H8S Family
Rewriting Flash Memory in User Program Mode Using Asynchronous Serial Communication

Introduction
Data to be rewritten in the flash memory on the master side is written to the flash memory on the slave side. Data to be rewritten is transferred using asynchronous serial communication.

Target Device
H8S/2268

Contents
1. Specifications .................................................................................................................................... 2
2. Applicable Conditions ....................................................................................................................... 3
3. Detailed Specifications ...................................................................................................................... 4
4. Principles of Operation ...................................................................................................................... 8
5. Sequence Diagram ............................................................................................................................ 15
6. Slave Main Program ......................................................................................................................... 20
7. Programming/Erasing Control Program on Slave Side .................................................................... 24
8. Asynchronous Serial Communication Program ............................................................................... 53
9. Program Listings ............................................................................................................................. 65
1. Specifications

(1) The user program mode is used to rewrite flash memory.
(2) The slave flash memory is programmed with the rewrite data in master flash memory.
(3) The rewrite data is transferred using asynchronous serial communication on SCI channel 0 (SCI_0).
(4) The flash memory rewrite start command is sent from the master to the slave side when switch 0 (SW0) on the master side is turned on, and rewriting of the slave flash memory begins.
(5) On both the master and slave sides LED1 is off and LED2 is lit during the flash memory rewrite operation, and LED1 is lit and LED2 is off after flash memory rewrite completes.
(6) The IRQ0 pin is connected to switch 0 (SW0) on the master side.
(7) Output ports are connected to LED1 and LED2 on the master side.
(8) On the slave side, LED1 is connected to output pin P10 and LED2 to output pin P11.
(9) A configuration example of the on-board rewrite circuit is shown in figure 1.

Figure 1  Configuration Example of On-Board Rewrite Circuit
2. Applicable Conditions

Compile conditions applied in this application note are as follows.

Please note that exact time such as wait may not be given in some cases due to different versions of compiler or how source programs are created.

Therefore, please be sure to check the codes output after compiling.

Table 1 Applicable Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating frequency</td>
<td>Input clock: 10 MHz</td>
</tr>
<tr>
<td></td>
<td>System clock: 10 MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral module clock: 10 MHz</td>
</tr>
<tr>
<td>Operating mode</td>
<td>Mode 7 (MD2 = 1, MD1 = 1, FWE = 0)</td>
</tr>
<tr>
<td>On-board programming board</td>
<td>User programming mode (MD2 = 1, MD1 = 1, FWE = 1)</td>
</tr>
<tr>
<td>C/C++ compiler</td>
<td>Manufactured by Renesas Technology Corp.</td>
</tr>
<tr>
<td></td>
<td>H8S, H8/300 Series C/C++ Compiler Ver.6.01.01</td>
</tr>
<tr>
<td>Compiler options</td>
<td>-cpu=2000a, -code = machinecode, -optimize=1, -regparam=3</td>
</tr>
<tr>
<td></td>
<td>-speed=(register,shift,struct,expression)</td>
</tr>
</tbody>
</table>

Table 2 Section Settings

<table>
<thead>
<tr>
<th>Address</th>
<th>Section Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H'000000</td>
<td>CV1</td>
<td>Reset routine</td>
</tr>
<tr>
<td>H'001000</td>
<td>P</td>
<td>Main program area</td>
</tr>
<tr>
<td>H'000400</td>
<td>PASSCI</td>
<td>Asynchronous serial communications program area</td>
</tr>
<tr>
<td>H'001000</td>
<td>DSMPL1</td>
<td>Sample data table 1</td>
</tr>
<tr>
<td>H'004000</td>
<td>DSMPL2</td>
<td>Sample data table 2</td>
</tr>
<tr>
<td>H'007FF6</td>
<td>DSMPL3</td>
<td>Sample data table 3</td>
</tr>
<tr>
<td>H'008000</td>
<td>PCPYFZRAM</td>
<td>Area in RAM for storage of the program for transferring the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programming/erasure programs</td>
</tr>
<tr>
<td>H'008100</td>
<td>FZTAT</td>
<td>Programming/erasure program area (*)</td>
</tr>
<tr>
<td></td>
<td>PFZTAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DFZTAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FZEND</td>
<td></td>
</tr>
<tr>
<td>H'FFB000</td>
<td>RAM</td>
<td>Destination in RAM for transfer of programming/erasure program (*)</td>
</tr>
<tr>
<td></td>
<td>PRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DRAM</td>
<td></td>
</tr>
</tbody>
</table>

Note: * Specifying ROM-support option

The ROM-support option of the linker must be set if programs are to be transferred from ROM to RAM and executed from RAM. An example of the ROM-support option setting for this sample task is given below.

rom=PFZTAT=PRAM,
DFZTAT=DRAM
3. Detailed Specifications

3.1 On-Board Programming Operation Conditions

- Device: HD64F2268 (H8S/2268F)
- CPU operation: User program mode
- Operating voltage: 3.3 V
- Operating frequency: 10 MHz

3.2 On-Board Programming Mode

- User Program Mode
  It is a prerequisite that the programming/erasing control program, rewrite start command receive program, RAM transfer program, and FWE control determination program be written beforehand to the flash memory of the slave MCU in the boot mode or writer mode.

3.3 Programming Method

- The rewrite data is received from the transfer source and used to rewrite the flash memory.
- Data from the transfer source is transferred by asynchronous serial communication using SCI channel 0 (SCI_0). The master is the transfer source and the slave the transfer destination.
3.4 Flowchart of Rewrite Procedure

![Flowchart of Rewrite Procedure](image)

Figure 2  User Program Mode Rewrite Procedure

3.5 Master-Slave Connection Diagram

![Master-Slave Connection Diagram](image)

Figure 3  Master-Slave Connection Diagram
3.6 Communication Specifications

Table 3 Communication Specifications

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer speed</td>
<td>31,250 bps</td>
</tr>
<tr>
<td>Communication type</td>
<td>Asynchronous serial communication</td>
</tr>
<tr>
<td>Data bits</td>
<td>8</td>
</tr>
<tr>
<td>Stop bits</td>
<td>1</td>
</tr>
<tr>
<td>Parity</td>
<td>No</td>
</tr>
</tbody>
</table>

3.7 Communication Commands

Table 4 Communication Commands

<table>
<thead>
<tr>
<th>Communication Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H'00</td>
<td>Normal transfer (command name: OK command)</td>
</tr>
<tr>
<td>H'01</td>
<td>Transfer error (command name: NG command)</td>
</tr>
<tr>
<td>H'11</td>
<td>Transmit start request</td>
</tr>
<tr>
<td>H'55</td>
<td>Rewrite start command</td>
</tr>
<tr>
<td>H'66</td>
<td>FWE pin setting command</td>
</tr>
<tr>
<td>H'77</td>
<td>Erase command</td>
</tr>
<tr>
<td>H'88</td>
<td>Programming command</td>
</tr>
</tbody>
</table>

3.8 Memory Mapping

The flash memory erase blocks of the H8S/2268F are listed in table 5.

Table 5 Flash Memory Erase Blocks

<table>
<thead>
<tr>
<th>Block (Size)</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>EB0 (4 Kbytes)</td>
<td>H'000000 to H'000FFF</td>
</tr>
<tr>
<td>EB1 (4 Kbytes)</td>
<td>H'001000 to H'001FFF</td>
</tr>
<tr>
<td>EB2 (4 Kbytes)</td>
<td>H'002000 to H'002FFF</td>
</tr>
<tr>
<td>EB3 (4 Kbytes)</td>
<td>H'003000 to H'003FFF</td>
</tr>
<tr>
<td>EB4 (4 Kbytes)</td>
<td>H'004000 to H'004FFF</td>
</tr>
<tr>
<td>EB5 (4 Kbytes)</td>
<td>H'005000 to H'005FFF</td>
</tr>
<tr>
<td>EB6 (4 Kbytes)</td>
<td>H'006000 to H'006FFF</td>
</tr>
<tr>
<td>EB7 (4 Kbytes)</td>
<td>H'007000 to H'007FFF</td>
</tr>
<tr>
<td>EB8 (32 Kbytes)</td>
<td>H'008000 to H'00FFFF</td>
</tr>
<tr>
<td>EB9 (64 Kbytes)</td>
<td>H'010000 to H'01FFFF</td>
</tr>
<tr>
<td>EB10 (64 Kbytes)</td>
<td>H'020000 to H'02FFFF</td>
</tr>
<tr>
<td>EB11 (64 Kbytes)</td>
<td>H'030000 to H'03FFFF</td>
</tr>
</tbody>
</table>
Memory maps during normal operation of the H8S/2268F and during the flash memory rewrite operation are shown in figure 4.

<table>
<thead>
<tr>
<th>Erase block</th>
<th>Memory address</th>
<th>Normal operation</th>
<th>Flash memory rewrite operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EB0</td>
<td>H'0000000</td>
<td>Vector table</td>
<td>Vector table</td>
</tr>
<tr>
<td></td>
<td>H'0001000</td>
<td>Main program</td>
<td>Main program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rewrite start command received</td>
<td></td>
</tr>
<tr>
<td>EB1</td>
<td>H'0010000</td>
<td>Rewrite target area</td>
<td>Rewrite data area</td>
</tr>
<tr>
<td>EB2</td>
<td>H'0020000</td>
<td>• Data table of main program</td>
<td>• Data table received from master side</td>
</tr>
<tr>
<td>EB3</td>
<td>H'0030000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB4</td>
<td>H'0040000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB5</td>
<td>H'0050000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB6</td>
<td>H'0060000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB7</td>
<td>H'0070000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB8</td>
<td>H'0080000</td>
<td>• RAM transfer program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H'0081000</td>
<td>• Programming/erasing control program</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FWE control determination program</td>
<td></td>
</tr>
<tr>
<td>EB9</td>
<td>H'0100000</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>EB10</td>
<td>H'0200000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB11</td>
<td>H'0300000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal flash memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal RAM</td>
<td>H'FFB000</td>
<td>• Programming/erasing control program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H'FFEFBF</td>
<td>• FWE control determination program</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4  Memory Maps (Slave)**
4. Principles of Operation

4.1 Normal Operation

(1) Normally, application accesses the data table in flash memory. The data table is received from the master side and rewritten.

