1. Abstract

This document describes the use example of CPU rewrite mode (EW0 mode).

2. Introduction

The application example described in this document applies to the following microcomputers (MCUs):

- MCU: M16C/63 Group, M16C/64 Group, M16C/64A Group, M16C/64C Group, M16C/65 Group (only program ROM 1 is 512K byte or less), M16C/65C Group, M16C/6C Group, M16C/5LD Group, M16C/56D Group, M16C/5L Group, M16C/56 Group, M16C/5M Group, M16C/57 Group

This application note can be used with other M16C Family MCUs which have the same special function registers (SFRs) as the above groups. Check the user’s manual for any modifications to functions. Careful evaluation is recommended before using the program described in this application note.
3. CPU Rewrite Mode

3.1 EW0 Mode Features

EW0 mode allows the user to rewrite the user ROM and the data areas by issuing program and erase commands generated from the CPU rewrite program already transferred to the RAM. The CPU continues to operate during program and erase operations in EW0 mode. Peripheral function interrupts will be accepted during program and erase commands if the vector and the interrupting program are located in the RAM.

3.2 EW0 Mode Settings

The MCU enters CPU rewrite mode when the FMR01 bit in the FMR0 register is set to 1 (CPU rewrite mode enabled) and is ready to accept commands. EW0 mode is selected by setting the FMR60 bit in the FMR6 register to 0. Software commands control programming and erasing. The FMR0 register or status register indicates whether a program or erase operation is completed as expected or not. Figure 3.1 shows Setting and Resetting of EW0 Mode.
Procedure to Enter EW0 Mode

1. **Single-chip mode or memory expansion mode**
2. **Transfer the rewrite control program to an area other than flash memory (1)**
3. **Set registers CM0, CM1, and PM1**
4. **Jump to the rewrite control program transferred to an area other than the flash memory. (In the following steps, use the rewrite control program in an area other than the flash memory)**

**Rewrite control program (1)**

- **FMR01 ← 0**
- **FMR01 ← 1**
- **FMR11 ← 1**
- **FMR6 ← 0x02**
- **FMR11 ← 0**
- **Execute the software commands**
- **Execute the read array command**
- **FMR01 ← 0**
- **Jump to the desired address in the flash memory**

**Note:**
1. Bits PM10 and PM13 in the PM1 register become 1 in CPU rewrite mode. Execute the rewrite program in internal RAM or an external area which can be used when both bits PM10 and PM13 are 1.
2. Do not use the area (40000h to BFFFFh) where accessible space is expanded when the PM13 bit is 0 and 4-MB mode is set.

**Figure 3.1 Settings and Resetting of EW0 Mode**
3.3 Memory Map

The flash memory is used as ROM in this product. The flash memory comprises program ROM 1, program ROM 2, and data flash.

The flash memory is divided into several blocks, each of which can be protected (locked) from programming or erasing. The flash memory can be rewritten in CPU rewrite, standard serial I/O, and parallel I/O modes.

If the size of program ROM 1 is over 512 KB, blocks 8 to 11 can be used when the IRON bit in the PRG2C register is 1 (program ROM 1 addresses 40000h to 7FFFFh enabled).

Program ROM 2 can be used when the PRG2C0 bit in the PRG2C register is set to 0 (program ROM 2 enabled).

Program ROM 2 includes a user boot code area.

Data flash can be used when the PM10 bit in the PM1 register is set to 1 (0E000h to 0FFFFh: data flash).

Data flash is divided into block A and block B.

Figure 3.2 shows a Flash Memory Block Diagram for M16C/65.

Refer to the respective hardware manuals for other models.
The Use Example of CPU Rewrite Mode (EW0 mode)

Figure 3.2 Flash Memory Block Diagram
4. Description of the Application Example

This application note provides a monitor program example where the sample program is received from the master device and the sample program execute and program ROM 2 area rewrite commands are executed.

Figure 4.1 shows the system structure.

Figure 4.1 System Structure Diagram

The control commands used in this application note are as follows.

<table>
<thead>
<tr>
<th>Table 4.1 Control Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Command Name</td>
</tr>
<tr>
<td>Program (write) command</td>
</tr>
<tr>
<td>Erase command</td>
</tr>
<tr>
<td>Sample program execute command</td>
</tr>
</tbody>
</table>

Note:
1. When the program and erase have successfully completed, 6Fh (‘o’) is returned. If an error occurs, 65h (‘e’) is returned.

