M16C/26

Using EW1 Mode for Flash Programming

1.0 Abstract

The following article introduces and shows how to use the EW1 mode of the CPU Rewrite feature on the M16C/26 (M30262) Flash microcontroller (MCU). The CPU Rewrite feature allows erasing and programming the on-chip (internal) user flash ROM area under control of a user's program. A short program written for the MSV30262-SKP demonstrates how to use this convenient feature.

2.0 Introduction

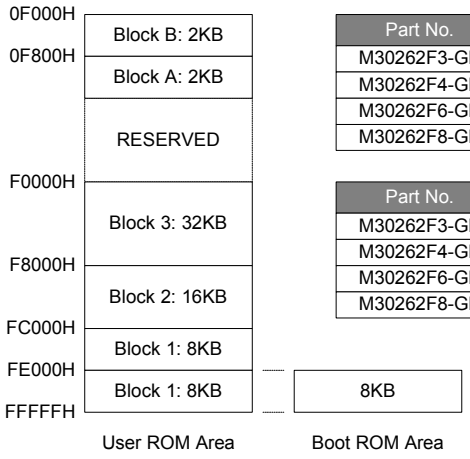
The Renesas M16C/26 is a 16-bit MCU, based on the M16C/60 CPU core, with up to 64KB of user flash and 4KB of Virtual EEPROM. The device has the ability to erase and program the internal user flash ROM area under control of a user's program with no external programming devices required. This feature is called "CPU Rewrite Mode".

The CPU Rewrite feature can be used in applications where data, such as registers, configuration status/ parameters, data log, etc., needs to be stored in non-volatile memory (i.e. flash memory) for future access.

3.0 CPU Rewrite

The M16C/26 has three flash programming modes: Parallel I/O Mode, Standard Serial I/O Mode, and CPU Rewrite Mode. The first two modes are mainly for programming the application code into the flash so details are not in the scope of this document.

In order to use CPU Rewrite Mode, the memory structure and control registers involved need to be identified. The internal flash memory map of the M16C/26, based on part number, is shown in Figure 1. Note that the flash is divided into blocks such that certain erase/programming functions are done on a per block basis.



Part No.	Block 3 Size	Start Address
M30262F3-GP	0 KB	N/A
M30262F4-GP	0 KB	N/A
M30262F6-GP	16KB	F4000H
M30262F8-GP	32KB	F0000H

Part No.	Block 2 Size	Start Address
M30262F3-GP	8KB	FA000H
M30262F4-GP	16KB	F8000H
M30262F6-GP	16KB	F8000H
M30262F8-GP	16KB	F8000H

Figure 1 M16C/26 (M30262F8) Flash Memory

Note: CPU Rewrite can only be used on the user ROM area but not the boot ROM area. The boot ROM area is used for serial I/O mode only and is not available for CPU Rewrite mode programming.

CPU Rewrite has two modes: EW0 and EW1. In EW0, the re-write program is executed from RAM (after being transferred from user or boot ROM). In EW1 mode, the re-write program can be executed in flash memory. However, care should be taken so the memory block where the CPU Rewrite program is being executed is different from the memory block where data will be written.

The registers used during CPU Rewrite mode are shown in Figure 2 and Figure 3. Another register, Flash Memory Control Register 4 (FMR4), which is for the Flash Erase-Suspend feature, is not discussed in this article.

Note: Currently, EW1 mode is only available for M16C/26 and M16C/62P MCU's. Contact your Renesas representative for details about other M16C MCU's or an article about Flash Erase-Suspend feature.

Flash memory control register 0

b7	b6	b5	b4	b3	b2	b1	b0
			0				

Symbol: FMR0 — Address: 01B7$_{16}$ — When reset: 00000001$_2$

Bit symbol	Bit name	Function	R	W
FMR00	RY/$\overline{BY}$ status flag	0: Busy (being written or erased) 1: Ready	○	×
FMR01	EW entry bit (Note 1)	0: Normal mode (Software commands invalid) 1: EW mode (Software commands acceptable)	○	○
FMR02	8Kbyte EW mode enable bit (Note 2)	0: EW mode disabled on 8Kbyte blocks 1: EW mode enabled on 8Kbyte blocks	○	○
FMR03	Flash memory reset bit (Note 3)	0: Normal operation 1: Reset	○	○
Reserved bit		Must always be set to 0	○	○
FMR05	User ROM area select bit (Note 4) (Only effective in boot mode)	0: Boot ROM area is accessed 1: User ROM area is accessed	○	○
FMR06	Program status flag	0: Pass 1: Error	○	×
FMR07	Erase status flag	0: Pass 1: Error	○	×