(2) The programming/erasing control program, rewrite start command receive program, RAM transfer program, and FWE control determination program are written beforehand to the slave flash memory.

(3) Data is transferred between the master and slave sides by asynchronous serial communication using SCI channel 0 (SCI_0).

(4) On the slave side LED1 is connected to output pin P10 and LED2 to output pin P11. LED1 and LED2 are off when P10 and P11 are high level. When P10 and P11 are low level LED1 and LED2 light.

![Figure 5 Normal Operation](image-url)
4.2 Preparation for On-Board Rewriting

(1) The rewrite start command H'55 is sent from the master when a rising edge of the IRQ0 signal is detected.
(2) At this point LED1 is off and LED2 is lit on the master.

![Diagram](image-url)

**Figure 6** Preparation for On-Board Rewriting
4.3 Start of On-Board Rewriting

(1) The RAM transfer program is initiated when the slave receives the H'55 command, and the programming/erasing control program is transferred to internal RAM.

(2) At this point LED1 is off and LED2 is lit on the slave.

Figure 7 Start of On-Board Rewriting
4.4 Startup of Programming/Erasing Control Program

(1) After transfer by the RAM transfer program completes, operation branches to the programming/erasing control program stored in RAM.

![Figure 8 Startup of Programming/Erasing Control Program](image)

4.5 Setting of FWE Pin

(1) The FWE pin setting command H'66 is received from the transfer source.
(2) The programming/erasing control program controls PF3 to set the FWE pin to 1.

![Figure 9 Setting of FWE Pin](image)
4.6 Erasing Flash Memory

(1) The erase command H'77 is received from the master.
(2) The programming/erasing control program erases the target block of flash memory.

Figure 10 Erasing Flash Memory
4.7 Programming Flash Memory

(1) The program command H'88 is received from the transfer source.
(2) The programming/erasing control program receives the new data table from the transfer source and writes it to flash memory.
(3) After programming completes LED1 is lit and LED2 is off on both the master and slave sides.

![Diagram of Programming Flash Memory]

Figure 11 Programming Flash Memory
4.8 Clearing the FWE Pin

(1) The programming/erasing control program controls PF3 to clear the FWE pin to 0.

4.9 Initiating the Program

(1) The device is reset and the new application, which accesses the new data table, is initiated.
5. Sequence Diagram

(1) Normal Operation

Figure 14  Normal Operation
(2) Erase Processing

Master

Transmit erase command H'77 and erase block count sequentially

2 bytes

H'77 H'01 to H'0C

Slave

Receive 2 bytes

1st. byte

==H'77?

H'77

Error handling

Receive start address of erase block

FLSHE=1

Is received data a start address?

Start address

Error handling

Erase target block

Erased normally?

Erased normally

End

Erase error

All erase blocks complete?

To next block

To programming position/size receive processing

Receive 1 byte

OK command?

No

OK command

Error handling

Transmit OK[H'00]

Receive start address of erase block

OK command?

No

OK command

Error handling

To programming position/size receive processing

Figure 15 Erase Processing
(3) Programming Position/Size Reception Processing

![Flowchart of Programming Position/Size Reception Processing](image)

**Figure 16** Programming Position/Size Reception Processing
(4) Programming Processing

**Diagram Description:**
- **Master:**
  - Receive 1 byte
  - If byte is H’11, proceed.
  - H’11 received?
    - If yes, proceed.
    - Is programming data to be transmitted 128 bytes or less?
      - If yes, transmit 128 bytes.
      - More than 128 bytes, transmit 128 bytes, and proceed.
    - If no, proceed.
  - All data transmitted?
    - If yes, receive 1 byte.
    - If no, proceed.
  - OK command?
    - If yes, LED1 lit, LED2 off.
    - If no, proceed.
  - LED1 lit

- **Slave:**
  - Transmit programming data transmit request command H’11
  - 1 byte
  - 128 bytes or less
  - More than 128 bytes
  - Receive 128 bytes
  - Program 128 bytes
  - Programmed normally?
    - If yes, all data received and programmed?
      - If yes, receive remaining data.
      - If no, transmit remaining data.
    - If no, to next data.
  - Programming error
  - Error handling
  - LED1 lit
  - LED2 off

**Figure 17** Programming Processing
(5) Error Handling

**Figure 18  Error Handling**
6. Slave Main Program

6.1 Hierarchy

The slave main program, which is run from flash memory, executes the user application programs (main applications), receives rewrite start commands, and transfers the programming/erasing control program from flash memory to internal RAM. The hierarchy of the routines used by the slave main program is shown in figure 19.

![Hierarchy of Slave Main Program](image)

Note: * Called from asynchronous serial communication program.

6.2 List of Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Executes main applications, receives rewrite start commands, transfers</td>
</tr>
<tr>
<td></td>
<td>programming/erasing control program from flash memory to internal RAM</td>
</tr>
<tr>
<td>copyfzram</td>
<td>Transfers programming/erasing control program from flash memory to internal RAM</td>
</tr>
<tr>
<td>FZMAIN</td>
<td>Programming/erasing control program</td>
</tr>
</tbody>
</table>
6.3 Description of Functions

(1) main() Function

(a) Specifications
void main (void)

(b) Principles of Operation
• Executes user application programs (main applications)
• Receives rewrite start commands
• Transfers programming/erasing control program from flash memory to internal RAM
• Branches to programming/erasing control program

(c) Arguments
• Input values: None
• Output values: None

(d) Global Variables
None

(e) Subroutines Used
com_init(): Initializes communication settings
SLrcv1byte(): Receives 1 byte of data
copyfzram(): Transfers programming/erasing control program to internal RAM
FZMAIN(): Branches to programming/erasing control program

(f) Internal Registers Used

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTPCRD</td>
<td></td>
<td>Module stop control register D</td>
<td>H'FFFC60</td>
<td>—</td>
</tr>
<tr>
<td>MSTPD6</td>
<td></td>
<td>Used by sample main application.</td>
<td>Bit 6</td>
<td>—</td>
</tr>
<tr>
<td>P7DDR</td>
<td></td>
<td>Port 7 data direction register</td>
<td>H'FFFFE36</td>
<td>—</td>
</tr>
<tr>
<td>P7DR</td>
<td></td>
<td>Used by sample main application.</td>
<td>H'FFFFF06</td>
<td>—</td>
</tr>
<tr>
<td>P0RT7</td>
<td></td>
<td>Port 7 register</td>
<td>H'FFFFFB6</td>
<td>—</td>
</tr>
<tr>
<td>P70</td>
<td></td>
<td>Used by sample main application.</td>
<td>Bit 0</td>
<td>—</td>
</tr>
<tr>
<td>P1DDR</td>
<td></td>
<td>Port 1 data direction register</td>
<td>H'FFFFE30</td>
<td>H'03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• P1DDR = H'03: P11 and P10 set as output pins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1DR</td>
<td></td>
<td>Port 1 data register</td>
<td>H'FFFFF00</td>
<td>—</td>
</tr>
<tr>
<td>P11DR</td>
<td></td>
<td>Port 11 data register</td>
<td>Bit 1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• P11DR = 0: P11 output level low</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• P11DR = 1: P11 output level high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10DR</td>
<td></td>
<td>Port 10 data register</td>
<td>Bit 0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• P10DR = 0: P10 output level low</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• P10DR = 1: P10 output level high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSR_0</td>
<td></td>
<td>Serial status register_0</td>
<td>H'FFFF7C</td>
<td>—</td>
</tr>
<tr>
<td>RDRF</td>
<td></td>
<td>Receive data register full</td>
<td>Bit 6</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RDRF = 0: No received data stored in RDR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RDRF = 1: Received data stored in RDR_0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(g) Flowchart

main()

Main application

com_init()
Initialize communications settings

Main application

RDRF == 1?
Received?

RDRF=0

RDRF=1

tmp = SLrcv1byte()
Receive 1 byte

tmp = 0

tmp = H'55?
Evaluate received data?

= H'55

P10DR = 1 (LED1 off)
P11DR = 0 (LED2 lit)

Copyzram()
RAM transfer program

FZMAIN()
Branches to programming/
erasing control program in internal RAM

Note: * Main applications are sample programs that are not related to flash memory programming. In this example such main applications are specified in the two locations indicated above.
(2) copyfzram() Function
   (a) Specifications
       void copyfzram (void)
   (b) Principles of Operation
       Transfers the flash memory programming/erasing control program to internal RAM
   (c) Arguments
       - Input values: None
       - Output values: None
   (d) Global Variables
       None
   (e) Subroutines Used
       None
   (f) Internal Registers Used
       None
   (g) Flowchart

   (3) FZMAIN() Function
       Calls the main routine of the programming/erasing control program.
7. Programming/Erasing Control Program on Slave Side

7.1 Hierarchy

The programming/erasing control program erases flash memory in block units, receives flash memory programming data, and programs flash memory. The hierarchy of the routines used by the programming/erasing control program is shown in figure 20. With the exception of the FZMAIN() function, the subroutines used perform either communication processing or flash memory programming/erasing processing.