UART0 clock asynchronous serial I/O mode is used in communications with the master device. The UART0 settings are as follows.

- Mode: Clock asynchronous serial I/O mode
- Communication bit rate: 38400 bps
- CTS/RTS: N/A
- Stop bit: 1 stop bit
- Parity: None
- Data bit length: 8 bits
The following explains the operations of this application note.

1) The monitor program waits to receive the control command.

2) When the received command is "prg"
   2-1) Receive sample size (2-byte data).
   2-2) Receive one packet (maximum 256 bytes) of program data.
   2-3) Receive packet data sum value (2-byte data).
   2-4) Compare received packet data sum value and the received sum value.
      2-4-1) If no match, send error code to master device.
      2-4-2) If they match, the CPU clock is set to 10 MHz or lower and one packet of data is written in the
              program ROM 2 area before returning the CPU clock to its original setting.
      2-4-2-1) When the data has been successfully written, the write complete code is sent to the master device.
      2-4-2-2) If a write error occurs, an error code is sent to the master device and program data receipt is
               stopped.
   2-5) If an error does not occur, steps (2-2) through (2-4) are repeated until receipt of the sample program is
       completed.

3) When the received command is "ers"
   3-1) The CPU clock is set to 10 MHz or lower and the program ROM 2 area is erased before returning the CPU
       clock to its original setting.
   3-2-1) When successfully erased, the erase complete code is sent to the master device.
   3-2-2) If an erase error occurs, an error code is sent to the master device.

4) When the received command is "run"
   4-1) The sample program written in the program ROM 2 area is executed.

Figure 4.2. shows an example of the monitor program operation.
Figure 4.2 Monitor Program Operation Example
## 5. Structure

| Declaration | typedef struct buff{
|             |unsigned char prg_data[RECORD_SIZE];
|             |unsigned short rev_sum;
|             |}REV_BUFF; |

| Variable | unsigned char prg_data[RECORD_SIZE] | RECORD_SIZE (256) byte sample program storage array |
|          | unsigned short rev_sum              | Sum value storage variable |

| Function | Structure that stores the received sample program (256 bytes) and the sum value. |
## 6. Function Table

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void peripheral_init(void)</th>
<th>void cpu_slow(void)</th>
<th>void cpu_fast(void)</th>
<th>unsigned char rev_byte(unsigned char *rev_data)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
<td>Peripheral function initial setting function</td>
<td>CPU slowdown process function</td>
<td>CPU speed up process function</td>
<td>Command 1-byte receive function</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Argument Type</td>
</tr>
<tr>
<td><strong>Variable(global)</strong></td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Meaning</td>
</tr>
<tr>
<td><strong>Returned value</strong></td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>unsigned char</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Sets the UART0 send and receive setting and the timer A0 10 ms timer setting.</td>
<td>Sets the main clock divider to CM06 = 0, CM17 - CM16 = 01 (divide-by-2 mode), PM17 = 1 (1 wait).</td>
<td>Sets the main clock divider to CM06 = 0, CM17-CM16 = 00 (no division mode), PM17 = 0 (no wait).</td>
<td>Stores the 1 byte received data in the array.</td>
</tr>
<tr>
<td><strong>Argument Type</strong></td>
<td></td>
<td></td>
<td></td>
<td>Argument Type</td>
</tr>
<tr>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
<td>Meaning</td>
</tr>
<tr>
<td><strong>Return Value Type</strong></td>
<td></td>
<td></td>
<td></td>
<td>Return Value Type</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td></td>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
<td>Meaning</td>
</tr>
<tr>
<td><strong>unsigned char</strong></td>
<td>COMPLETE</td>
<td>TIMEOUT</td>
<td>RECEIVE_ERROR</td>
<td>Successfully completed</td>
</tr>
<tr>
<td><strong>COMPLETE</strong></td>
<td>Timeout</td>
<td></td>
<td></td>
<td>Timeout</td>
</tr>
<tr>
<td><strong>TIMEOUT</strong></td>
<td></td>
<td></td>
<td></td>
<td>Receive error</td>
</tr>
<tr>
<td><strong>RECEIVE_ERROR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Argument**
- Argument Type: Meaning
- Argument Type: unsigned char *rev_data
- Receive command storage array address

**Variable(global)**
- None

**Returned value**
- None

**Function**
- Stores the 1 byte received data in the array.
### Declaration
```
unsigned char rev_cmd_check(unsigned char *cmd_buff)
```

