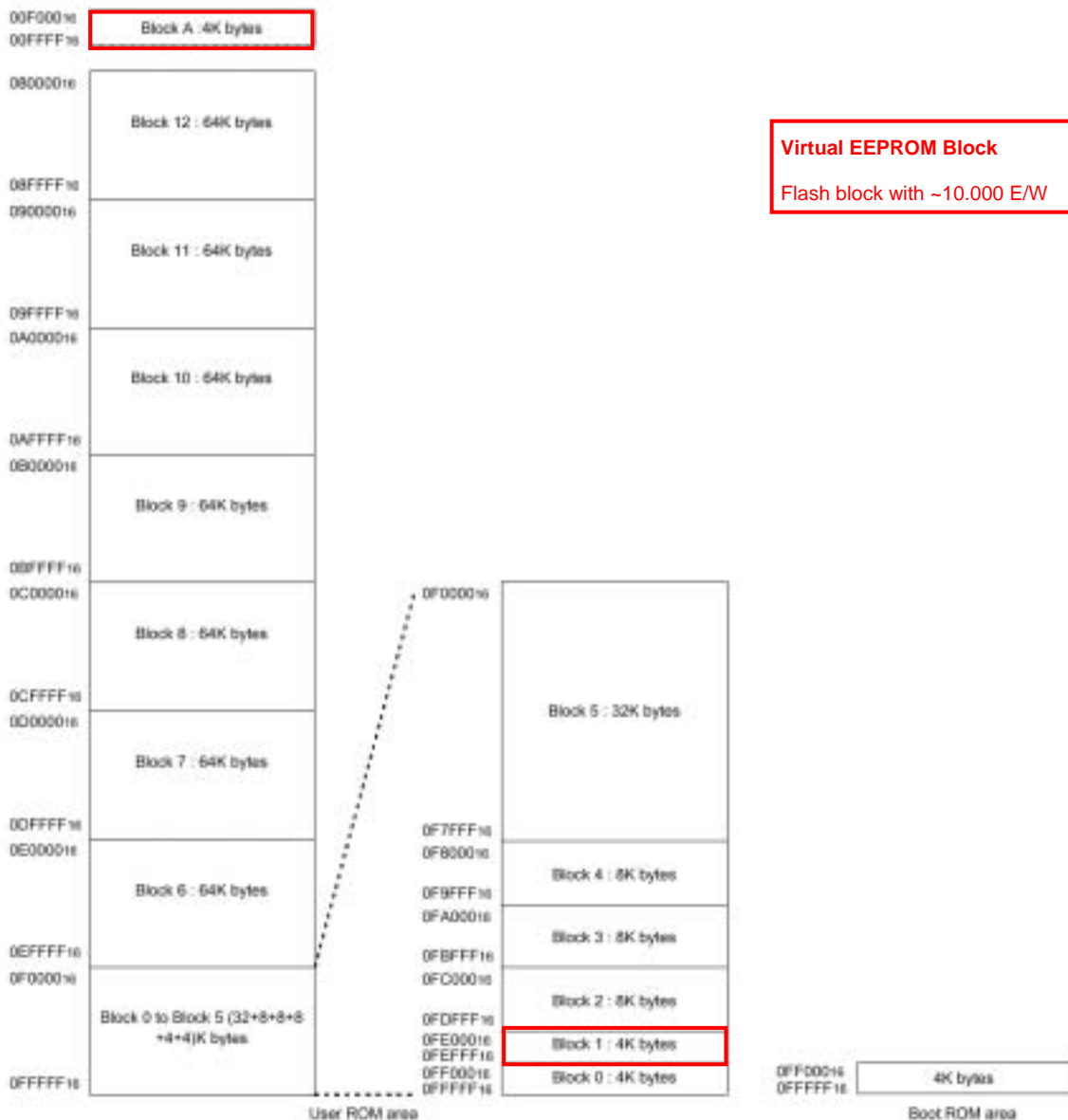


Note 1: During memory expansion and microprocessor modes, can not be used.
 Note 2: In memory expansion mode, can not be used.
 Note 3: Shown here is a memory map for the case where the PM10 bit in the PM1 register is "1" and the PM13 bit in the PM1 register is "1".

Figure 1 Flash Memory

1.1 Flash Memory Map

The M16C62P flash memory is separated between a user ROM area and a boot ROM area. Below Figure 2 shows the block diagram for the flash memory. The user Flash has an additional 4K-byte flash block A. The user flash area is split into several blocks. Each block can individually be protected (locked) against programming or erasure. The user flash area can be rewritten in different CPU modes, like the standard serial input/output and parallel input/output mode. Enable Block A by setting the PM1 register's PM10 bit to "1" (block A enabled, CS2 area at addresses 1000016 to 26FFF16). The boot ROM area is located at addresses that overlap the user ROM area and can only be rewritten in parallel input/output mode. To start the program in the boot ROM area reset the hardware and set the CNVSS pin 50 too high.



Note 1: The boot ROM area can only be rewritten in parallel input/output mode.
 Note 2: To specify a block, use an even address in that block.
 Note 3: Shown here is a block diagram during single-chip mode.
 Note 4: Block A can be made usable by setting the PM1 register's PM10 bit to "1" (block A enabled, CS2 area allocated at addresses 10000h to 20FFFh).
 Block A cannot be erased by the Erase All Unlocked Block command. Use the Block Erase command to erase it.

Figure 2 Flash Memory Map

1.2 Virtual EEPROM blocks

The flash Block A and Block 1 are the related blocks that are intended for the usage as virtual EEPROM. Both blocks have a size of 4k-bytes. Below table show the address allocation and their memory size.

Table 1 Block Addresses of Flash Block A and Flash Block 1

	Size	Address	highest even Block Address
Block A	4 k bytes	F000h – FFFFh	FFFEh
Block 1	4 k bytes	FE000h – FEFFFh	FEFFEh

Block A and Block 1 guarantee minimum programming and erase cycles of 10,000 times. All other blocks can be programmed and erased 1,000 times. All flash blocks can only be programmed in units of 1 word (2 bytes), but it is possible to read-out one or two bytes. To erase one data byte you have to erase the complete flash block.