Figure 2 Flash Memory Control Register 0 (FMR0)

Flash memory control register 1

b7	b6	b5	b4	b3	b2	b1	b0
0		0	0	0	0		0

Symbol: FMR1 — Address: 01B5$_{16}$ — When reset: 0XX00101$_2$

Bit symbol	Bit name	F unction	R	W
Reserved bit		Must always be set to 0	○	○
FMR11	EW mode select bit (Note 1)	0: EW0 mode 1: EW1 mode	○	○
			○	○
Nothing is assigned. In an attempt to write to these bits, write 0. The value, if read, turns out to be indeterminate.			–	–
Reserved bit		Must always be set to 0	○	○

Figure 3 Flash Memory Control Register 1 (FMR1)

Figure 4 shows the CPU Rewrite process in EW1 mode.

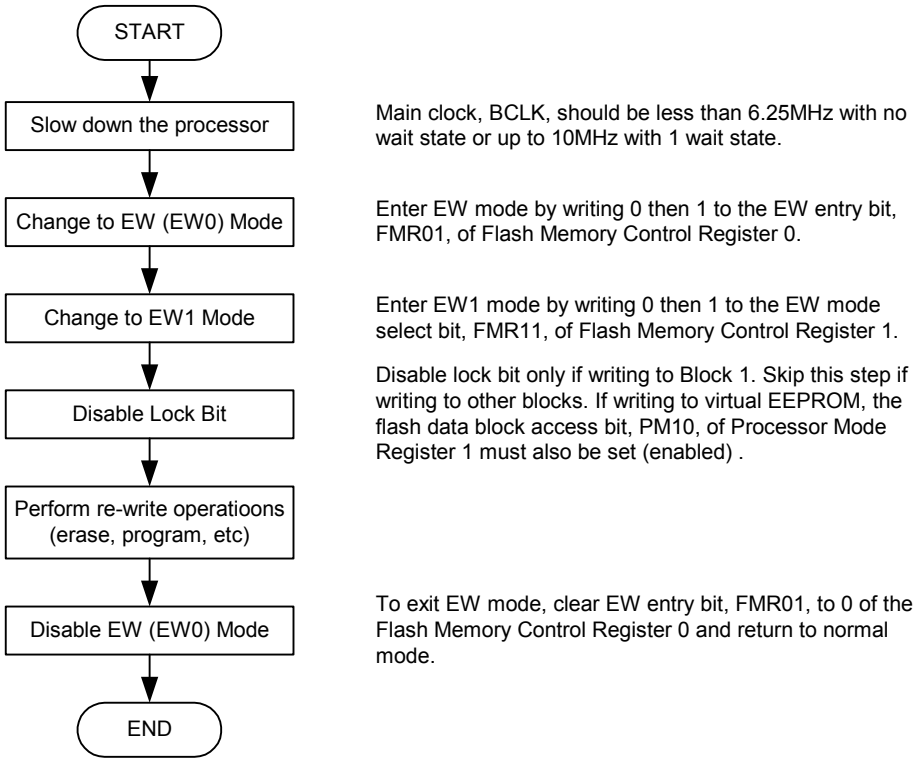


Figure 4 CPU Rewrite Process Flowchart

Table 1 lists the software commands that can be used in CPU Rewrite mode.

Table 1. Software Commands in CPU Rewrite Mode

Command	First bus cycle			Second bus cycle		
	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)
Read array	Write	X	FF_{16}			
Read status register	Write	X	70_{16}	Read	X	SRD (Note 2)
Clear status register	Write	X	50_{16}			
Program (Note 3)	Write	WA	40_{16}	Write	WA (Note 3)	WD (Note 3)
Block erase	Write	X	20_{16}	Write	BA (Note 4)	$D0_{16}$

Note 1: When a software command is input, the high-order byte of data (D_8 to D_{15}) 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 address of each block that is an even address.)
Note 5: X denotes a given address in the user ROM area (that is an even address).

3.1 CPU Rewrite Routine

This section shows the different software routines that are used to implement the CPU Rewrite process as described in Figure 4. The main program calls these routines (C functions), to perform erase, programming, status checks, etc. The software routines described here can be found in 'flash-26-ew1.c' file under the C:\MTOOL\MSV30262-SKP\Sample_Code\EW1 folder after MSV30262-SKP software installation.