### Outline
Command check function

### Argument
<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char *cmd_buff</td>
<td>Start address of the received command storage array</td>
</tr>
</tbody>
</table>

### Variable (global)
None

### Returned value
<table>
<thead>
<tr>
<th>Return Value Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>REV_ERASE</td>
<td>Erase command received</td>
<td></td>
</tr>
<tr>
<td>REV_PROGRAM</td>
<td>Program command received</td>
<td></td>
</tr>
<tr>
<td>REV_RUN</td>
<td>Sample program execution command received</td>
<td></td>
</tr>
<tr>
<td>REV_ERROR</td>
<td>Receive error</td>
<td></td>
</tr>
</tbody>
</table>

### Function
Determines the received character string and returns the appropriate command.

### Declaration
```
void erase(void)
```

### Outline
Flash erase function

### Argument
None

### Variable (global)
None

### Returned value
None

### Function
Executes the block erase function located on the RAM, determines if the erase was successful and sends a message.

### Declaration
```
void receive_program(void)
```

### Outline
Flash write function

### Argument
None

### Variable (global)
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>REV_BUFF rb</td>
<td>Structure that stores the received program data and the sum value</td>
</tr>
</tbody>
</table>

### Returned value
None

### Function
Receives the sample program size, program data and sum value sent from the master device. Executes the program function located on the RAM and writes the received program data. Determines if it the write was received correctly and sends a message.

### Declaration
```
unsigned short rev_size(void)
```

### Outline
Sample program size receive function

### Argument
None

### Variable (global)
None

### Returned value
<table>
<thead>
<tr>
<th>Return Value Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short</td>
<td>rev_size</td>
<td>Received size data</td>
</tr>
</tbody>
</table>

### Function
Receives the sample program size sent from the master device.
### The Use Example of CPU Rewrite Mode (EW0 mode)

**Declaration**  
`unsigned char rev_data(REV_BUFF *buff,unsigned short *size)`  

**Outline**  
Sample program data receive function  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>REV_BUFF *buff</td>
<td>Start address of the receive data storage structure</td>
<td></td>
</tr>
<tr>
<td>unsigned short *size</td>
<td>Address of the variable where the size is stored</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable(global)</th>
<th>Variable Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>REV_BUFF rb</td>
<td>Stores sum value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned value</th>
<th>Return Value Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>COMPLETE</td>
<td>Receive successful</td>
<td></td>
</tr>
<tr>
<td>FAIL</td>
<td>Receive failed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Function**  
Receives 256 bytes sample program data and the sum value. Compares the received packet data sum value and the received sum value. When the received data is below the record size, the remaining space is filled with 0xFFh.

---

**Declaration**  
`void note_program_start(void)`  

**Outline**  
Sample program execute function  

<table>
<thead>
<tr>
<th>Argument</th>
<th>None</th>
</tr>
</thead>
</table>

| Variable(global) | None |

| Returned value | None |

**Function**  
Executes the sample program written in the 10000h address.

---

**Declaration**  
`void send_message(const unsigned char *mess)`  

**Outline**  
Message send function  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>const unsigned char mess*</td>
<td>Start address of the send message array</td>
<td></td>
</tr>
</tbody>
</table>

| Variable(global) | None |

| Returned value | None |

**Function**  
Sends a message.

---

**Declaration**  
`void send_to_ram(void)`  

**Outline**  
Write control program transfer function  

<table>
<thead>
<tr>
<th>Argument</th>
<th>None</th>
</tr>
</thead>
</table>

| Variable(global) | None |

| Returned value | None |

**Function**  
Transfers the erase function, program function and the full status check function to the RAM.

---

**Declaration**  
`void send_to_ram_vector(void)`  

**Outline**  
Interrupt handler for RAM transfer function  

<table>
<thead>
<tr>
<th>Argument</th>
<th>None</th>
</tr>
</thead>
</table>

| Variable(global) | None |

| Returned value | None |

**Function**  
Transfers the interrupt handler to use on the RAM.
### The Use Example of CPU Rewrite Mode (EW0 mode)

#### Declaration

- **void renewal_of_ram_vector(void)**

**Outline**
Relocatable vector table for RAM create function

**Argument**
None

**Variable (global)**
None

**Returned value**
None

**Function**
Create the relocatable vector table for the RAM.

#### Declaration

- **unsigned char block_erase_ew0(unsigned short far* ers_addr)**

**Outline**
Block erase function

**Argument**

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short far*</td>
<td>ers_addr</td>
</tr>
</tbody>
</table>