![Hierarchy of Programming/Erasing Control Program](image)

Note: * Called from asynchronous serial communication program.

7.2 List of Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FZMAIN</td>
<td>Main routine of programming/erasing control program</td>
</tr>
<tr>
<td>fwe_check</td>
<td>Controls and determines state of FWE pin</td>
</tr>
<tr>
<td>blk_check</td>
<td>Determines the bit number of the block to be erased from erase start address</td>
</tr>
<tr>
<td>blk1_erase</td>
<td>Erases designated blocks of flash memory</td>
</tr>
<tr>
<td>ferase</td>
<td>Erases designated blocks</td>
</tr>
<tr>
<td>ferasevf</td>
<td>Verifies erase of designated blocks</td>
</tr>
<tr>
<td>fwrite128</td>
<td>Verifies write of 128 bytes</td>
</tr>
<tr>
<td>fwrite</td>
<td>Writes to target address</td>
</tr>
<tr>
<td>fwritevf</td>
<td>Verifies target address, creates overwrite data</td>
</tr>
</tbody>
</table>
### 7.3 List of Constants

Table 9 List of Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>H’00</td>
<td>Normal return value</td>
</tr>
<tr>
<td>NG</td>
<td>H’01</td>
<td>Error return value</td>
</tr>
<tr>
<td>WNG</td>
<td>H’02</td>
<td>Write error</td>
</tr>
<tr>
<td>MAXBLK1</td>
<td>H’0C</td>
<td>Total number of flash memory blocks (12)</td>
</tr>
<tr>
<td>OW_COUNT</td>
<td>H’06</td>
<td>Overwrite count</td>
</tr>
<tr>
<td>WLOOP1</td>
<td>$1 \times \text{MHZ/KEISU1} + 1 = 4$ (H’04)</td>
<td>WAIT statement execution count, 1-µs WAIT</td>
</tr>
<tr>
<td>WLOOP2</td>
<td>$2 \times \text{MHZ/KEISU1} + 1 = 7$ (H’07)</td>
<td>WAIT statement execution count, 2-µs WAIT</td>
</tr>
<tr>
<td>WLOOP4</td>
<td>$4 \times \text{MHZ/KEISU1} + 1 = 14$ (H’0E)</td>
<td>WAIT statement execution count, 4-µs WAIT</td>
</tr>
<tr>
<td>WLOOP5</td>
<td>$5 \times \text{MHZ/KEISU1} + 1 = 17$ (H’11)</td>
<td>WAIT statement execution count, 5-µs WAIT</td>
</tr>
<tr>
<td>WLOOP10</td>
<td>$10 \times \text{MHZ/KEISU1} + 1 = 34$ (H’22)</td>
<td>WAIT statement execution count, 10-µs WAIT</td>
</tr>
<tr>
<td>WLOOP20</td>
<td>$20 \times \text{MHZ/KEISU1} + 1 = 67$ (H’43)</td>
<td>WAIT statement execution count, 20-µs WAIT</td>
</tr>
<tr>
<td>WLOOP50</td>
<td>$50 \times \text{MHZ/KEISU1} + 1 = 167$ (H’A7)</td>
<td>WAIT statement execution count, 50-µs WAIT</td>
</tr>
<tr>
<td>WLOOP100</td>
<td>$100 \times \text{MHZ/KEISU1} + 1 = 334$ (H’14E)</td>
<td>WAIT statement execution count, 100-µs WAIT</td>
</tr>
<tr>
<td>TIME10</td>
<td>$10 \times \text{MHZ/KEISU1} + 1 = 34$ (H’22)</td>
<td>WAIT statement execution count, 10-µs WAIT</td>
</tr>
<tr>
<td>TIME30</td>
<td>$30 \times \text{MHZ/KEISU1} + 1 = 101$ (H’65)</td>
<td>WAIT statement execution count, 30-µs WAIT</td>
</tr>
<tr>
<td>TIME200</td>
<td>$200 \times \text{MHZ/KEISU1} + 1 = 667$ (H’29B)</td>
<td>WAIT statement execution count, 200-µs WAIT</td>
</tr>
<tr>
<td>TIME10000</td>
<td>$(10000/\text{KEISU1}) \times \text{MHZ} + 1 = 33334$ (H’8236)</td>
<td>WAIT statement execution count, 10-ms WAIT</td>
</tr>
</tbody>
</table>

Note: MHZ: Operating frequency of 10 MHz
      KEISU1: Minimum number of state per loop in for statements.

### 7.4 RAM Usage

The stack memory used by the FZMAIN function is listed in table 10. Additional stack memory is used for program operation, but the precise amount differs depending on factors such as the version of the compiler used and the option settings.

Table 10 RAM Usage

<table>
<thead>
<tr>
<th>Data</th>
<th>Stack Memory Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming data</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Overwrite data</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Additional programming data</td>
<td>128 bytes</td>
</tr>
</tbody>
</table>
7.5 Description of Functions

(1) FZMAIN() Function
   (a) Specifications
      void FZMAIN(void)
   (b) Principles of Operation
      • Controls and determines state of FWE pin
      • Erases flash memory
      • Receives flash memory programming data
      • Programs flash memory
      • Start by reset after programming completes
   (c) Arguments
      • Input values: None
      • Output values: None
   (d) Global Variables
      None
   (e) Subroutines Used
      fwe_check(): Controls and determines the state of FWE pin
      rcv1byte(): Receives 1 byte of data
      rcvnbyte(): Receives n bytes of data
      trs1byte(): Transmits 1 byte of data
      fwrite128(): Writes 128 bytes, verifies write
      blk_check(): Determines bit number of the block to be erased from the erase start address
      blk1_erase(): Erases designated blocks of flash memory
### (f) Internal Registers Used

#### Table 11 Registers Used by FZMAIN() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1DR</td>
<td>Port 1 data register</td>
<td>H'FFFF00 —</td>
<td>Bit 1 1</td>
<td></td>
</tr>
<tr>
<td>P11DR</td>
<td>Port 11 data register</td>
<td>Bit 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10DR</td>
<td>Port 10 data register</td>
<td>Bit 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCSR_0</td>
<td>Timer control/status register_0</td>
<td>H'FFFF74 H'00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVF</td>
<td>Overflow flag</td>
<td>Bit 7 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WT/IT</td>
<td>Timer mode select</td>
<td>Bit 6 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TME</td>
<td>Timer enable</td>
<td>Bit 5 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKS2</td>
<td>Clock select 2 to 0</td>
<td>Bit 2 CKS2 = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKS1</td>
<td>CKS2 = 0, CKS1 = 0, CKS0 = 0: (\phi/2) clock input selected for TCNT_0</td>
<td>Bit 1 CKS1 = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKS0</td>
<td>CKS2 = 0, CKS1 = 0, CKS0 = 0: (\phi/2) clock input selected for TCNT_0</td>
<td>Bit 0 CKS0 = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCNT_0</td>
<td>Timer counter</td>
<td>H'FFFF74 H'FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSTCSR</td>
<td>Reset control/status register</td>
<td>H'FFFF76 H'5F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSTE</td>
<td>Reset enable</td>
<td>Bit 6 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 11 Registers Used by FZMAIN() Function (cont.)

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLPWCR</td>
<td>PDWND</td>
<td>Flash memory power control register</td>
<td>H'FFFFAC</td>
<td>H'80</td>
</tr>
</tbody>
</table>

- **PDWND**
  - PDWND = 0: Transition to power-down modes for flash memory enabled
  - PDWND = 1: Transition to power-down modes for flash memory disabled

Notes:
1. The method for writing to TCSR_0 is different from that for general registers.
   - Writing is accomplished by word transfer with H'FFFF74 as the target.
   - The value of the upper byte is H'A5 and the lower byte is the programming data.
   - In this function, the value written is as follows:
     \[
     \text{TCSR}_0 = H'A500
     \]

2. The method for writing to TCNT_0 is different from that for general registers.
   - Writing is accomplished by word transfer with H'FFFF74 as the target.
   - The value of the upper byte is H'5A and the lower byte is the programming data.
   - In this function, the value written is as follows:
     \[
     \text{TCNT}_0 = H'5AFF
     \]

3. The method for writing to RSTCSR is different from that for general registers.
   - Writing is accomplished by word transfer with H'FFFF76 as the target.
   - When writing to the RSTE bit, the value of the upper byte is H'5A and the lower byte is the programming data.
   - In this function, the value written is as follows:
     \[
     \text{RSTCSR} = H'5A5F
     \]
(g) Flowcharts

```
FZMAIN()

trs1byte()  Transmit OK

tmp=rcv1byte()  Receive 1 byte

/tmp≠H'66?
 非H'66?

FLPWCR = H'80
Disable transition to power-down modes for flash memory

fwe_check()  Set to FWE pin

trs1byte()  Transmit OK

RSTCSR = H'5A5F
Set to reset the MCU when watchdog timer (WDT) overflow occurs

TCSR_0 = H'A500
Halt WDT

ERRCASE
Error handling
```
Rewriting Flash Memory in User Program Mode
Using Asynchronous Serial Communication

1. \( \text{rcvndt}[] \leftarrow \text{rcvnbyte}() \)
   Receive 2 bytes
   \( \text{rcvndt}[0] \): Erase command H'77
   \( \text{rcvndt}[1] \): Total number of blocks to be erased

   \( \text{rcvndt}[0] \neq \text{H'77} \)
   1st. byte \( \neq \text{H'77} \)?

   \( = \text{H'77} \)

   ERRCASE
   Error handling

   \( \text{trs1byte}() \)
   Transmit OK

   \( \text{tmp} \leftarrow \text{rcvndt}[1] \times 4 \)

   E_ADDR[] ← \text{rcvnbyte}()
   Receive \( \text{tmp} \) bytes of erase start address

   \( i = 0 \)

2. \( i < \text{rcvndt}[1] \)
   Erase blocks complete?