2. CPU Rewrite Modes

In the CPU rewrite mode it is possible to rewrite the different Flash blocks area by executing the software commands from the CPU. Therefore, the user ROM area can be rewritten directly, while the microcomputer is mounted on-board without an external flash programmer. During CPU rewrite mode, the user ROM area can operate in Erase Write 0 (EW0) mode or Erase Write 1 (EW1) mode. Table 2 describes the differences between EW0 and EW1 mode.

Table 2 CPU Rewrite Modes EW0 and EW1

Item	EW0	EW1
<ul style="list-style-type: none"> Operation Mode 	<ul style="list-style-type: none"> Single chip mode Memory expansion mode Boot mode 	<ul style="list-style-type: none"> Single chip mode
<ul style="list-style-type: none"> Areas in which a rewrite control program can be located 	<ul style="list-style-type: none"> User Flash Areas Boot Flash area 	<ul style="list-style-type: none"> User Flash area
<ul style="list-style-type: none"> Areas in which a rewrite control program can be executed 	<ul style="list-style-type: none"> Must be transferred to any other area than the flash memory (e.g. RAM) before being executed (Note 2) 	<ul style="list-style-type: none"> Can be executed directly in the user ROM area
<ul style="list-style-type: none"> Areas which can be rewritten 	<ul style="list-style-type: none"> User ROM area 	<ul style="list-style-type: none"> User ROM area - However, this does not include the area in which a rewrite control program exists.
<ul style="list-style-type: none"> Software command limitation 	<ul style="list-style-type: none"> none 	<ul style="list-style-type: none"> Program, Block erase command cannot be executed on any block in which a rewrite control program exists Erase all unlocked Block command Cannot be executed when the lock bit for any block in which a rewrite control program exists is set to "1" (unlocked) or the FMR0 register's FMR02 bit is set to "1" (lock bit disabled) Read Status Register command cannot be executed
<ul style="list-style-type: none"> Modes after program or erase 	<ul style="list-style-type: none"> Read Status Register mode 	<ul style="list-style-type: none"> Read Array mode
<ul style="list-style-type: none"> CPU status during Auto Write and Auto Erase 	<ul style="list-style-type: none"> Operating 	<ul style="list-style-type: none"> Hold state (I/O ports retain the state in which they were before the command was executed)(Note 1)
<ul style="list-style-type: none"> Flash memory status detection 	<ul style="list-style-type: none"> Read the FMR0 registers FMR00, FMR06 and FMR07 bits in a program Execute the Read Status Register command to read the status register's SR7, SR5, and SR4 flags. 	<ul style="list-style-type: none"> Read the FMR0 registers FMR00, FMR06 and FMR07 bits in a program

Note 1: Make sure no interrupts (except NMI and watchdog timer interrupts) and DMA transfers will occur.

Note 2: When in CPU rewrite mode, the PM10 and PM13 bits in the PM1 register are set to "1". The rewrite control program can only be executed in the internal RAM or in an external area that is enabled for use when the PM13 bit = 1. When the PM13 bit = 0 and the flash memory is used in 4M-byte mode, the extended accessible area (40000₁₆ to BFFFF₁₆) cannot be used.

2.1 Important information to flash data

- Before entering into CPU rewrite mode (EW0 or EW1 mode), select 10 MHz or less as CPU clock.
- Take note about the CM06 bit (CM0 register) and the CM17 to CM16 bits (CM1 register). Also, set the wait state PM17 bit (PM1 register) to "1" .
- Make sure that any interrupt which has a vector in the variable vector table or address match interrupt will not be accepted during the auto program or auto erase period. Avoid using watchdog timer interrupts.
- The NMI interrupt can be used because the FMR0 register and FMR1 reg. are initialized when this interrupt occurs.
- To set the FMR01, FMR02 or FMR11 bit to "1 ", write "0 " and then "1 " in succession. This is necessary to ensure that no interrupts or DMA transfers will occur before writing "1 " after writing "0 ". Also only when NMI pin is "H " level
- Avoid rewriting any block in which the rewrite control program is stored.
- Write the command code and data at even addresses.
- To erase flash block take highest even address