3.1.1 Slow Down Processor

The processor speed, main clock (BLCK), must meet some speed requirements when performing CPU Rewrite operations. The main clock cannot be set greater than 10MHz. The speed requirements for CPU Rewrite operations are as follows:

- If main clock (BLCK) is greater than 6.25MHz but less than 10MHz, a wait state must be inserted.
- If main clock (BCLK) is less than 6.25MHz, a wait state is not necessary.

The code to slow down MCU speed is shown below.

```
void SlowMCU(void)
{
       asm("STC FLG,R0");    // Save contents of flag register
       asm("MOV.W R0,_flags_saved");

       asm("FCLR I");               // Turn off maskable interrupts
       cm0_saved = cm0;      // Save current CPU clock setting
       cm1_saved = cm1;
       pm1_saved = pm1;

       prcr = 3;             // Unprotect registers CM0 and PM0
       cm1 = 0xA0;           // Use Xin, Xin drive HIGH, Xin/4 (f4): BCLK=5MHz
//     pm17 = 1;             // if BLCK > 6.25MHz, insert a wait state
cm06 = 0;             // CM16 and CM17 are valid
}
```

3.1.2 Change to EW Mode and then EW1 Mode

To switch to EW (EW0) mode (or EW1 mode), a 0 is written to the bit and then followed by a 1.

```
fmcr01 = 0;
fmcr01 = 1;                        // Set EW0 select bit
fmcr11 = 0;
fmcr11 = 1;                        // Set to EW1 mode
```

3.1.3 Disable Lock Bit

Write a 0 and then a 1 to the 8KB EW Mode Enable bit, fmcr02, to disable “lock” if using the two Block 1 areas.

```
// disable flash memory lock bit (write-protect bit)
// only if re-write operations on the two Block 1 blocks
fmcr02 = 0;
fmcr02 = 1;
```

3.1.4 Perform CPU Rewrite Operations

After completing CPU Rewrite initialization, flash erase or write operations can be executed on the specified user Flash ROM area. Care must be taken to ensure that the block where the CPU Rewrite program is running is different from the block operated on. For example, if the CPU Rewrite code is running in Block 3, erase/program operations is NOT performed on Block 3 but on other blocks (Blocks 1,2, A, or B).

Code that performs programming operations (for the demo, Block B) is shown below. As can be seen below, status of the process can be checked using the software commands listed in Table 1.

```
while(bytes) {
        *flsh_addr = 0x50;          // Clear status register
        *flsh_addr = 0x40;          // Send write command
        *flsh_addr =  *data_buff;   // Write next word of data

        while (!(*flash_status_addr & 0x0100)); // check ready bit to ensure
writing is complete

        // Read flash program status flag
        if( *flash_status_addr & 0x4000 ){ // Write Ok/NG? - NG if true
                fmcr01 = 0;   // disable EW mode by clearing EW entry bit
                RestoreMCU(); // Restore clock back to original speed
                              // and restores I flag back
                return 0;     // Write Fail (Cancel the rest of operation)
        }
        bytes -= 2;           // subtract 2 from byte counter
        data_buff++;          // increase to next data index
        flsh_addr++;          // increase to next flash index
}
```

3.1.5 Disable EW Mode and Return to Normal Operation

After completing CPU Rewrite Operations, we need to disable CPU Rewrite mode and return to normal operation. To accomplish this, the EW Entry Bit is cleared to 0 and then the restore MCU function is called.

```
        fmcr01 = 0;           // disable EW mode by clearing EW entry bit
        RestoreMCU();         // Restore clock back to original speed
                              // and restores I flag back
```

The code to restore MCU speed is shown below.

```
void RestoreMCU(void)
{
        pm1 = pm1_saved;
        cm1 = cm1_saved;
        cm0 = cm0_saved;
        prcr = 0;             // Protection register back on
        // Restore contents of flags (I flag in particular)
        asm("MOV.W _flags_saved,R0");
        asm("LDC R0,FLG");
        asm("FSET I");              // Turn on maskable interrupts
}
```

4.0 EW1 Demo Program

The demo program was written to run on the MSV30262-SKP board and has two modes but both operate on Block B (addresses 0x0F000 to 0x0F0FF), one of the two data (virtual EEPROM) blocks. One mode writes incremental data (0 – F) while the other mode writes fixed data, 'M16C/26 Firefly'. Pressing S4 will toggle between modes. The program has additional functions to erase data or display the data on the LCD for verification without the use of a debugger.