   Yes, specified number of blocks all erased

   \( \text{trnsbyte}() \)
   Transmit OK

   2

3. \( i++ \)

4. \( \text{rtn} \neq \text{OK} \)
   \( \text{rtn} = \text{blk1_erase}() \)
   Erase 1 block

5. ERRCASE
   Error handling

   \( \text{rtn} = \text{OK} \)
   Normal erase

   \( i++ \)
ERRCASE

Error handling

tmp ← rcv1byte()
Receive 1 byte

tmp = H'88?
Received data ≠ H'88?

= H'88

ERRCASE
Error handling

trs1byte()
Transmit OK

rcv.wdt ← rcvncbyte()
Write start address
Write size
Receive total 8 bytes

Lower 7 bits of received
write start address ≠ 0?

≠ 0

ERRCASE
Error handling

All 0

rcv.lw.restsize = 0?

= H'0000

ERRCASE
Error handling

> H'0000

ERRCASE
Error handling

trs1byte()
Transmit OK

5
Rewriting Flash Memory in User Program Mode Using Asynchronous Serial Communication

5

rcv.lw.restsize ≠ 0
Programming data remaining?

rcv.lw.restsize = 0
Programming complete

Remaining programming data 128 bytes or less?

Yes

memset()
Copy H'FF to 128 bytes from start of W_BUF[]

rcv.lw.restsize = 0
rcv.lw.restsize -= 128

rtn=fwrite128()
Write 128 bytes of data

rtn=OK
Normal write

rcv.lw.ad_tmp += 128
Move flash memory write address 128 bytes

More than 128 bytes

rcv.lw.restsize=0
Programming complete

memset()
Copy H'FF to 128 bytes from start of W_BUF[]

rcv.lw.restsize=0
Programming data remaining?

rcv.lw.restsize ≠ 0
Programming data remaining?

trs1byte()
Transmit H'11 (transmit start request)

rtn=OK
Normal write

ERRCASE
Error handling

P10DR = 0 (LED1 lit)
P11DR = 1 (LED2 off)
Set and startup watchdog timer (WDT) to perform reset
TCNT=H'5AFF
TCSR=H'5A578
Reset start

rtn=OK
Write error

rtn ≠ OK?
Write error?

rcvnbyte()
Receive 128 bytes of data in W_BUF[]

rcv.lw.restsize = 0
rcv.lw.restsize -= 128

rtn=fwrite128()
Write 128 bytes of data

rtn=OK
Normal write

rcv.lw.restsize ≠ 0
Programming data remaining?
(2) fwe_check() Function
(a) Specifications
void fwe_check(void)
(b) Principles of Operation
• Controls and determines the state of FWE pin
(c) Arguments
None
(d) Global Variables
None
(e) Subroutines Used
None
(f) Internal Registers Used

<table>
<thead>
<tr>
<th>Table 12 Registers Used by fwe_check() Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register</strong></td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>SCRX</td>
</tr>
<tr>
<td>FLSHE</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>IER</td>
</tr>
<tr>
<td>IRQ3E</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>PFDDR</td>
</tr>
<tr>
<td>PF3DDR</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Table 12 Registers Used by `fwe_check()` Function (cont.)

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMER</td>
<td>RAM simulation register</td>
<td>H'FFFFEDB</td>
<td>H'00</td>
<td></td>
</tr>
<tr>
<td>RAMS</td>
<td>RAM select</td>
<td>Bit 3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RAMS = 0: Disables RAM emulation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RAMS = 1: Enables RAM emulation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFDR</td>
<td>Port F data register</td>
<td>H'FFFF0E</td>
<td>H'08</td>
<td></td>
</tr>
<tr>
<td>PF3DR</td>
<td>Port F3 data register</td>
<td>Bit 3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PF3DR = 0: PF3 output level low</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PF3DR = 1: PF3 output level high</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCR</td>
<td>A/D control register</td>
<td>H'FFFF99</td>
<td>H'00</td>
<td></td>
</tr>
<tr>
<td>TRGS1</td>
<td>Timer trigger select 1 and 0</td>
<td>Bit 7</td>
<td>TRGS1 = 0</td>
<td></td>
</tr>
<tr>
<td>TRGS0</td>
<td>TRGS1 = 0, TRGS0 = 0: Starts A/D conversion by software</td>
<td>Bit 6</td>
<td>TRGS0 = 0</td>
<td></td>
</tr>
<tr>
<td>FLMCR1</td>
<td>Flash memory control register 1</td>
<td>H'FFFFEA8</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>FWE</td>
<td>Flash write enable</td>
<td>Bit 7</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FWE = 0: FWE input pin outputs low level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FWE = 1: FWE input pin outputs high level</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(g) Flowchart

```
(fwe_check()
  IER=H'00
  Disable IRQ3 interrupts
  ADCR=H'00
  Start A/D conversion by software
  PFDRR=H'08
  Set PF3 to output pin
  PFDR=H'08
  Set PF3 to 1 and FWE to 1
  SCRX=H'08
  Set FLSHE to 1 to enable access to FLMCR1, FLMCR2, EBR1, and EBR2
  RAMER=H'00
  Disable RAM emulation
  tmp = FWE
  Read FWE input pin
  tmp==0?
  Is value of FWE 1?
    =0
    =1
  END
```
(3) blk1_erase() Function
(a) Specifications
char blk1_erase(
   unsigned long ers_ad,
   unsigned char ET_COUNT
)
(b) Principles of Operation
• Determines the bit number of the block to be erased from the erase start address
• Erases designated blocks of flash memory
(c) Arguments
• Input values:
   ers_ad: Erase start address
   ET_COUNT: Maximum erase count
• Output values:
   Return value: Result flag (OK = H'00, NG = H'01)
(d) Global Variables
   None
(e) Subroutines Used
   blk_check(): Determines the bit number of the block to be erased from the erase start address
   ferase(): Erases designated blocks
   ferasevf(): Verifies erase of designated blocks
(f) Internal Registers Used

Table 13 Registers Used by blk1_erase() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLMCR1</td>
<td></td>
<td>Flash memory control register 1</td>
<td>H'FFFFFA8</td>
<td>—</td>
</tr>
<tr>
<td>SWE1</td>
<td></td>
<td>Software write enable</td>
<td>Bit 6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SWE1 = 0: Disable flash memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>programming/erasing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SWE1 = 1: Enables flash memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>programming/erasing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(g) Flowchart

```
blk1_erase()

rtn = blk_check()
  Confirm received erase address E_ADR[] and fetch erase start and end addresses and erase block number

rtn==OK?
  Address is correct

  Set SWE1 bit in FLMCR1 to 1

  1µs elapsed?
    Yes

    Initial verify
    rtn = ferasevf();

    i = 0

  No

  i < ET_COUNT?
    Yes

  rtn==OK?
    rtn=OK

    Clear SWE1 bit in FLMCR1 to 0

    rtn=OK

  No

  100µs elapsed?
    Yes

return(rtn)

END
```

(4) blk_check() Function

(a) Specifications

```c
char blk_check(
    unsigned long eck_ad,
    unsigned long *eck_st,
    unsigned long *eck_ed,
    unsigned char *blk_no
)
```

(b) Principles of Operation

- Determines the bit number of the block to be erased from the erase start address
- Determines if received erase start address is correct by comparison with BLOCKADR[] and returns the result flag, erase start address, erase end address, and bit numbers of the erase target blocks

(c) Arguments

- Input values:
  - eck_ad: Erase start address
  - *eck_st: Verified erase start address
  - *eck_ed: Verified erase end address
  - *blk_no: Bit number of the erase target block
- Output values:
  - Return values: Result flag (OK = H'00, NG = H'01)
  - *eck_st: Verified erase start address
  - *eck_ed: Verified erase end address
  - *blk_no: Bit number of the erase target block

(d) Global Variables

- BLOCKADR[]: Stores the start addresses of blocks in flash memory

```c
unsigned long BLOCKADR[13] ={
    /* Erase Block Address */
    H'000000, /* EB0 4KBYTE */
    H'001000, /* EB1 4KBYTE */
    H'002000, /* EB2 4KBYTE */
    H'003000, /* EB3 4KBYTE */
    H'004000, /* EB4 4KBYTE */
    H'005000, /* EB5 4KBYTE */
    H'006000, /* EB6 4KBYTE */
    H'007000, /* EB7 4KBYTE */
    H'008000, /* EB8 32KBYTE */
    H'010000, /* EB9 64KBYTE */
    H'020000, /* EB10 64KBYTE */
    H'030000, /* EB11 64KBYTE */
    H'040000, /* End Block Address */
};
```

(e) Subroutines Used

- None

(f) Internal Registers Used

- None
(g) Flowchart

```
<table>
<thead>
<tr>
<th>blk_check()</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>eck_ad ≠ BLOCKADR[i]?</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>Store bit number of erase target block</td>
</tr>
<tr>
<td>*blk_no=i</td>
</tr>
<tr>
<td>Store erase start address</td>
</tr>
<tr>
<td>*eck_st = BLOCKADR[i]</td>
</tr>
<tr>
<td>i++</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Store erase end address</td>
</tr>
<tr>
<td>*eck_ed = BLOCKADR[i]-1</td>
</tr>
<tr>
<td>return(OK)</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>i &gt; MAXBLK1?</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>i++</td>
</tr>
<tr>
<td>return(NG)</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>
```