**Variable (global)**
None

**Returned value**

<table>
<thead>
<tr>
<th>Return Value Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>COMPLETE</td>
<td>Successfully completed</td>
</tr>
<tr>
<td></td>
<td>CMD_SEQ_ERR</td>
<td>Command sequence error</td>
</tr>
<tr>
<td></td>
<td>PROGRAM_ERR</td>
<td>Program write error</td>
</tr>
<tr>
<td></td>
<td>ERASE_ERR</td>
<td>Erase error</td>
</tr>
</tbody>
</table>

**Function**
Erases the specified block in EW0 mode and executes a full status check. Returns the appropriate error message when an error occurs.

#### Declaration

- **unsigned char program_write_ew0(unsigned short far* write_addr,unsigned short *buff)**

**Outline**
Program function

**Argument**

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short far*</td>
<td>write_addr</td>
</tr>
<tr>
<td>unsigned short</td>
<td>*buff</td>
</tr>
</tbody>
</table>

**Variable (global)**
None

**Returned value**

<table>
<thead>
<tr>
<th>Return Value Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>COMPLETE</td>
<td>Successfully completed</td>
</tr>
<tr>
<td></td>
<td>CMD_SEQ_ERR</td>
<td>Command sequence error</td>
</tr>
<tr>
<td></td>
<td>PROGRAM_ERR</td>
<td>Program write error</td>
</tr>
<tr>
<td></td>
<td>ERASE_ERR</td>
<td>Erase error</td>
</tr>
</tbody>
</table>

**Function**
Writes 256 bytes of data from the specified address in EW0 mode.

#### Declaration

- **unsigned char full_status_check(void)**

**Outline**
Full status check function

**Argument**
None

**Variable (global)**
None

**Returned value**

<table>
<thead>
<tr>
<th>Return Value Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>COMPLETE</td>
<td>Successfully completed</td>
</tr>
<tr>
<td></td>
<td>CMD_SEQ_ERR</td>
<td>Command sequence error</td>
</tr>
<tr>
<td></td>
<td>PROGRAM_ERR</td>
<td>Program write error</td>
</tr>
<tr>
<td></td>
<td>ERASE_ERR</td>
<td>Erase error</td>
</tr>
</tbody>
</table>

**Function**
Executes a full status check and returns the results.
### The Use Example of CPU Rewrite Mode (EW0 mode)

#### Declaration
```c
void asm_smovf(void _far *source, void _near *dest, unsigned int _size)
```

#### Outline
RAM transfer process function

#### Argument
<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>void _far *source</td>
<td>Source address (program)</td>
</tr>
<tr>
<td>void _near *dest</td>
<td>Destination address (RAM area)</td>
</tr>
<tr>
<td>unsigned int _size</td>
<td>Transfer size</td>
</tr>
</tbody>
</table>

#### Variable(global)
None

#### Returned value
None

#### Function
Transfers the specified area to the RAM area.

#### Declaration
```c
void ram_int_dummy(void)
```

#### Outline
Interrupt handler for RAM of dummy function

#### Argument
None

#### Variable(global)
None

#### Returned value
None

#### Function
Dummy function for the RAM. Add program if needed.
7. Flowchart

7.1 Main Function

main(void)

- CPU initial setting processing
  - mcu_init()

- Peripheral function initialization processing
  - peripheral_init()

Reception size initialization

3 bytes received?
- Yes (i = 3)
  - The command 1 byte reception
    - rev_byte(&rev_command[i])

- No (i = 2)
  - Received correctly?
    - No (result = TIMEOUT or REV_ERROR)
    - Yes (result = COMPLETE)
      - Increment receive size
      - Reception size initialization

A reception command check
  - rev_cmd_check(&rev_command[0])

A reception command

- command = REV_ERASE
  - Flash erase
    - erase()

- command = REV_PROGRAM
  - Flash write
    - receive_program()