2.2 Some important registers

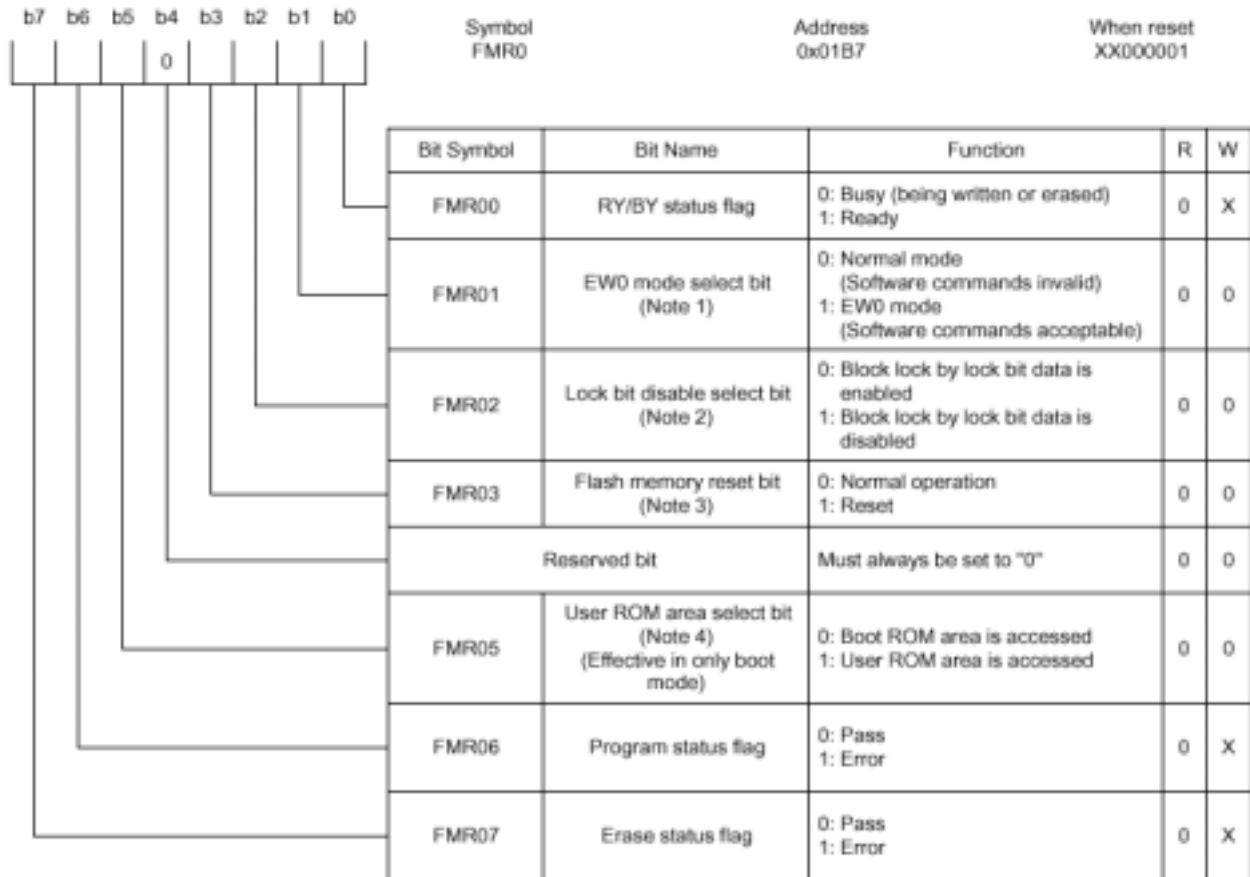
2.2.1 Processor Mode Register (PM1)

2.2.2 Flash Memory Control Register 0 (FMR0)

Bit	Symbol	Address	After reset	Function	RW
b7					
b6					
b5					
b4					
b3					
b2					
b1					
b0					
	PM1	0005 ₁₆	0X001000 ₂		
				set PM10 bit to enable flash BLOCK A -> virtual EEPROM	
	PM10			CS2 area switch bit (data block enable bit) (Note 2) 0: 08000 ₁₆ to 26FFF ₁₆ (block A disable) 1: 10000 ₁₆ to 26FFF ₁₆ (block A enable)	RW
	PM11			0 : Address output 1 : Port function	RW
	PM12			0 : Watchdog timer interrupt 1 : Watchdog timer reset (Note 4)	RW
	PM13			Internal reserved area expansion bit (Note 6)	RW
	PM14			b5 b4 0 0 : 1 Mbyte mode (Do not expand) 0 1 : Must not be set 1 0 : Must not be set 1 1 : 4 Mbyte mode	RW
	PM15				RW
	(b6)			Reserved bit	RW
	PM17			0 : No wait state 1 : With wait state (1 wait)	RW

Note 1: Write to this register after setting the PRC1 bit in the PRCR register to "1" (write enable).

Note 2: For the mask ROM version, this bit must be set to "0" . For the flash memory version, the PM10 bit also controls block A by enabling or disabling it. However, the PM10 bit is automatically set to "1" when the FMR0 1 bit in the FMR0 register is "1" (CPU rewrite mode).



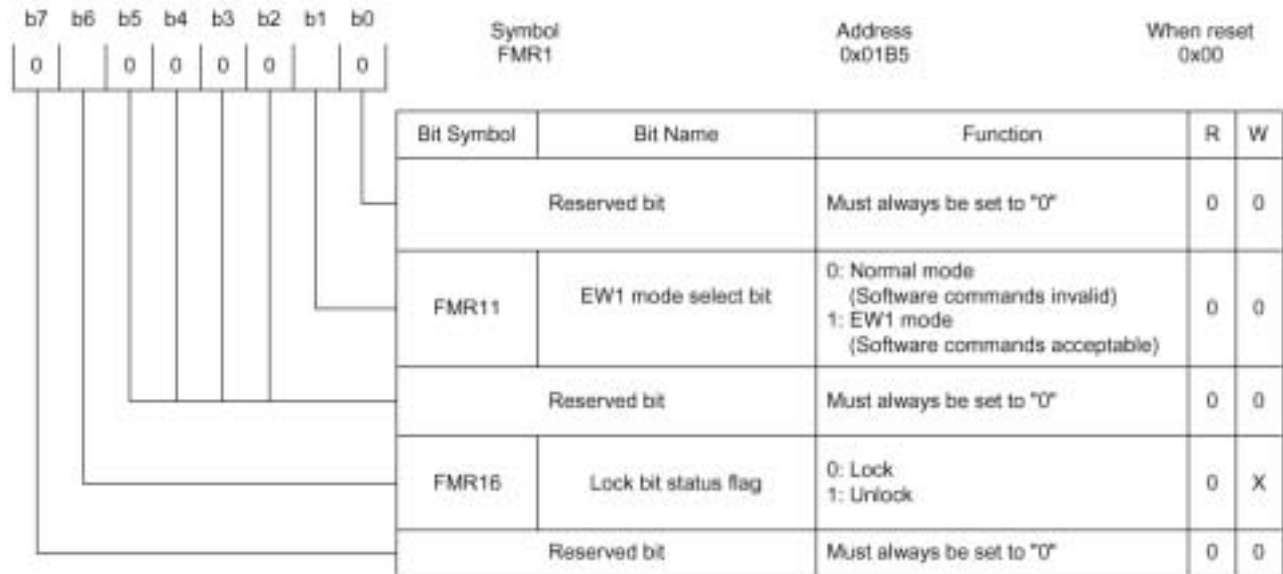
Note 1: For this bit to be set to "1", the user needs to write a "0" and then a "1" to it in succession. If this procedure is not followed the bit will not be set. The reason for this is that it is necessary to ensure that no interrupt or DMA transfer is executed during the interval. Write to this bit only when executing out of an area other than the internal flash memory and when the NMI pin is "H" level. Clear this bit to "0" after a read array command or flash reset.