A copy of the source files can be found under the C:\MTOOL\MSV30262-SKP\Sample_Code\EW1 folder after MSV30262-SKP software installation. The program was compiled using the KNC30 Compiler, which also came with the MSV30262-SKP. It can be modified to suit a user application.

4.1 EW1 Demo – Mode 0

Mode 0 is the default mode after running the program. The following CPU Rewrite operations on Block B (0x0F000 – 0x0F0FF) can be performed in Mode 0:

- Pressing S2 will write incremental data, 0 – F.
- Pressing S3 will write 'M16C/26 Firefly '.
- Pressing S4 will toggle to Mode 1.

4.2 EW1 Demo – Mode 1

Mode 1 allows the user to view the data in Block B (0x0F000 – 0x0F0FF) for verification purposes. The following functions are performed in Mode 1.

- Pressing S2 will display data starting from 0x0F000. Pressing S2 again will display the data of the next address and so on.
- Pressing S3 will erase the data.
- Pressing S4 will toggle back to Mode 0.

5.0 Conclusion

CPU Rewrite Mode – EW1 allows a simpler method of saving data to on-chip user ROM area. It can be used in applications where data, such as configuration parameters, log, status, etc., needs to be stored in non-volatile memory for later access.

6.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

- M16C/26 datasheets, M30262eds.pdf

User's Manual

- M16C/20/60 C Language Programming Manual, 6020c.pdf
- M16C/20/60 Software Manual, 6020software.pdf
- MSV30262-SKP Users Manual, Users_Manual_MSV30262.pdf