(5) ferase() Function

(a) Specifications

```
void ferase(unsigned char e_blk_no)
```

(b) Principles of Operation

- Erases a designated block in flash memory

(c) Arguments

- Input values:
  - e_blk_no: Erase target block number
- Output values:
  - None

(d) Global Variables

- None

(e) Subroutines Used

- None

(f) Internal Registers Used
### Table 14 Registers Used by ferase() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLMCR1</td>
<td>ESU1</td>
<td>Erase setup</td>
<td>H'FFFFA8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESU1 = 0: Clears erase setup state</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESU1 = 1 when FWE1 = 1 and SWE1 = 1: Enters erase setup state</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E1</td>
<td>Erase</td>
<td>Bit 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E1 = 0: Clears the erase mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>E1 = 1 when SWE1 = 1 and ESU1 = 1: Enters the erase mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBR1</td>
<td>EB7</td>
<td>Erase block register 1</td>
<td>H'FFFFFAA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Setting a bit from EB7 to EB0 to 1 enables erasing of the corresponding block of flash memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBR2</td>
<td>EB11</td>
<td>Erase block register 2</td>
<td>H'FFFFFAB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EB10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EB9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EB8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCSR_0*1</td>
<td>OVF</td>
<td>Overflow flag</td>
<td>H'FFFF74</td>
<td>H'7F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OVF = 0: No TCNT_0 overflow</td>
<td>And 7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OVF = 1: TCNT_0 overflow</td>
<td>And 7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>WT/IT</td>
<td>Timer mode select</td>
<td>Bit 6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WT/IT = 0: Interval timer</td>
<td>And 6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WT/IT = 1: Watchdog timer</td>
<td>And 6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>TME</td>
<td>Timer enable</td>
<td>Bit 5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TME = 0: TCNT_0 count start</td>
<td>And 5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TME = 1: TCNT_0 count halt</td>
<td>And 5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CKS2</td>
<td>Clock select 2 to 0</td>
<td>Bit 2</td>
<td>CKS2 = 1</td>
</tr>
<tr>
<td></td>
<td>CKS1</td>
<td></td>
<td>Bit 1</td>
<td>CKS1 = 1</td>
</tr>
<tr>
<td></td>
<td>CKS0</td>
<td></td>
<td>Bit 0</td>
<td>CKS0 = 1</td>
</tr>
<tr>
<td>Notes:</td>
<td></td>
<td>*1. The method for writing to TCSR_0 is different from that for general registers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Writing is accomplished by word transfer with H'FFFF74 as the target.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value of the upper byte is H'A5 and the lower byte is the programming data.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In this function, the value written is as follows:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCSR_0 = H'A57F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(g) Flowchart

```
ferase()

tmp=1

e_blk_no<8
  Determine block to be erased?

  blk_no<8
    Block to be erased is among EB0 to EB7
    blk_no = e_blk_no - 8
    EBR1 = (tmp << e_blk_no)
    Set erase block in EBR1 to 1

  blk_no>=8
    Block to be erased is among EB8 to EB11
    e_blk_no<8
    Determine block to be erased?

TCSR_0 = H'A500
  Set and start watchdog timer_0
  Operating clock: \( \phi/131,072 \)

Set ESU1 bit in FLMCR1 to 1

No
  100 \( \mu \)s elapsed?

  Yes
    E1 = 1
    Set E1 bit in FLMCR1 to 1

  No
    10 ms elapsed?

    Yes
      E1 = 0
      Clear E1 bit in FLMCR1 to 0

    No
      10 \( \mu \)s elapsed?

      Yes
        Clear ESU1 bit in FLMCR1 to 0

      No
        10 \( \mu \)s elapsed?

        Yes
          TCSR_0 = H'A500
          Halt watchdog timer_0

        No
          EBR1 = 0
          EBR2 = 0

  END
```
(6) ferasevf() Function

(a) Specifications

```c
char ferasevf(
    unsigned short *evf_st,
    unsigned short *evf_ed
)
```

(b) Principles of Operation

- Verifies erase of designated blocks in flash memory

(c) Arguments

- Input values:
  - `evf_st`: Erase start address
  - `evf_ed`: Erase end address
- Output values:
  - Return value: Result flag (OK = H'00, NG = H'01)

(d) Global Variables

None

(e) Subroutines Used

None

(f) Internal Registers Used

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLMCR1</td>
<td>EV1</td>
<td>Erase verify</td>
<td>H'FFFFA8</td>
<td>—</td>
</tr>
<tr>
<td>EV1</td>
<td></td>
<td></td>
<td>Bit 3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EV1 = 0: Cancels the erase verify mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EV1 = 0: Enters the erase verify mode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(g) Flowcharts

ferasevf ()

- Set EV1 bit in FLMCR1 to 1
- i = 0
- i++
- i < WLOOP20
- Elapsed time < 20 µs?

Yes

rtn = OK

ead = *evf_st
- Set verify start address in ead
- j = 0

< evf_ed
- Determine end address

Yes

≥ evf_ed
No

eae[j*2] = H’FF
- Dummy write H’FF to verify address

i = 0
j++

i++
Yes

j = 0

< evf_ed
- Determine end address

Yes

≥ evf_ed
No

i < WLOOP2
- Elapsed time < 2 µs?

No

rtn = NG

evf_st[j] ≠ H’FFFF

= H’FFFF

rtn = NG
(7) fwrite128() Function

(a) Specifications
char fwrite128(
    unsigned char *wt_buf,
    unsigned char *wt_adr,
    unsigned short WT_COUNT
)

(b) Principles of Operation
- Programs and verifies 128 bytes of data

(c) Arguments
- Input values:
  *wt_adr: Write address
  *wt_buf: 128 bytes of programming data
  WT_COUNT: Maximum number of writes
- Output values:
  Return value: Result flag (OK = H'00, NG = H'01)
  *wt_adr: Write address
  *wt_buf: 128 bytes of programming data

(d) Global Variables
None

(e) Subroutines Used
fwrite: Writes to the target address
fwritevf: Verifies the target address, creates overwrite data
Table 16  Registers Used by fwrite128() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLMCR1</td>
<td></td>
<td>Flash memory control register 1</td>
<td>H'FFFFA8</td>
<td>—</td>
</tr>
<tr>
<td>SWE1</td>
<td></td>
<td>Software write enable</td>
<td>Bit 6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SWE1 = 0</td>
<td>Disable flash memory programming/erasing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWE1 = 1</td>
<td>Enable flash memory programming/erasing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rewriting Flash Memory in User Program Mode
Using Asynchronous Serial Communication

(g) Flowchart

1. `fwrite128()`
   - Copy 128 bytes from `w_buf[]` to `BUFF[]`
   - Set SWE1 bit in FLMCR1 to 1
   - If not NG, go to 1 µs elapsed?

2. Initial programming verify
   - `rtn = fwritevf()`
   - If `rtn == NG`, set `rtn=NG` and go to 1

3. Set programming wait time
   - `TM = TIME30`
   - `j = 0`

4. Initial programming
   - `fwrite(TM)`
   - `rtn = fwritevf()`
   - If `j < WT_COUNT`, go to 1
   - If `j >= WT_COUNT`, go to 6

5. Initial programming verify
   - `fwrite(TM)`
   - `rtn = fwritevf()`
   - If `j < OW_COUNT`, go to 1
   - If `j >= OW_COUNT`, go to 10

6. `fwrite(TIME10)`
   - Clear SWE1 bit in FLMCR1 to 0
   - If `rtn != NG`, go to 1
   - `j++`

7. 100 µs elapsed?
   - If not NG, return `rtn`

8. END
(8) fwrite() Function
   (a) Specifications
       void fwrite(
           unsigned char *buf,
           unsigned char *w_adr,
           unsigned char ptime
       )
   (b) Principles of Operation
       Writes to target address
   (c) Arguments
       • Input values:
           *buf: Write start address (overwrite data or additional programming data)
           *w_adr: Write address
           ptime: Setting time for the P1 bit (10 µs, 30 µs, or 2,000 µs)
       • Output values:
           None
   (d) Global Variables
       None
   (e) Subroutines Used
       None
   (f) Internal Registers Used
### Table 17 Registers Used by fwrite() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLMCR1</td>
<td></td>
<td>Flash memory control register 1</td>
<td>H’FFFFA8</td>
<td>—</td>
</tr>
<tr>
<td>PSU1</td>
<td></td>
<td>Program setup</td>
<td>Bit 4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>• PSU1 = 0: Program setup canceled</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PSU1 = 1: Transition to program setup state</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td>Program</td>
<td>Bit 0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>• P1 = 0: Cancels the program mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• P1 = 1 when SWE1 = 1 and PSU1 = 1: Enters the program mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCSR_0</td>
<td></td>
<td>Timer control/status register 0</td>
<td>H’FFFF74</td>
<td>H’79</td>
</tr>
<tr>
<td></td>
<td>OVF</td>
<td>Overflow flag</td>
<td>Bit 7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• OVF = 0: No TCNT_0 overflow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• OVF = 1: TCNT_0 overflow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WT/IT</td>
<td>Timer mode select</td>
<td>Bit 6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>• WT/IT = 0: Interval timer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• WT/IT = 1: Watchdog timer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TME</td>
<td>Timer enable</td>
<td>Bit 5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>• TME = 0: TCNT_0 count start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• TME = 1: TCNT_0 count halt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CKS2</td>
<td>Clock select 2 to 0</td>
<td>Bit 2</td>
<td>CKS2 = 0</td>
</tr>
<tr>
<td></td>
<td>CKS1</td>
<td>CKS2 = 0, CKS1 = 0, CKS0 = 1: φ/64 clock input</td>
<td>Bit 1</td>
<td>CKS1 = 0</td>
</tr>
<tr>
<td></td>
<td>CKS0</td>
<td>selected for TCNT_0</td>
<td>Bit 0</td>
<td>CKS0 = 1</td>
</tr>
</tbody>
</table>