Note 2: For this bit to be set to "1" the user needs to write a "0" and then a "1" to it in succession when the CPU rewrite mode select bit = "1". If this procedure is not followed the bit will not be set. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval.

Note 3: Effective only when the CPU rewrite mode select bit = 1. After write "1", write "0" when RY/BY status flag is "1".

Note 4: Write to this bit only when executing out of an area other than the internal flash memory.

2.2.3 Flash Memory Control Register 1 (FMR1)



Command	First Bus Cycle			Second bus cycle		
	Mode	Address	Data (D0 to D7)	Mode	Address	Data (D0 to D7)
Read array	Write	X	FF			
Read status register	Write	X	70	Read	X	SRD (Note 2)
Clear status register	Write	X	50			
Program (Note 3)	Write	WA	40	Write	WA (Note 3)	WD (Note 3)
Block erase	Write	X	20	Write	BA (Note 4)	D0
Erase all unlock block	Write	X	A7	Write	X	D0
Lock bit program	Write	BA	77	Write	BA	D0
Read lock bit status	Write	X	71	Write	BA	D0 (Note 5)

Note 1: When a software command is input, the high-order byte of data (D8 to D15) is ignored.

Note 2: SRD = Status Register Data (Set an address to even address in the user ROM area)

Note 3: WA = Write Address (even address), WD = Write Data (16-bit data)

Note 4: BA = Block Address (Enter the maximum even address of each block.)

Note 5: Please note that the second bus cycle is different from that of conventional M16C/62 group MCU's. Read the bit 6 of the flash memory control register 1 (FMR16) for lock bit status. "0": lock, "1": unlock.

Note 6: X denotes a given address in the user ROM area (that is an even address).

2.3 Command for Flash rewrite mode

2.4 The Virtual EEPROM management Software

The virtual EEPROM management software uses the EW1 mode, to program and to erase data in the flash BLOCK_A and BLOCK_1. Like mentioned in chapter “1.2 Virtual EEPROM block” it is not possible to erase only one, two and more bytes, therefore the complete flash block has to be cleared.

For example, the flash BLOCK_A stores an array about 245 bytes parameter data (e.g. calibration value, alarm level, etc.) in the 4K-byte block. The remaining not used bytes in the block are given away. If now only one byte has to be changed, the complete 4k flash block (e.g. BLOCK_A) must be erased first, and then can be rewritten with the new value on the same address. So always the same flash address is used. Note that it is possible to E/W every address 10.000 times.

Solution: The virtual EEPROM management software divides the virtual flash BLOCK_A or BLOCK_1 into different UNITS. In the below example each UNIT has got 256 bytes. By dividing 4096/256, a total number of 16 UNITS is available. Now store in the first time the 256 byte parameter array into UNIT 0. At the second time – e.g. changes of parameter values - store these array into UNIT 1 and so on.

Advantages: On one hand it is now possible to use the total flash block – with even distribution for every flash address - and on the other hand it increase the E/W cycle. For the above example the E/W cycle increase from 10.000 cycle to 16.000 cycle. Figure 3 shows this functionality.

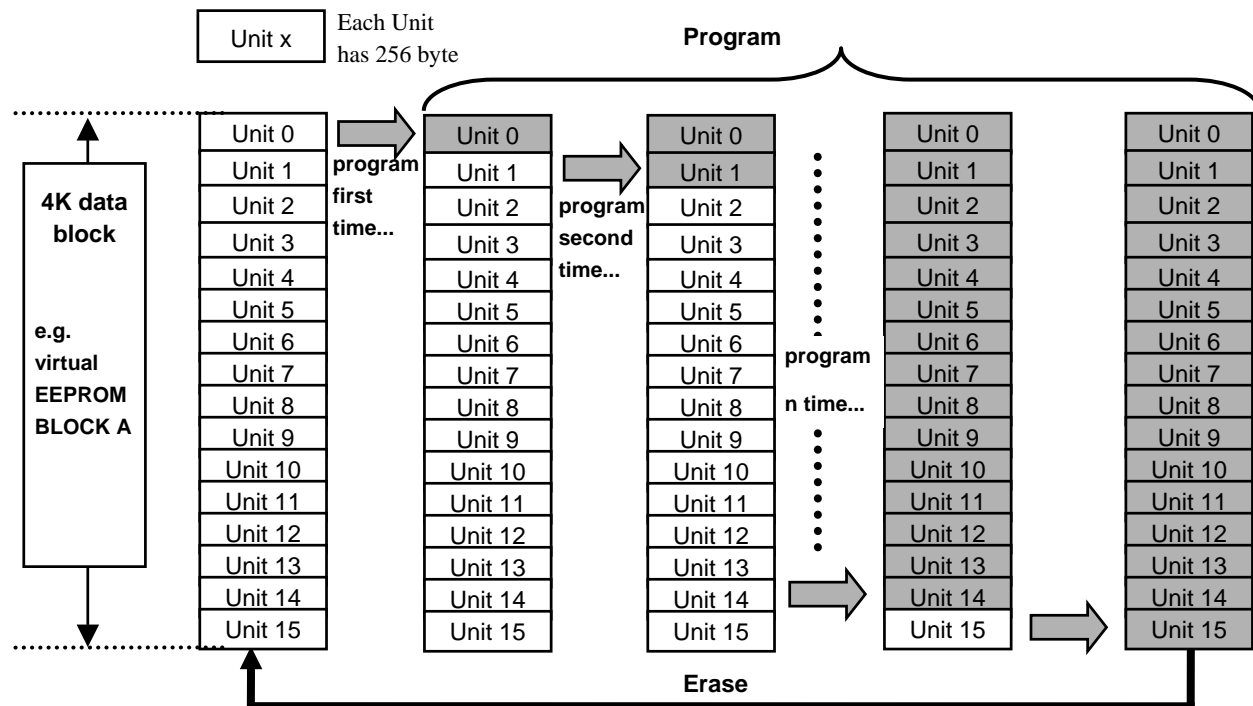
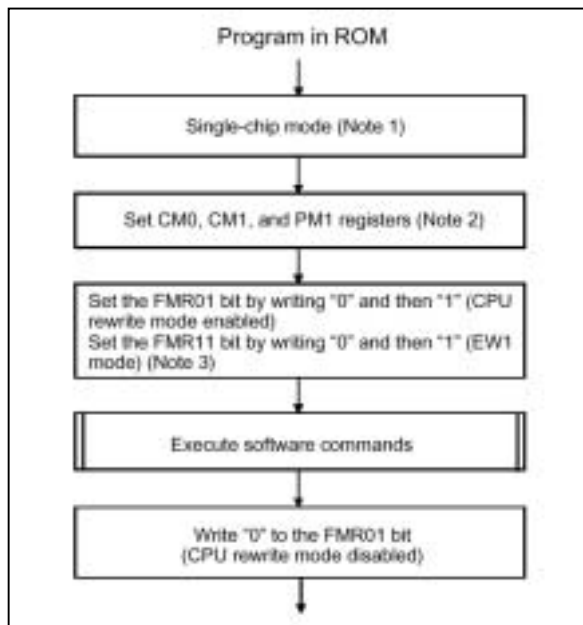