- command = REV_RUN
  - Sample program practice
    - note_program_start()
7.2 CPU Initial Setting Function

```c
mcu_init(void)
{
    prcr ← 0x43  // Protection OFF
    pm0 ← 0x00  // Single-chip mode
    pm1 ← 0x08  // The entire area is usable. No wait state

    prg2c ← 0x00  // Enable program ROM 2

    cm2 ← 0x00  // Main clock used as CPU clock.

    cm1 ← 0xA0  // Main clock divide-by-4 mode

    cm0 ← 0x08  // Main clock divide-by-2 mode

    cm1 ← 0x60  // Main clock no division mode

    cm1 ← 0x20  // Protection ON

    prcr ← 0x00

    return
}
```
7.3 Peripheral Function Initial Setting Function

```c
peripheral_init(void) {

UART0 setting
udkse0 ← 0x00  // UART0 clock prior to division: f1
prc0 ← 0x01
pclkr ← 0x03
prc1 ← 0x00
u0mr ← 0x05
u0c0 ← 0x08
u0brg ← XIN_BRG
u0c1 ← 0x07

Transmission rate of XIN
XIN_BRG = \frac{8 \text{ MHz}}{38400 \text{ bps} \times 16} - 1

Timer A0 setting
tacs0 ← 0x00
ta0mr ← 0x82
ta0 ← TIM10MS
ta0ic ← 0x00
ta0s ← 1

return

SI/O clock: f1SIO, Timer A clock: f1TIMAB
Protection OFF
Protection ON
UART mode character bit length is 8 bits, Internal clock, one stop bit, Parity disabled, No reverse
U0BRG count source: f1SIO
Transmission enabled
Reception enabled

8 MHz
38400 bps × 16

Bits TCK1 to TCK0 enabled
One-shot timer mode
Count source: f32
10 ms timer setting
Timer A0 interrupt disabled
Timer A0 count starts

}```
7.4 CPU Slowdown Process Function

```
cpu_slow(void)
prcr ← 0x03
    Protection OFF
    cm1 ← 0x60
        Main clock divide-by-2 mode
    cm0 ← 0x08
    pm17 ← 1
        Wait state (1 wait)
prcr ← 0x00
    Protection ON
return
```

7.5 CPU Speed Up Process Function

```
cpu_fast(void)
prcr ← 0x03
    Protection OFF
    cm1 ← 0x20
        Main clock no-division mode
    cm0 ← 0x08
    pm17 ← 0
        No wait state
prcr ← 0x00
    Protection ON
return
```
7.6 Command 1-byte Receive Function

```
rev_byte(unsigned char *rev_data)

result ← COMPLETE  # The initialization of the reception result

ta0ic ← 0x00  # Timer A0 interrupt request clear

ta0os ← 1  # Timer A0 one-shot start

1 byte received?
  Yes (ri_u0c1 = 1)
    10 ms elapsed?
      Yes (ir_ta0ic = 1)
        result ← TIMEOUT
      No (ir_ta0ic = 0)
      result ← COMPLETE
    No (ri_u0c1 = 0)

result ← RECEIVE_ERROR  # It is stored away once by a buffer.

rev_buff ← u0rb  # It is stored away reception data by an array.

*rev_data ← rev_buff & 0x00ff

result ← COMPLETE  # The initialization of the reception result

result ← RECEIVE_ERROR

return (result)
```

Timeout is processed for command receive only. Timeout is not processed for other messages received, so the user must add them, if necessary.
7.7 Command Check Function

```
rev_cmd_check (unsigned char *cmd_buff)

command ← REV_ERROR

cmd_buff = ers

command ← REV_ERASE

cmd_buff = prg

command ← REV_PROGRAM

cmd_buff = run

command ← REV_RUN

return (command)
```

- The initialization of the reception result
- The erase command reception
- The program command reception
- The sample program practice command reception
7.8 Flash Erase Function

erase(void)

result ← FAIL

The note control program transmission
send_to_ram()

Interrupt handler for RAM transfer function
send_to_ram_vector()

Relocatable vector table for RAM create function
renewal_of_ram_vector_t()

ep ← RP_ERASE

CPU slow process
cpu_slow()

Save to the INTB register

Set address of the relocatable vector table for RAM to the INTB register

Execute erase program located on the RAM
(*ep)((unsigned short far*)(BLOCK_PGROM2_END))

Recover the INTB register

CPU fast process
cpu_fast()

Erased correctly?

Yes (result = COMPLETE)

The erase completion transmission
send_message(mess_erase_ok)

No (result = FAIL)

The erase error transmission (1)
send_message(mess_err)

return

Note:
1. This application note only sends a message when an error occurs.
Add error processes as necessary.
7.9 Flash Write Function

receive_program(void)

write_start_addr ← BLOCK_PGROM2

Set 10000h address (program ROM 2 start address).

Sample program size received
rev_size()

Receive the sent program size.

No error and no size received

No

Yes

Sample program data received
rev_data(&rb,&r_size)

No (result = FAIL)

Do sum values match?

Yes (result = COMPLETE)

Write control program transferred
send_to_ram()

Transfer write control program to RAM.

Interrupt handler for RAM transfer function
send_to_ram_vector()

Transfer interrupt handler used on RAM.

Relocatable vector table for RAM
create_function
renewal_of_ram_vector_t()

Create relocatable vector table for RAM.

wp ← RP_PROGRAM

Store write program address in wp.

CPU slow process
cpu_slow()

Save to the INTB register

Set address of relocatable vector table for RAM to the INTB register

Execute write process located on the RAM
("wp)(unsigned short far*,unsigned short")

Recover the INTB register

CPU fast process
cpu_fast()

Update write address.

write_start_addr ← write_start_addr + (RECORD - SIZE / 2)

Written correctly?

No

Yes (result = COMPLETE)

The program completion transmission
send_message(mess_program_ok)

The program error transmission
send_message(mess_err)

Note:
1. This application note only sends a message when an error occurs.
Add error processes as necessary.
7.10 Sample Program Size Receive Function