Notes:*1. The method for writing to TCSR_0 is different from that for general registers.
- Writing is accomplished by word transfer with H‘FFFF74 as the target.
- The value of the upper byte is H’A5 and the lower byte is the programming data.
- In this function, the value written is as follows:
  TCSR_0 = H’A579
Rewriting Flash Memory in User Program Mode Using Asynchronous Serial Communication

(g) Flowchart

1. fwrite()
2. \( i = 0 \)
3. \( i < 128 \)
4. Is total programming data < 128 bytes?
5. Yes
6. No
7. End programming
8. \( w_{\text{adr}}[i] = \text{buf}[i] \)
9. Write 1 byte of overwrite address or additional programming address to write address
10. \( i++ \)
11. TCSR_0 = H'A579
12. Set and start watchdog timer_0
13. Operating clock: \( \phi/64 \)
14. Set PSU1 bit in FLMCR1 to 1
15. \( 50 \mu s \) elapsed?
16. Yes
17. No
18. P1=1
19. Set P1 bit in FLMCR1 to 1
20. \( \text{ptime} (10,30,200 \mu s) \)
21. WAIT?
22. Yes
23. No
24. P1=0
25. Clear P1 bit in FLMCR1 to 0
26. \( 5 \mu s \) elapsed?
27. Yes
28. No
29. Clear PSU1 bit in FLMCR1 to 0
30. \( 5 \mu s \) elapsed?
31. Yes
32. No
33. TCSR_0 = H'A500
34. Halt watchdog timer_0
35. END
(9) fwritevf() Function

(a) Specifications

```c
char fwritevf(
    unsigned short *owbuff,
    unsigned short *buff,
    unsigned short *wvf_buf,
    unsigned short *wvf_adr
)
```

(b) Principles of Operation
- Verifies target address and creates overwrite data

(c) Arguments
- Input values:
  * `owbuff`: 128 bytes of additional programming data
  * `buff`: 128 bytes of overwrite data
  * `wvf_buf`: 128 bytes of programming data
  * `wvf_adr`: Write address
- Output values:
  * `owbuff`: 128 bytes of additional programming data
  * `buff`: 128 bytes of overwrite data
  * `wvf_buf`: 128 bytes of programming data
  * `wvf_adr`: Write address

(d) Global Variables
- None

(e) Subroutines Used
- None

(f) Internal Registers Used

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLMC1</td>
<td></td>
<td>Flash memory control register 1</td>
<td>H'FFFFA8</td>
<td>—</td>
</tr>
<tr>
<td>PV1</td>
<td>Program-verify</td>
<td></td>
<td>Bit 2</td>
<td>1</td>
</tr>
</tbody>
</table>
- PV1 = 0: Cancels the program-verify mode
- PV1 = 1: Enters the program-verify mode
Rewriting Flash Memory in User Program Mode
Using Asynchronous Serial Communication

(g) Flowcharts

fwritev()
Set PV1 bit in FLMCR1 to 1

i=0

j++
Yes

i<WLOOP4
Elapsed time<4\mu s?

No

wad = *wvf_adr
Set verify address in wad

j = 0

j≥128/2
Verify end

j<128/2
Verifying of 128 bytes completed?

j<128/2

wad[*2] = H'FF
Dummy write H'FF to verify address

i=0

i++

i<WLOOP2
Elapsed time<2\mu s?

No

owbuff[j]=buff[j]|wvf_adr[j]
Calculate additional programming data
Verify data and overwrite data OR condition

buff[j]=~wvf_adr[j]|wvf_buf[j]
Calculate overwrite data
Verify data NOT condition and programming data OR condition

tmp = ~wvf_adr[j]&wvf_buf[j]
Error check

tmp≠0?
tmp=0

tmp≠0

1
Clear PV1 bit in FLMCR1 to 0

\[ i = 0 \]

\[ j = 0 \]

\[ j \geq 128/2 \]

\[ \text{tmp} = 0 \]

\[ \text{tmp} = 0 \]

\[ \text{rtn} = \text{WNG} \]

\[ \text{rtn} = \text{OK} \]

\[ j = 0 \]

\[ \text{rtn} = \text{NG} \]

\[ \text{buff}[j] \neq \text{H'FFFF} \]

\[ \text{rtn} = \text{NG} \]

\[ \text{tmp} = 0? \]

\[ \text{tmp} = 0 \]

\[ \text{rtn} = \text{OK} \]

\[ j = 0 \]

\[ j < 128/2 \]

\[ j \geq 128/2 \]

\[ \text{j++} \]

\[ \text{buff}[j] \neq \text{H'FFFF} \]

\[ \text{rtn} = \text{NG} \]

\[ \text{rtn} = \text{OK} \]

\[ j = 0 \]

\[ \text{rtn} = \text{WNG} \]

return(rtn)

END
8. Asynchronous Serial Communication Program

8.1 Hierarchy
The asynchronous serial communication program performs processing of communications with the master side.

![Hierarchy of Asynchronous Serial Communication Program](image)

8.2 List of Functions

<table>
<thead>
<tr>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>com_init</td>
<td>Initializes asynchronous serial communication</td>
</tr>
<tr>
<td>rcv1byte</td>
<td>Receives 1 byte of data</td>
</tr>
<tr>
<td>rcvnbtye</td>
<td>Receives n bytes of data</td>
</tr>
<tr>
<td>trs1byte</td>
<td>Transmits 1 byte of data</td>
</tr>
<tr>
<td>trsnbyte</td>
<td>Transmits n bytes of data</td>
</tr>
</tbody>
</table>

8.3 Description of Functions

(1) com_init() Function
(a) Specifications
   void com_init(void)
(b) Principles of Operation
   - Initializes asynchronous serial communication
(c) Arguments
   - Input values: None
   - Output values: None
(d) Global Variables
   None
(e) Subroutines Used
   None
(f) Internal Registers Used
### Table 20 Registers Used by com_init() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTPCRB</td>
<td>Module stop control register B</td>
<td></td>
<td>H'FFFFDE9</td>
<td>H'7F</td>
</tr>
</tbody>
</table>
| MSTPB7     | Serial communication interface 0 | • MSTPB7 = 0: Clear module stop mode for SCI_0  
• MSTPB7 = 1: Enter module stop mode for SCI_0 |          |           |
| SMR_0      | Serial mode register 0 |                                                                                   | H'FFFF78 | H'00      |
| C/ A       | Communication mode | • C/ A = 0: Asynchronous communication mode  
• C/ A = 1: Clock synchronous communication mode | Bit 7    | 0         |
| CHR        | Character length | • CHR = 0: 8-bit data length selected for asynchronous communication mode  
• CHR = 1: 7-bit data length selected for asynchronous communication mode | Bit 6    | 0         |
| PE         | Parity enable  | • PE = 0: Disables appending and checking of parity bits during transmission in asynchronous communication mode  
• PE = 1: Enables appending and checking of parity bits during transmission in asynchronous communication mode | Bit 5    | 0         |
| O/ E       | Parity mode   | • O/ E = 0: Even parity for appending and checking of parity bits  
• O/ E = 1: Odd parity for appending and checking of parity bits | Bit 4    | 0         |
| STOP       | Stop bit length | • STOP = 0: Stop bit length of 1 bit selected for asynchronous communication mode  
• STOP = 1: Stop bit length of 2 bits selected for asynchronous communication mode | Bit 3    | 0         |
| MP         | Multiprocessor mode | • MP = 0: Disables multiprocessor communications function  
• MP = 1: Enables multiprocessor communications function | Bit 2    | 0         |
| CKS1       | Clock select 1 and 0 | • CKS1 = 0, CKS0 = 0: \(\phi\) clock selected as clock source for internal baud rate generator | Bit 1    | CKS1 = 0  |
| CKS0       | Clock select 1 and 0 |                                                                                   | Bit 0    | CKS0 = 0  |
Table 20 Registers Used by com_init() Function (cont.)