Figure 3 Method of virtual EEPROM Management

2.5 EW1 Mode

The EW1 mode is selected by setting FMR11 bit to “1 ” (by writing “0 ” and then “1 ” in succession) after setting the FMR01 bit to “1 ” (by writing “0 ” and then “1 ” in succession). Read the FMR0 register to check the status of program or erase operation at completion. The status register cannot be read during EW1 mode.



2.6 Implementing the Virtual EEPROM management software example

Include “VEEprom.h” in any project file that requires the virtual EEPROM management functions and add the “VEEprom.c” file to your project. The “VEEprom.c” file contains function examples about how to erase and write data to a flash/virtual block. Also, a software example for a virtual EEPROM management like described in chapter 2.4 is included. In the “VEEprom.h” header file, the virtual EEPROM software has to be configured. Important values like type (unsigned char, int,..) and array length (e.g. array[length]) have to be defined in this file. Resultant the UNIT quantity is calculated automatically.

main.c:

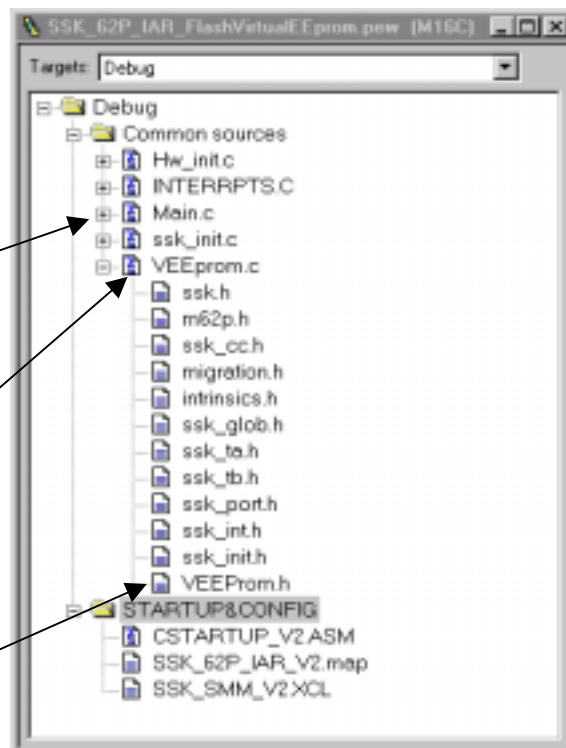
- demo loop
- with RUN led via interrupt
- activate Flash virtual EEPROM via switch

VEEProm.c example:

- write data to flash
- erase flash block
- virtual EEPROM management
- ...

VEEProm.h

Setup virtual EEPROM blocks



- Filename: VEEPROM.C

functions	Description
void <i>demo_writeflashVEE(</i> void <i>)</i>	example how to realize a VIRUTAL EEPROM management
void <i>demo_readflashVEE(</i> void <i>)</i>	example to read flash block
void <i>demo_errorFlashVEE(</i> void <i>)</i>	example for flash error handling
unsigned char <i>copy_array2VEE(</i> UNITTYPE *array, unsigned long VEE_block <i>)</i>	<ul style="list-style-type: none"> ➤ copy parameter array to Virtual EEPROM • return value: outcome of write operation; 0 successful; all other not successful (*1) • parameters: <i>*array</i> -> start pointer of the array, that has to be copied <i>VEEblock</i> -> target flash block
unsigned char <i>write_data2VEE(</i> UNITTYPE data, unsigned long VEEadr <i>)</i>	<ul style="list-style-type: none"> ➤ write one data word to the Virtual EEPROM • return value: outcome of write operation; 0 successful; all other not successful (*1) • parameters: <i>data</i> -> contain data value to store into flash <i>VEEadr</i> -> address of the virtual flash to write to
void <i>copy_VEE2array(</i> UNITTYPE *array, unsigned long VEE_block <i>)</i>	<ul style="list-style-type: none"> ➤ copy array from Virtual EEPROM to data array (in RAM) • return value: no • parameters: <i>*array</i> -> start pointer of target address , where flash data has to be copy to <i>VEEblock</i> -> source flash block, where data is coming from
UNITTYPE <i>read_VEE2data(</i> unsigned long VEEadr <i>)</i>	<ul style="list-style-type: none"> ➤ read one Virtual EEPROM data and store it to RAM array • return value: outcome data from virtual flash • parameters: <i>VEEadr</i> -> source flash block, where data is coming fro
unsigned char <i>erase_VEE(</i> unsigned long block <i>)</i>	<ul style="list-style-type: none"> ➤ erase total Virtual Eeprom block • return value: outcome of write operation; 0 successful; all other not successful (*1) • parameters: <i>block</i> -> enter even highest address of the flash block
unsigned char <i>ReadLockBit(</i> unsigned long block <i>)</i>	<ul style="list-style-type: none"> ➤ check if flash block is locked • return value: lock bit on or off • parameters: <i>block</i> -> enter the flash block address
unsigned char <i>ClrLockBit(</i> unsigned long block <i>)</i>	<ul style="list-style-type: none"> ➤ clear the lock bit in flash block • return value: lock bit on or off • parameters: <i>block</i> -> enter the flash block
void <i>init_VEE_EW1_mode(</i> void <i>)</i>	➤ setup erase or write VEEprom – to EW1 mode
unsigned char <i>init_first_VEE(</i> unsigned long block <i>)</i>	<ul style="list-style-type: none"> ➤ example to init VEE for fist time • return value: outcome of write operation; 0 successful; all other not successful (*1) • parameters:

	<i>block</i> -> enter the first flash block address
void <i>init_MCU_2FlashMode</i> (void)	➤ set the MCU mode to flash programming
void <i>RestoreMCU</i> (void)	➤ Restore MCU mode before flash mode back to original speed and restores I flag back

(*1) outcome of write operation; 0 successful; all other not successful:

return value "0" => flash write/erase successful; no flash error

return value bit 0 = "1" => flash erase error

return value bit 1 = "1" => flash write error

return value bit 2 = "1" => try to erase a non virtual EEPROM block (block A, block 1)

- **Filename: VEEPROM.h**

functions	Description
#define UNITTYPE unsigned int	to define unique type for all data in the virtual EEPROM
#define ONE_UNITSIZE 256	set space for one UNIT size needed memory size[byte] => UNITTYPE * ONE_UNITSIZE // example: 512 byte => int * 256
#define MAX_VIRTUAL_EEPROM_SIZE 4096	VIRTUAL EEPROM size for BLOCK_A, BLOCK_1
#define MAX_UNITS	Formula: MAX_VIRTUAL_EEPROM_SIZE/(ONE_UNITSIZE * sizeof(UNITTYPE))

2.6.1 Software project available

In Chapter "3. Software code for virtual EEPROM Management" the software code can easily be copied and added into your project. Also, Renesas Technologie Europe offers you a demo project - **REB05B0015-0100Z.zip** – for the M16C62P.

3. Software Code for virtual EEPROM management

3.1 Header file VEEProm.h

```

/* ***** */
/*  DISCLAIMER: */
/*  We (RENESAS TECHNOLOGY EUROPE) do not warrant that the Software is */
/*  free from claims by a third party of copyright, patent, trademark, */
/*  trade secret or any other intellectual property infringement. */
/*  Under no circumstances are we liable for any of the following: */
/*  1. third-party claims against you for losses or damages; */
/*  2. loss of, or damage to, your records or data; or */
/*  3. economic consequential damages (including lost profits or */
/*     savings) or incidental damages, even if we are informed of */
/*     their possibility. */
/*  We do not warrant uninterrupted or error free operation of the */
/*  Software. We have no obligation to provide service, defect */
/*  correction, or any maintenance for the Software. We have no */
/*  obligation to supply any Software updates or enhancements to you */
/*  even if such are or later become available. */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS. */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A */
/*  PARTICULAR PURPOSE. */
/* ----- */
/*  Copyright (c) 2003 RENESAS TECHNOLOGY EUROPE All rights reserved. */
/* ----- */
/*  Project Name: SSK M16C62P Group */
/*  File      : VEEProm.h */
/*  Description : #defines and parametrise the VIRTUAL EEPROM */
/* ----- */
/*  Supported  : IAR ICCM16C V2.11 */
/*  Compiler   */
/* ----- */
/*          R E V I S I O N   H I S T O R Y */
/* ----- */
/*  Date      Ver  Author  Description */
/* ----- */
/*  28/10/2003  1.0  ABI    Creation */
/* ***** */
/* ----- */
/*          R E M A R K S */
/* ----- */
// ***** external prototypes Flash function *****
// -----
// INFO: for below external function see file: VEEprom.c extern unsigned char

init_first_VEE(unsigned long block); // init flash block
extern void demo_writeflashVEE(void); // example to write flash
// VIRUTAL EEPROM