```
rev_size(void)

1 byte received?

Yes (ri_u0c1 = 1)

rev_size ← u0rb & 0x00ff

1 byte received?

Yes (ri_u0c1 = 1)

rev_size |= u0rb & 0x00ff

Store size upper data.

Shift stored data 8 bits to the left.

Store size lower data and match to upper data.

return (rev_size)
```
### 7.11 Sample Program Data Receive Function

```c
rev_data(REV_BUFF *buff,
unsigned short *size)

result ← FAIL
Initialize receive results.

sum ← 0
Initialize sum value.

Receive data remaining and
the record size not received

No

Yes

1 byte received?
No (ri_u0c1 = 0)

Yes (ri_u0c1 = 1)

buff -> prg_data[i]
← u0rb & 0x00ff
Store received data.

sum ← sum + (buff -> prg_data[i])
Add sum value.

*size ← *size - 1
Decrement the receive size.

1 byte received?
No (ri_u0c1 = 0)

Yes (ri_u0c1 = 1)

buff -> rev_sum
← u0rb & 0x00ff
Receive sum value (lower data).

1 byte received?
No (ri_u0c1 = 0)

Yes (ri_u0c1 = 1)

buff -> rev_sum
← (buff -> rev_sum ) | ((u0rb & 0x00ff) << 8)
Combine the received sum value (upper data) and the sum value (lower data).

Was the data stored in the record size prg_data[i]?
No (i < RECORD_SIZE)

Yes (i = RECORD_SIZE)

buff -> prg_data[i] ← 0xff
If the received data is less than 256 bytes, the remaining space is filled with FFh.

Sum value is equal
No

Yes (sum = (buff -> rev_sum))

result ← COMPLETE

return (result)
```
### 7.12 Sample Program Execute Function

```c
note_program_start(void)

p ← BLOCK_PGROM2 Store 10000h address (program ROM 2 start address) in p.

Execute write program

("p")

return
```

### 7.13 Message Send Function

- `send_message(const unsigned char *mess)`

1. (*mess = '0') All character strings sent? (Yes)
2. (*mess != '0') Is there data in the send buffer? (No)

   - Yes (ti_u0c1 = 0)
     - Wait until no data remains in the U0TB register.
   - No (ti_u0c1 = 1)
     - u0tb ← *mess Send 1 byte.

return

### 7.14 Write Control Program Transfer Function

```c
send_to_ram(void)

size ← dummy - block_erase_ew0 + 1 Calculate the size of the function to be transferred.

asm("PUSHC FLG") Save to the flag register.

asm("fclr I") Maskable interrupt disabled.

RAM transfer process

asm_smovf(1) Transfer block erase function, program function and full status check function to RAM area.

asm("POPC FLG") Recover the flag register.

return
```

- Note:
  1. ((void far *) block_erase_ew0, (void near *)ram_p, ((unsigned short) size / 2))
7.15 Interrupt Handler for RAM Transfer Function

send_to_ram_vector(void)

size ← dummy2 - ram_int_dummy + 1

asm( "PUSHC FLG" )

asm( "FCLR I" )

Transfer interrupt handler used on the RAM.

asm_smovf(1)

asm( "POPC FLG" )

return

Calculate the size of the function to be transferred.

Save to the flag register.

Disable maskable interrupt.

Recover the flag register.

Note:
1. ((void far *)ram_int_dummy, (void near *)ram_vector, ((unsigned short)size / 2))

7.16 Relocatable Vector Table for RAM Create Function

renewal_of_ram_vector_t(void)

i ← 0

Initialize counter.

Yes (i ≥ 64)

Create relocatable
vector for RAM completed?

No (i < 64)

Set the address of the interrupt handler for the RAM to the relocatable vector table for the RAM.

i ← i + 1

Increase counter.
### 7.17 Block Erase Function