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRR_0</td>
<td>Bit rate register 0</td>
<td>BRR = H’09: Selects 31250-bps transmit bit rate matching operating clock selected by CKS1 and CKS0 in SMR_0</td>
<td>H’FFFF79</td>
<td>H’09</td>
</tr>
<tr>
<td>SCR_0</td>
<td>Serial control register</td>
<td>TE = 0: Disables transmit operation</td>
<td>Bit 5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TE = 1: Enables transmit operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RE = 0: Disables receive operation</td>
<td>Bit 4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RE = 1: Enables receive operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKE1</td>
<td>Clock enable 1 and 0</td>
<td>CKE1 = 0, CKE0 = 0: Selects internal clock as clock source in asynchronous communication mode and sets SCK0 as I/O port</td>
<td>Bit 1</td>
<td>CKE1 = 0</td>
</tr>
<tr>
<td>CKE0</td>
<td></td>
<td>CKE0 = 0</td>
<td>Bit 0</td>
<td>CKE0 = 0</td>
</tr>
<tr>
<td>SSR_0</td>
<td>Serial status register 0</td>
<td>TDRE = 0: Transmit data written to TDR_0 has not been transferred to TSR_0</td>
<td>Bit 7</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TDRE = 1: Transmit data has not been written to TDR_0 or transmit data written to TDR_0 has been transferred to TSR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RDRF = 0: No received data stored in RDR_0</td>
<td>Bit 6</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RDRF = 1: Received data stored in RDR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORER</td>
<td>Overrun error</td>
<td>ORER = 0: Indicates reception is in progress or has completed</td>
<td>Bit 5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Framing error</td>
<td>ORER = 1: Indicates an overrun error occurred during reception</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FER</td>
<td>Framing error</td>
<td>FER = 0: Indicates reception is in progress or has completed</td>
<td>Bit 4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FER = 1: Indicates a framing error occurred during reception</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PER</td>
<td>Parity error</td>
<td>PER_0 = 0: Indicates reception is in progress or has completed</td>
<td>Bit 3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PER_0 = 1: Indicates a parity error occurred during reception</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEND</td>
<td>Transmit end</td>
<td>TEND_0 = 0: Indicates transmission is in progress</td>
<td>Bit 2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TEND_0 = 1: Indicates transmission has ended</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>Bit Name</td>
<td>Description</td>
<td>Address</td>
<td>Set Value</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>SCMR</td>
<td>SMIF</td>
<td>Smart card interface mode select</td>
<td>H'FFFF7E</td>
<td>H'F2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SMIF = 0: Normal asynchronous mode or clock synchronous mode</td>
<td>Bit 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SMIF = 1: Smart card interface mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEMR_0</td>
<td>ABCS</td>
<td>Serial expansion mode register 0</td>
<td>H'FFFDF8</td>
<td>H'00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Asynchronous basic clock select</td>
<td>Bit 3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ABCS = 0: Frequency of basic clock for 1-bit interval is 16 times the transfer rate in asynchronous mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ABCS = 1: Frequency of basic clock for 1-bit interval is 8 times the transfer rate in asynchronous mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACS2</td>
<td></td>
<td>Asynchronous clock source select</td>
<td>Bit 2</td>
<td>ACS2=0</td>
</tr>
<tr>
<td>ACS1</td>
<td></td>
<td>ACS2 = 0, ACS1 = 0, ACS0 = 0: External clock</td>
<td>Bit 1</td>
<td>ACS1=0</td>
</tr>
<tr>
<td>ACS0</td>
<td></td>
<td>input selected as asynchronous clock source</td>
<td>Bit 0</td>
<td>ACS0=0</td>
</tr>
</tbody>
</table>
(g) Flowchart

com_init()

MSTPCRB &= H'7F
Cancel module stop mode for SCI_0

SCR_0 &= H'CF
Disable transmission and reception

SCR_0 &= H'FC
Set internal clock as clock source

SMR_0 = H'00
Asynchronous mode
8-bit data length
No parity
1 stop bit
Internal baud rate generator
= \phi \text{ clock}

SCMR = H'F2
Operate in asynchronous mode or clock synchronous mode

SEMR_0 = H'00
Set frequency of basic clock for 1-bit interval to 16 times the transfer rate

BRR_0 = 7
Set transfer speed to 38400 bps

i = 0

i \geq 270?
Wait 1-bit interval(26.04\mu s) or more

\text{i<270}

Clear ORER, FER, and PER in SSR_0 to 0

SCR_0 = H'30
Enable transmission and reception

END
(2) rcv1byte() Function
   (a) Specifications
      unsigned char rcv1byte(void)
   (b) Principles of Operation
      Receives 1 byte of asynchronous serial data
   (c) Arguments
      • Input values: None
      • Output values: 1 byte received data
   (d) Global Variables
      None
   (e) Subroutines Used
      None
   (f) Internal Registers Used

Table 21 Registers Used by rcv1byte() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSR_0</td>
<td>RDRF</td>
<td>Receive data register full</td>
<td>Bit 6</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>ORER</td>
<td>Overrun error</td>
<td>Bit 5</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>FER</td>
<td>Framing error</td>
<td>Bit 4</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>PER</td>
<td>Parity error</td>
<td>Bit 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>RDR_0</td>
<td>Receive data register 0</td>
<td>Bit 6</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-bit register that stores received data</td>
<td>H'FFFF7D</td>
<td>—</td>
</tr>
</tbody>
</table>
rcv1byte()

tmp = RDRF

SSR&=H’38
ORER or FER or PER =1?

No

Yes

ORER or FER or PER =1

No

tmp = 0

Yes

tmp == 0 ?

No

tmp = RDR_0
Write received data

RDRF = 0
Clear RDRF flag

return(tmp(received data))

END
(3) rcvnbyte() Function
   (a) Specifications
       void rcvnbyte(
               unsigned char *ram
               unsigned char dtno,
       )
   (b) Principles of Operation
       Receives n bytes of asynchronous serial data
   (c) Arguments
       • Input values:
         *ram: RAM start address for storing received data
         dtno: Number of bytes of received data
       • Output values: 1-byte received data
         *ram: received data
   (d) Global Variables
       None
   (e) Subroutines Used
       rcv1byte: Receives 1 byte of asynchronous serial data
   (f) Internal Registers Used
       None
   (g) Flowchart

```
rcvnbyte()

   dtno=0 ?
   Finished to specified byte?

   dtno=0

   *ram = rcv1byte()
   Receive 1 byte and copy to RAM

   *ram ++

   dtno --

   END
```
(4) trs1byte() Function
(a) Specifications
   void trs1byte(unsigned char tdt)
(b) Principles of Operation
   Transmits 1 byte of asynchronous serial data
(c) Arguments
   • Input values:
     tdt: 1-byte transmit data
   • Output values: None
(d) Global Variables
   None
(e) Subroutines Used
   None
(f) Internal Registers Used

Table 22 Registers Used by trs1byte() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDR_0</td>
<td>Transmit data register 0</td>
<td>8-bit register that stores transmit data</td>
<td>H'FFFF7B</td>
<td>—</td>
</tr>
<tr>
<td>SSR_0</td>
<td>Serial status register 0</td>
<td></td>
<td>H'FFFF7C</td>
<td>—</td>
</tr>
</tbody>
</table>
| TDRE     | Transmit data register empty | • TDRE = 0: Indicates transmit data written to TDR_0 has not been transferred to TSR_0
|          |               | • TDRE = 1: Indicates transmit data has not been written to TDR_0 or transmit data written to TDR_0 has been transferred to TSR_0 | Bit 7     | —         |
| TEND     | Transmit end   | • TEND = 0: Indicates transmission is in progress
|          |               | • TEND = 1: Indicates transmission has ended     | Bit 2     | —         |
(g) Flowchart

trs1byte()

Yes, data remains in TDR_0

TDRE==0?

No, data is not in TDR_0

TDR_0 = tdt

TDRE = 0

Yes, transmission is in progress

TEND==0?

No, transmission ended

END
(5) trsnbyte() Function

(a) Specifications

void trsnbyte(unsigned char *tdt, unsigned char dtno)

(b) Principles of Operation

Transmits n bytes of asynchronous serial data

(c) Arguments

- Input values:
  * tdt: Start address of transmit data
  dtno: Size of transmission

- Output values: None

(d) Global Variables

None

(e) Subroutines Used

None

(f) Internal Registers Used

Table 23 Registers Used by trsnbyte() Function

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Name</th>
<th>Description</th>
<th>Address</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDR_0</td>
<td></td>
<td>Transmit data register 0</td>
<td>H'FFFFF7B</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 8-bit register that stores transmit data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSR_0</td>
<td></td>
<td>Serial status register 0</td>
<td>H'FFFFF7C</td>
<td>—</td>
</tr>
<tr>
<td>TDRE</td>
<td></td>
<td>Transmit data register empty</td>
<td>Bit 7</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TDRE = 0: Indicates transmit data written to TDR_0 has not been transferred to TSR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TDRE = 1: Indicates transmit data has not been written to TDR_0, or transmit data written to TDR_0 has been transferred to TSR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEND</td>
<td></td>
<td>Transmit end</td>
<td>Bit 2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TEND = 0: Indicates transmission is in progress</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TEND = 1: Indicates transmission has ended</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(g) Flowchart

```
trsnbyte()

dttno=0 ?
Finished to specified
byte?

Yes, data remains in TDR
No, data is not in TDR

TDRE==0?

TDR_0 = *tdt
TDRE = 0
*ttdt++
dtno --

TEND==0?

Yes, transmission is in progress
No, transmission ended

END
```
9. Program Listings

9.1 Slave Main Program

/****************************************************************************
 /*               */
 /*  H8S/2268F            */
 /*  Flash Memory Write/Erase Application Note    */
 /*               */
 /*  Communication Interface        */
 /*  : Asynchronous Serial Interface      */
 /*  Function            */
 /*  : Slave Main Program     */
 /*               */
 /*  External Clock  : 10MHz       */
 /*  Internal Clock  : 10MHz       */
 /*  Sub Clock   : 32.768kHz      */
 /*               */
/****************************************************************************
#include <machine.h>
#include "string.h"