7.0 Software Code

The EW1 demo's CPU Rewrite routines (in flash-26-ew1.c) is shown below. The complete project, written in C, can be compiled/linked using the KNC30 Compiler and will be provided upon request. Please contact your Renesas representative for details.

```
/***********************************************************************
*       File Name:  flash-26-ew1.c
*
*       Content: CPU Re-write functions for M16C/26 that includes erase, program,
*                CPU speed slow down and CPU speed restore.
*
*       Revision 1.1  2003-02-21
***********************************************************************/
#include "flash-26-ew1.h"
#include "sfr262.h"

/* We want to read the fmcr0 register (address 0x1B7), but the spec says that we should only
read the status bits in the flash memory control register using even addressing when in EW1
Mode. Therefore, we will read address 0x1B6 and only use the upper byte */

const unsigned int * flash_status_addr = (unsigned int *)0x1B6;

/* Variables for saving the Processor Mode and Clock Mode registers */
static unsigned char pm1_saved, cm0_saved, cm1_saved;
static unsigned int flags_saved;
```

```
/* List of highest even addresses for each block for M16C/26 */
const unsigned long block_addresses[6] = {
                0xFFFFE,0xFDFFE,0xFBFFE,0xF7FFE,       // Code Blocks 0, 1, 2, & 3
                0xFFFE, 0xF7FE                         // Data Flash (Virtual EEPROM) Blocks
4 & 5
                };

/* Prototypes of functions only used by this file */
void SlowMCU(void);
void RestoreMCU(void);

/***********************************************************************
Name:   FlashErase
Parameters:
block
The block number to erase (0 - 5)
Returns:
1 = Erase Successful
        0 = Erase error reported by flash memory control register 0
Description:
        Erases an entire flash block using EW1 Mode
***********************************************************************/
int FlashErase( int block ) {

        far unsigned int * flsh_addr;

        // Get highest even block address
        flsh_addr = (far unsigned int *) block_addresses[ block ];
        SlowMCU();                      // Must change main clock speed to meet flash
                                        // requirements as well as turn off maskable
                                        // interrupts
        fmcr01 = 0;
        fmcr01 = 1;                     // Set EW0 select bit
        fmcr11 = 0;
        fmcr11 = 1;                     // Set to EW1 mode

        // disable flash memory lock bit (write-protect bit)
        // only if re-write operations on the two Block 1 blocks
        fmcr02 = 0;
        fmcr02 = 1;

        *flsh_addr = 0x50;                      // Clear status register
        *flsh_addr = 0x20;                      // Send erase command
        *flsh_addr = 0xD0;                      // Send erase confirm command

        while (!(*flash_status_addr & 0x0100)); // check ready bit to ensure erase is complete
```

```
        if( *flash_status_addr & 0x8000 ){      // Erasing error?
                fmcr01 = 0;                     // disable EW mode by clearing EW entry bit
                RestoreMCU();                   // Restore clock back to original speed
                                                // and restores I flag back
                return 0;                       // Erase Fail
        }
        fmcr01 = 0;                             // disable EW mode by clearing EW entry bit
        RestoreMCU();                           // Restore clock back to original speed
                                                // and restores I flag back
        return 1;                               // Erase Pass
}
/***********************************************************************
Name:   FlashWrite
Parameters: flash_addr
Flash address location to write to. Must be an EVEN address!
        buffer_addr
                Address location of data buffer to write to flash bytes
                The number of bytes to write. Must be an EVEN number!
Returns:
        1 = Operation Successful
        0 = Write Error reported by flash control register
Description:
                Writes bytes into flash. The number of bytes to write MUST be an even
                number because the flash controller has to write a WORD at a time. The
                flash address MUST be an even number as well because the flash
                controller needs to write WORDS to even addresses only.
***********************************************************************/
int FlashWrite( unsigned long flash_addr,
                far unsigned char * buffer_addr,
                unsigned int bytes) {

        far unsigned int * flsh_addr;
        far unsigned int * data_buff;

        flsh_addr = (far unsigned int *) flash_addr;
        data_buff = (far unsigned int *) buffer_addr;

        SlowMCU();                      // Must change main clock speed to meet flash
                                        // requirements as well as turn off maskable
                                        // interrupts
        fmcr01 = 0;
        fmcr01 = 1;                     // Set EW0 select bit
        fmcr11 = 0;
        fmcr11 = 1;                     // Set to EW1 mode

        // disable flash memory lock bit (write-protect bit)
        // only if re-write operations on the two Block 1 blocks
        fmcr02 = 0;
        fmcr02 = 1;

        while(bytes) {
                *flsh_addr = 0x50;                      // Clear status register
                *flsh_addr = 0x40;                      // Send write command
                *flsh_addr =  *data_buff;               // Write next word of data
```

```
                while (!(*flash_status_addr & 0x0100)); // check ready bit to ensure writing
is complete

                // Read flash program status flag
                if( *flash_status_addr & 0x4000 ){     // Write Ok/NG? - NG if true

                        fmcr01 = 0;             // disable EW mode by clearing EW entry bit
                        RestoreMCU();           // Restore clock back to original speed
                                                // and restores I flag back
                        return 0;               // Write Fail (Cancel the rest of operation)
                }
                bytes -= 2;                     // subtract 2 from byte counter
                data_buff++;                    // increase to next data index
                flsh_addr++;                    // increase to next flash index
        }
        fmcr01 = 0;                             // disable EW mode by clearing EW entry bit
        RestoreMCU();                           // Restore clock back to original speed
                                                // and restores I flag back
        return 1;                               // Write Pass
}
/***********************************************************************
Name:         SlowMCU
Parameters:   none
Returns:      nothing
Description:  Sets the processor mode for programming flash and saves current
              settings to restore later. When programming the M16C/26, you
              cannot run the processor faster than 6.25MHz (without wait state)
              when performing flash commands. See spec for more details.
***********************************************************************/
void SlowMCU(void)
{
        asm("STC FLG,R0");      // Save contents of flag register
        asm("MOV.W R0,_flags_saved");
        asm("FCLR I");          // Turn off maskable interrupts

        cm0_saved = cm0;        // Save current CPU clock setting
        cm1_saved = cm1;
        pm1_saved = pm1;

        prcr = 3;               // Unprotect registers CM0 and PM0
        cm1 = 0xA0;             // Use Xin, Xin drive HIGH, Xin/4 (f4): 20MHz/4 = 5MHz
//      pm17 = 1;               // if Xin/2 (f2): 20MHz/2 = 10MHz, a wait state must be inserted
        cm06 = 0;               // CM16 and CM17 are valid
}
/***********************************************************************
Name:         RestoreMCU
Parameters:   none
Returns:      nothing
Description:  Restores the processor mode back to original settings.
***********************************************************************/
```

```
void RestoreMCU(void)
{
        pm1 = pm1_saved;
        cm1 = cm1_saved;
        cm0 = cm0_saved;

        prcr = 0;                       // Protection register back on

                                        // Restore contents of flags (I flag in particular)
        asm("MOV.W _flags_saved,R0");
        asm("LDC R0,FLG");
        asm("FSET I");                  // Turn on maskable interrupts
}
```

Keep safety first in your circuit designs!

- 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

- 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.
- 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.
- 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).
- 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.
- 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.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- 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.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.