```c
block_erase_ew0(unsigned short far* ers_addr)
asm( "PUSHC FLG" )  // Save to the flag register.
asm( "FCLR I" )     // Disable maskable interrupt.

fmr01 ← 0
Enable CPU rewrite mode.

fmr01 ← 1

fmr1 ← 0x82
Enable FMR6 register write.

fmr6 ← 0x02
Set to EW0 mode.

fmr1 ← 0x80
Enable FMR6 register write.

*ers_addr ← ERS_CMD
Write 0020h in the block highest (even-numbered) address.

*ers_addr ← CNF_CMD
Write 00D0h in the block highest (even-numbered) address.

Currently erasing?

Yes (fmr00 = 0)

No (fmr00 = 1)

Full status check
full_status_check()

Erased correctly?

Yes
(erase_result = COMPLETE)

*ers_addr ← CLR_STS_CMD

(erase_result = CMD_SEQ_ERR)

*ers_addr ← CLR_STS_CMD

(erase_result = PROGRAM_ERR)

*ers_addr ← CLR_STS_CMD

(erase_result = ERASE_ERR)

*ers_addr ← READ_CMD
Execute the read array command.

fmr01 ← 0
Disable CPU rewrite mode.

asm( "POPC FLG" )  // Recover the flag register.

return(erase_result)
```

Note:
1. CPU clock frequencies that can be used in CPU rewrite mode (EW0 and EW1 modes) differ for each product. Refer to the user's manual for details.
7.18 Program Function

```c
program_write_ew0(unsigned short far* write_addr, unsigned short *buff) {
    asm("PUSHC FLG")
    asm("FCLR I")
    fmr01 ← 0
    fmr01 ← 1
    fmr1 ← 0x82
    fmr6 ← 0x02
    fmr1 ← 0x80
    Enable CPU rewrite mode.
    Enable FMR6 register write.
    Set to EW0 mode.
    Disable FMR6 register write.

    256 bytes written?
    No (i < (RECORD_SIZE / 2))
    Yes (i ≥ (RECORD_SIZE / 2))

    write_addr[i] ← PRG_CMD
    Write 0041h in write address.
    write_addr[i] ← buff[i]
    Write data.
    write_addr[i] ← buff[i+1]
    Write data.

    Currently writing?
    Yes (fmr00 = 0)
    No (fmr00 = 1)

    Full status check
    full_status_check()

    Written correctly?
    Yes (program_result = COMPLETE)
    No

    i ← i + 2

    (program_result = CMD_SEQ_ERR)
    *write addr ← CLR_STS_CMD
    (program_result = PROGRAM_ERR)
    *write addr ← CLR_STS_CMD
    (program_result = ERASE_ERR)
    *write addr ← CLR_STS_CMD

    write_addr[0] ← READ_CMD
    Execute the read array command.
    fmr01 ← 0
    Disable CPU rewrite mode.
    asm("POPC FLG")
    Recover the flag register.
    return(program_result)
}
```

Note:
1. The frequency of the CPU clock used for the CPU rewrite mode (EW0 and EW1 modes) may be different depending on MCUs. Refer to the User’s manual for details.
7.19 Full Status Check Function

```
full_status_check(void)
```

```
FMR07 = 1 and FMR06 = 1?
```

- Yes: Command sequence error
- No: Yes

```
FMR07 = 1?
```

- Yes: Erase error
- No: No

```
FMR06 = 1?
```

- Yes: Program error
- No: No

```
return (check_result)
```

7.20 RAM Transfer Process Function

```
asm_smovf (void _far * _source, void _near * _dest, unsigned int _size)
```

- [Argument]
  - * _source: source function start address (R2R0)
  - * _dest: destination RAM area start address (A1)
  - _size: transfer data size (by word) (R3)

```
asm( "pushm R1,A0" )
```

Save to R1 register and A0 register.

```
asm( "mov.w R0,A0" )
```

Transfer the lower 16 bits of the source address (R0) to the A0 register.