```

```
extern void demo_readflashVEE(void); // example to read flash VIRUTAL EEPROM
extern void demo_errorFlashVEE(void); // flash error handling

//#####
//***** define virtual EEprom Parameter *****
//#####
// The M16C62P version contain two Virtual EEPROM blocks BLOCK_A and BLOCK_1.
// "UNITTYPE" defines automaticaly the array type
// e.g. cDemoParaVar[ONE_UNITSIZE]
// ( pls. see file: VEEPROM.c).
// "ONE_UNITSIZE" defines the quantity of values in the array (independent of
// type)
// Example: 1. choise array name:          e.g. cDemoParaVar[ONE_UNITSIZE]
//          2. choise array type:          UNITYPE    => unsigned int
//          3. choise quantity of values: ONE_UNITSIZE => 256
//          4. Result:                     unsigned int cDemoParaVar[256]
//-----
#define UNITTYPE        unsigned int        //unique type for all data in the
                                           // virtual eeprom

#define ONE_UNITSIZE 256 // set needed UNIT size ...
                       // ... memory size[byte] => UNITTYPE * ONE_UNITSIZE
                       // example: 512 byte => int * 256

//-----
// !!!!!!!!!!!!!!! Don't change below parameter, values !!!!!!!!!!!!!!!
//-----
// The M16C62P version contain two Virtual EEPROM blocks BLOCK_A and BLOCK_1.
// Both blocks have a size of 4 Kbyte.
// In this VEE-Management demo Software the possible UNITS
// (in each BLOCK_A or _1)
// are calculated automatically after entering the UNITTYPE, ONE_UNITSIZE
// parameters in above #defines.
//-----

#define MAX_VIRUAL_EEPROM_SIZE    4096 // VIRTUAL EEPROM size for
                                       // BLOCK_A, BLOCK_1

#define MAX_UNITS    MAX_VIRUAL_EEPROM_SIZE/(ONE_UNITSIZE * sizeof(UNITTYPE))

//-----
// ----- Flash - start block address -----
//-----

#define BLOCK_0    0xFF000 // 4k byte Data Flash Block
#define BLOCK_1    0xFE000 // 4k byte Data Flash (Virtual EEPROM)
#define BLOCK_2    0xFC000 // 8k byte Data Flash Block
#define BLOCK_3    0xFA000 // 8k byte Data Flash Block
#define BLOCK_4    0xF8000 // 8k byte Data Flash Block
#define BLOCK_5    0xF0000 // 32k byte Data Flash Block
#define BLOCK_6    0xE0000 // 64k byte Data Flash Block
#define BLOCK_7    0xD0000 // 64k byte Data Flash Block
#define BLOCK_8    0xC0000 // 64k byte Data Flash Block
#define BLOCK_9    0xB0000 // 64k byte Data Flash Block
#define BLOCK_10   0xA0000 // 64k byte Data Flash Block
#define BLOCK_11   0x90000 // 64k byte Data Flash Block
```

```

#define BLOCK_12 0x80000          // 64k byte Data Flash Block
#define BLOCK_A  0x0F000          // Data Flash (Virtual EEPROM)

//-----
// Data Flash bock addresses - area for M16C62P
//-----
//----- to erase flash block take highest even addresse
#define ERASE_BLOCK_0 0xFFFFE     // 4k byte Data Flash Block
#define ERASE_BLOCK_1 0xFEFFE     // 4k byte Data Flash (Virtual EEPROM)
#define ERASE_BLOCK_2 0xFDFFE     // 8k byte Data Flash Block
#define ERASE_BLOCK_3 0xFBFFE     // 8k byte Data Flash Block
#define ERASE_BLOCK_4 0xF9FFE     // 8k byte Data Flash Block
#define ERASE_BLOCK_5 0xFFFFE     // 32k byte Data Flash Block
#define ERASE_BLOCK_6 0xEFFFE     // 64k byte Data Flash Block
#define ERASE_BLOCK_7 0xDFFFE     // 64k byte Data Flash Block
#define ERASE_BLOCK_8 0xCFFFE     // 64k byte Data Flash Block
#define ERASE_BLOCK_9 0xBFFFE     // 64k byte Data Flash Block
#define ERASE_BLOCK_10 0xAFFFE    // 64k byte Data Flash Block
#define ERASE_BLOCK_11 0x9FFFE    // 64k byte Data Flash Block
#define ERASE_BLOCK_12 0x8FFFE    // 64k byte Data Flash Block
#define ERASE_BLOCK_A 0x0FFFE     // Data Flash (Virtual EEPROM)

//-----
//-----
#define FLASH_ERROR_LED P3.B._5    // FLASH ERROR LED=
#define P3_LED_OUT      P3.B_REG   // output all LED on port 3
#define VEE_ERROR_LED  P3.B._5    // output VEE- flash error

```

3.2 Program file VEEProm.c

```

/* ----- */
/* Copyright (c) 2003 RENESAS TECHNOLOGY EUROPE All rights reserved. */
/* ----- */
/* Project Name: SSK M16C62P Group */
/* File : VEEporm.c */
/* Description : Virtual EEPROM Memory Management (for BLOCK_A, BLOCK_1) */
/* to increase the erase/write cycles for the virtual EEPROM */
/* - Program , re-program, erase and read the M16C62P */
/* - Virtual - EEPROM Flash in EW1 mode */
/* ----- */
/* Supported : IAR ICCM16C Version 2.11 */
/* Compiler */
/* ----- */
/* R E V I S I O N H I S T O R Y */
/* ----- */
/* Date Ver Author Description */
/* ----- */
/* 30/10/2003 0.1 ABI Creation */
/* ***** */
/* ----- */
/* R E M A R K S */
/* ----- */
/* Functionality: */
/* ----- */
/* This code implements the basic functions to flash and to manage the */
/* VIRTUAL EEPROM (BLOCK_A and BLOCK_1). The functions shows, how to write */
/* one word or copy a data array into the virtual flash area. */
/* - Press P8.B._2 (SWITCH_0) to write/copy data to the "virtual EEPROM */
/* ----- */
/* To increase the program and erase cycle of each virtual block (A,1), the */
/* program shows an example for an "Virtual EEPROM Memory Management". */
/* Therefor classify the virtual block in different UNITS and copy a data */
/* array into it. Erasy automatically the virtual flash block. */
/* ----- */
/* When an error occurs during the operation, the program enter into the */
/* error function and switch on the ERROR LED (port pin P3.6). The error */
/* falso can be analysed by reading the "flash_status" byte. */
/* ----- */
/* Before entering CPU rewrite mode (EW0 or EW1 mode), select 10 MHz or less*/
/* for CPU clock using the CM06 bit in the CM0 register and the CM17 to CM16*/
/* bits in the CM1 register. Also, set the PM17 bit in the PM1 register to 1*/
/* (with wait state). */
/* ----- */
/* - Make sure that any interrupt which has a vector in the variable vector */
/* table or address match interrupt will not be accepted during the auto */
/* program or auto erase period. Avoid using watchdog timer interrupts. */
/* ----- */
/* - The NMI interrupt can be used because the FMR0 register and FMR1 reg. */
/* are initialized when this interrupt occurs. The jump address for the */
/* interrupt service routine should be set in the fixed vector table. */
/* ----- */
/* Because the rewrite operation is halted when a NMI interrupt occurs, the */