```
asm( "mov.w R2,R1" )
```

Transfer the upper 4 bits of the source address (R2) to the R1 register.

```
asm( "movb R1L,R1H" )
```

Transfer the upper 4 bits of the source address (R1L) to the R1H register.

```
asm( "smovf.w" )
```

Transfer to RAM area.

```
asm( "popm R1,A0" )
```

Recover R1 register and A0 register.

```
return
```
8. Sample Program
A sample program can be downloaded from the Renesas Electronics website.
To download, click “Application Notes” in the left-hand side menu of the M16C Family page.

9. Reference Documents
M16C/63 Group User’s Manual: Hardware Rev.1.00
M16C/64 Group User’s Manual: Hardware Rev.1.05
M16C/64A Group User’s Manual: Hardware Rev.1.10
M16C/64C Group User’s Manual: Hardware Rev.0.10
M16C/65 Group User’s Manual: Hardware Rev.1.10
M16C/65C Group User’s Manual: Hardware Rev.0.10
M16C/66C Group User’s Manual: Hardware Rev.1.00
M16C/5LD Group, M16C/56D Group User’s Manual: Hardware Rev.1.10
M16C/5L Group, M16C/56 Group User’s Manual: Hardware Rev.1.00
M16C/5M Group, M16C/57 Group User’s Manual: Hardware Rev.1.01
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

C Compiler Manual
M16C Series, R8C Family C Compiler Package V.5.45
C Compiler User’s Manual Rev.2.00
The latest version can be downloaded from the Renesas Electronics website.

Website and Support
Renesas Electronics website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep. 02, 2009</td>
<td>—</td>
<td>First edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Oct. 14, 2009</td>
<td>27, 28</td>
<td>The processing of the read array command execution is added</td>
</tr>
<tr>
<td>1.02</td>
<td>Jan. 29, 2010</td>
<td>26</td>
<td>Updated the flowchart of &quot;7.14 Write Control Program Transfer Function&quot;</td>
</tr>
<tr>
<td>1.03</td>
<td>Jan. 31, 2011</td>
<td>—</td>
<td>M16C/5L, M16C/5LD, M16C/5M, M16C/64C, M16C/65C, M16C/65D, M16C/56D, M16C/56, and M16C/57 Groups are added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Partially modified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>The condition of M16C/65 Group can be used is added in &quot;2. Introduction&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>The explanation in “Figure 3.1 Settings and Resetting of EW0 Mode” is devised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>The explanation in “Figure 4.1 System Structure Diagram” is added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>The processing of “Transfer and set relocatable vector for RAM” command is added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Function table of Interrupt handler for RAM transfer function is added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Function table of Relocatable vector table for RAM create function is added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Function table of Interrupt handler for RAM of dummy function is added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21</td>
<td>The processing of Interrupt handler for RAM transfer function, Relocatable vector table for RAM create function, and Save to the INTB register commands are added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
<td>The processing of Interrupt handler for RAM transfer function, Relocatable vector table for RAM create function, and Save to the INTB register commands are added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>In &quot;7.14 Write Control Program Transfer Function&quot;: The processing of Save to the flag register and Recover to the flag register commands are added The handler of Interrupt enabled command is deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26</td>
<td>&quot;7.15 Interrupt Handler for RAM Transfer Function&quot; and &quot;7.16 Relocatable Vector Table for RAM Create Function” are added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27</td>
<td>Note 1 is added in &quot;7.17 Block Erase Function&quot; The processing of Save to the flag register, Disable maskable interrupt, and Recover to the flag register commands are added Devised from “fmr0 ← 0x00” to “fmr01 ← 0” Devised from “fmr0 ← 0x02” to “fmr01 ← 1”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
<td>Note 1 is added in &quot;7.18 Program Function” The processing of Save to the flag register, Disable maskable interrupt, and Recover to the flag register commands are added Devised from “fmr0 ← 0x00” to “fmr01 ← 0” Devised from “fmr0 ← 0x02” to “fmr01 ← 1”</td>
</tr>
</tbody>
</table>

All trademarks and registered trademarks are the property of their respective owners.
General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   - In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   - In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products
   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.
   - The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.
Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application example. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacturer, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.

6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

7. Renesas Electronics products are classified according to the following three quality grades: “Standard,” “High Quality,” and “Specific.” The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failures at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of reconfigurable software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics products" means any product developed or manufactured by or for Renesas Electronics.