```

```

/*rewrite program must be executed again after exiting the interrupt      */
/* service routine.                                                         */
/*                                                                           */
/* To set the FMR01,FMR02,or FMR11 bit to "1 ",write "0 " and then "1 " in */
/* succession. This is necessary to ensure that no interrupts or DMA      */
/* transfers will occur before writing "1 " after writing "0 ".Also only    */
/* when NMI pin is "H " level.                                           */
/*                                                                           */
/* Avoid rewriting any block in which the rewrite control program is stored.*/
/*                                                                           */
/* -Writing Command and Data                                              */
/* Write the command code and data at even addresses.                    */
/*                                                                           */
/* *****/
#include "ssk.h" // includes all SSK macros related header and
#undef EXTERN // M306NA SFR header and compiler dependant switches
#define EXTERN
#include "VEEProm.h" // classify virtual EEPROM BLOCK_A and B into different
#undef EXTERN // UNITS
//-----
//***** define prototypes *****
//-----

void demo_writeflashVEE(void); // example to flash VIRUTAL EEPROM
void demo_readflashVEE(void); // example to read
void demo_errorFlashVEE(void); // flash error handling

unsigned char copy_array2VEE(UNITTYPE *array, unsigned long VEE_block);
// copy parameter array to Virtual EEPROM

unsigned char write_data2VEE(UNITTYPE data, unsigned long VEEadr);
// write one data word to the Virtual EEPROM

void copy_VEE2array(UNITTYPE *array, unsigned long VEE_block);
UNITTYPE read_VEE2data(unsigned long VEEadr); // read one Virtual EEPROM data
// to RAM array

unsigned char erase_VEE(unsigned long block); // erase total Virtual EEPROM
unsigned char ReadLockBit(unsigned long block); // check if flash block is
// locked
unsigned char ClrLockBit(unsigned long block); // clear the lock bit in flash
// block

void init_VEE_EW1_mode(void); // setup erase orwrite VEE - Flash routine;
unsigned char init_first_VEE(unsigned long block);

void RestoreMCU(void); // Restore clock back to original speed and
// restores I flag back
void init_MCU_2FlashMode(void); // set the MCU mode for flash programming

//-----
//local variables
//-----

```



```
//=====
// input:
// output: return 1 -> block not locked
//         return 0 -> block is locked
// Info:   return lock bit
//-----
unsigned char ClrLockBit(unsigned long block)
{
    unsigned int far *VEE_flash_adr;

//---

    VEE_flash_adr = (unsigned int far *)block; // pointer to VEE-flash address
                                           // for ease -> set pointer to highest even address
    init_VEE_EW1_mode(); // init Flash for EW1 mode
    init_MCU_2FlashMode();

//---

    *VEE_flash_adr = 0x77; // (1. cycle) send read lock bit
    *VEE_flash_adr = 0xD0; // (2.cycle) send read lock bit

//---

    while (!FMR0.B._0); // check ready bit to ensure writing is complete

    if(FMR1.B._6 == 1) // Read flash erase status flag
        return 1; // return "1" block not locked
    else
        return 0; // return "0" block is locked
}
//##### End #####
```

3.3 Program file Main.c

```

/* -----*/
/* Copyright (c) 2003 RENESAS TECHNOLOGY EUROPE All rights reserved. */
/* -----*/
/* Project Name: SSK M16C62P Group */
/* File : main.c */
/* Description : Demo- Software for Virtual EEPROM Managment */
/* -----*/
/* Supported : IAR ICCM16C V2.11 */
/* Compiler */
/* -----*/
/* REVISION HISTORY */
/* -----*/
/* Date Ver Author Description */
/* -----*/
/* 30/10/2003 1.0 ABI Creation */
/* -----*/
#define _MAIN_C_ // Only used in its own header
#include "ssk.h" // includes all SSK macros related header and
                // M306NA SFR header and compiler dependant switches
#include "main.h" // variables, #defines
#include "hw_init.h" // init MCU timer, interrupts, ports,...

#define EXTERN extern
#include "VEEProm.h" // define for VEEprom, classify virual EEPROM
#undef EXTERN // BLOCK_A and B into differnt UNITS

//-----
//***** Main *****
//-----

void main(void)
{

    ssk_testsfr_init(); // init SFR to make them e.g. all visible
                        // in the PD30F debugger

    ssk_Port_init();

    ssk_TimerB_init(); // init timer TB0 interrupt
    _SSK_TB_START(0); // Start timer TB0

    enable_interrupt();

//----- Initialise Virtual EEPROM e.g. BLOCK_A (or BLOCK_1) -----

    if(init_first_VEE(BLOCK_1)) // first init VEEprom block
                                //[[in file: VEEprom.c]
        demo_errorFlashVEE(); // flash error handling

//-----

```

```
for (;;)
{
    if(SWITCH_0)          // press SWITCH_0 (P8.2) to flash Virtual EEPROM
        SWITCH_LED_0 = 0; // SWITCH_LED_0 = "1" no activ => LED=OFF
    else
    {
        SWITCH_LED_0 = 1; // switch = "L" => activate write VEE
        demo_writeflashVEE(); // SWITCH_LED_0 = "0" activate => LED=ON
                               // example: write flash Virtual EEPROM [in
                               // file: VEEprom.c]
        demo_readflashVEE(); // example: read flash Virtual EEPROM [in
                              // file: VEEprom.c]
    }

//-----
    if(TOGGLE_FLAG)      // this TOGGLE_FLAG is set via interrupt
        RUN_LED =1;     // toggle Run-LED
    else
        RUN_LED =0;     // toggle Run-LED
}

}
// #####
/* ----- End of of Code -----*/
```

Reference

Renesas Technology Corporation Semiconductor Home Page

<http://renesas.com>

Further documents about flash with M16C

<http://www.renesas.com/eng/products/mpumcu/16bit/m16c/apl/mc60apnode.html>

E-mail Support

[E-mail: support_apl@renesas.com](mailto:support_apl@renesas.com)

User's Manual

M16C Datasheet / User's Manual

(Use the latest version on the home page: <http://www.renesas.com>)

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Nov.03.03	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (<http://www.renesas.com>).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.