/******************************************************************************/
/*  Symbol Definition          */
/******************************************************************************/
struct BIT {
  unsigned char b7:1;    /* bit7 */
  unsigned char b6:1;    /* bit6 */
  unsigned char b5:1;    /* bit5 */
  unsigned char b4:1;    /* bit4 */
  unsigned char b3:1;    /* bit3 */
  unsigned char b2:1;    /* bit2 */
  unsigned char b1:1;    /* bit1 */
  unsigned char b0:1;    /* bit0 */
};
#define SSR_0_BIT    (*(volatile struct BIT *)0xFFFF7C)   /* Serial Status Register    */
#define RDRF_0    SSR_0_BIT.b6        /* Receive Data Register Full   */
#define LPCR     *(volatile unsigned char *)0xFFFC30  /* LCD Port Control Register   */
#define LCR     *(volatile unsigned char *)0xFFFC31  /* LCD Control Register    */
#define LCR2     *(volatile unsigned char *)0xFFFC32  /* LCD Control Register 2    */
#define LCDRAM    (volatile unsigned char *)0xFFFC4A   /* LCD RAM       */
#define MSTPCRD    *(volatile unsigned char *)0xFFFC60  /* Module Stop Control Registers D  */
#define P1DDR    *(volatile unsigned char *)0xFFFFE30  /* Port 1 Data Direction Register */
#define P1DR     *(volatile unsigned char *)0xFFFFE30  /* Port 1 Data Register    */
#define P1DR_BIT  *(volatile struct BIT *)0xFFFFE30     /* Port 1 Data Register    */
#define P11DR     *(volatile unsigned char *)0xFFFFE30  /* Port 1 Data Register    */
#define P10DR     *(volatile unsigned char *)0xFFFFE30  /* Port 10       */
#define P7DDR    *(volatile unsigned char *)0xFFFFE30  /* Port 7 data direction register */
#define P7DR     *(volatile unsigned char *)0xFFFFE30  /* Port 7 data register    */
#define PORT7     *(volatile unsigned char *)0xFFFFE30  /* Port 7 register      */
#define PORT7_BIT  *(volatile struct BIT *)0xFFFFE30    /* Port 7 register      */
#define PORT7 b0  PORT7_BIT.b0      /* Port 70       */
REJ06B0484-0200/Rev.2.00 August 2006 Page 66 of 79

H8S Family
Rewriting Flash Memory in User Program Mode
Using Asynchronous Serial Communication

/**************************************************************************
* Function define
**************************************************************************/
extern void FZMAIN( void );
void main( void );
void copyfzram( void );
extern void com_init( void );
extern void transbyte( unsigned char *tdt, unsigned char dtno );
extern unsigned char SAMPLEDT1[10]; /* 0x001000 - 0x001005 Sample Data */
extern unsigned char SAMPLEDT2[10]; /* 0x005000 - 0x005005 Sample Data */
extern unsigned char SAMPLEDT3[10]; /* 0x02FFAA - 0x02FFFF Sample Data */

/**************************************************************************
* Vector Address
**************************************************************************/
#pragma section V1          /* VECTOR SECTOIN SET */
void (*const VEC_TBL1[])(void) = {
  main              /*  0 0  R e s e t */
};
#pragma entry main(sp=0x00FFEFC0)
#pragma section           /* P */
/* Main Program */

/**************************************************************************
* Main Program
**************************************************************************/
void main( void )
{
  unsigned char  tmp;
  unsigned char  tmp2;
  unsigned char  swcnt;

  set_ccr(0x80);
  set_exr(0x00);
  MSTPCRD = 0xBF; /* module stop mode is cleared */
  P7DDR = 0xF0;
  P7DR = 0xE0;
  P1DDR = 0xFF;
  P1DR = 0xFF;
  com_init(); /* Communication Initialize */
  swcnt = 0; /* User Application Program Sample */
  do{
    if(swcnt == 1){
      transbyte(SAMPLEDT1[0], 10);
    }
    else if(swcnt == 2){
      transbyte(SAMPLEDT2[0], 10);
    }
    else if(swcnt == 3){
      transbyte(SAMPLEDT3[0], 10);
    }
    swcnt++;
    if(swcnt > 3){
      swcnt = 1;
    }

    do{
      tmp2 = P70;
      tmp = RDRF_0;
      tmp2 = tmp2&(~tmp);
        }while(tmp2);
    if(tmp != 0) /* Data Receive? */
      tmp = rcvbyte();
  }
}
while(tmp != 0x55);          /* Flash Memory Erase/Write Start?  */
/*-------- Flash Memory Write Mode -------------*/
P1DDR = 0x03;
P10DR = 1;           /* LED1 OFF          */
P11DR = 0;           /* LED2 ON          */
copyfzram();

FZMAIN();            /* Flash Memory Write Main Program     */
}

#pragma section CPYFZRAM   /* VECTOR SECTION SET        */
/************************************************************/
/* Copy FZTAT to RAM          */
/************************************************************/
void copyfzram ( void )
{
    char *X_BGN;
    char *X_END;
    char *Y_BGN;
    X_BGN = __sectop("FZTAT");        /* Flash , Ram Address Copy      */
    X_END = __secend("FZTAT");
    Y_BGN = __sectop("RAM");
    memcpy(Y_BGN, X_BGN, X_END-X_BGN);       /* Flash -> RAM Copy        */
}
9.2 Slave Programming/Erasing Control Program

/******************************************/
/*                */
/*  H8S/2268F           */
/*  Flash Memory Write/Erase Application Note    */
/*                */
/*  Communication Interface        */
/*  : Asynchronous Serial Interface      */
/*  Function            */
/*  : Slave Flash Memory Write/Erase Control Program  */
/*                */
/*  External Clock : 10MHz */
/*  Internal Clock : 10MHz */
/*  Sub Clock : 32.768kHz */
/*                */
/******************************************/
#pragma section FZTAT
#include <machine.h>
#include "string.h"

/******************************************/
/*  Symbol Definition          */
/******************************************/

struct BIT {
  unsigned char b7:1;    /* bit7 */
  unsigned char b6:1;    /* bit6 */
  unsigned char b5:1;    /* bit5 */
  unsigned char b4:1;    /* bit4 */
  unsigned char b3:1;    /* bit3 */
  unsigned char b2:1;    /* bit2 */
  unsigned char b1:1;    /* bit1 */
  unsigned char b0:1;    /* bit0 */
};
#define FLMCR1   *(volatile unsigned char *)0xFFFFA8  /* Flash Memory Control Register 1   */
#define FLMCR1_BIT  (*(volatile struct BIT *)0xFFFFA8)   /* Flash Memory Control Register 1   */
#define FWE     FLMCR1_BIT.b7       /* Flash Write Enable      */
#define SWE1    FLMCR1_BIT.b6       /* Software Write Enable     */
#define ESU1    FLMCR1_BIT.b5       /* Erase Setup       */
#define PSU1    FLMCR1_BIT.b4       /* Program Setup       */
#define EV1    FLMCR1_BIT.b3       /* Erase Verify       */
#define PV1    FLMCR1_BIT.b2       /* Program Verify       */
#define E1    FLMCR1_BIT.b1       /* Erase         */
#define P1    FLMCR1_BIT.b0       /* Program        */
#define FLMCR2   *(volatile unsigned char *)0xFFFFA9  /* Flash Memory Control Register 2   */
#define FLMCR2_BIT  (*(volatile struct BIT *)0xFFFFA9)   /* Flash Memory Control Register 2   */
#define FLER    FLMCR2_BIT.b7       /* FLER         */
#define EBR1    *(volatile unsigned char *)0xFFFFAA  /* Erase Block Register 1     */
#define EBR2    *(volatile unsigned char *)0xFFFFAB  /* Erase Block Register 2     */
#define RAMER   *(volatile unsigned char *)0xFFFFEDB  /* RAM Emulation Register     */
#define FLMCRCR  *(volatile unsigned char *)0xFFFFFAC  /* Flash Memory Power Control Register */
#define SCRX    *(volatile unsigned char *)0xFFFFDB4  /* Serial Control Register X    */
#define SCRX_BIT  (*(volatile struct BIT *)0xFFFFDB4)   /* Serial Control Register X    */
#define FLWCKE   SCRX_BIT.b3        /* Flash Memory Control Register Enable */
#define TCSRW_0   *(volatile unsigned short *)0xFFFF74  /* Timer Control/Status Register W   */
#define TCMRW_0   *(volatile unsigned short *)0xFFFF74  /* Timer Counter W      */
#define RSTCSR   *(volatile unsigned short *)0xFFFF76  /* Timer Control/Status Register W   */
#define PIDCR    *(volatile unsigned char *)0xFFFFE30  /* Port 1 Data Direction Register */
#define PIDR     *(volatile unsigned char *)0xFFFFF00  /* Port 1 Data Register */
#define PIDR_BIT  (*(volatile struct BIT *)0xFFFFF00)    /* Port 1 Data Register */
#define P11DR    PIDR_BIT.b1        /* Port 11        */
#define P10DR    PIDR_BIT.b0        /* Port 10        */
#define IER     *(volatile unsigned char *)0xFFFFE14  /* IRQ Enable Register */
#define ADCR    *(volatile unsigned char *)0xFFFFF99  /* A/D Control Register */
#define FFDI    *(volatile unsigned char *)0xFFFFE3E  /* Port F Data Direction Register */
#define FFDR    *(volatile unsigned char *)0xFFFFF0E  /* Port F Data Register */
/** Function define */
/*/ Function define */
void FZMAIN ( void );
void fwe_check ( void );
char blk1_erase ( unsigned long ers_ad, unsigned char ET_COUNT );
char blk_check ( unsigned long eck_ad, unsigned long *eck_st, unsigned long *eck_ed, unsigned char *blk_no );
void ferase ( unsigned char e_blk_no );
char fwrite128 ( unsigned char *wt_buf, unsigned char *wt_adr, unsigned short WT_COUNT );
void fwrite ( unsigned char *buf, unsigned char *w_adr, unsigned short ptime );
char fwritevf ( unsigned short *owbuff, unsigned short *buff, unsigned short *wvf_buf, unsigned short *wvf_adr );
extern unsigned char rcv1byte ( void );
extern void rcvnbyte ( unsigned char *ram, unsigned char dtno );
extern void trs1byte ( unsigned char tdt );

 ITE WAIT TIME *****************************/
#define MHZ 10 /* 20MHZ */
#define KEISU1 3 /* 1Loop 3Step <-- DEC.B(1)+BNE(2) */
#define KEISU2 5 /* 1Loop 5Step <-- INC.W(1)+CMP.W(2)+BCS(2) */
#define WLOOP1 1*MHZ/KEISU1+1 /* LOOP WAIT TIME */
#define WLOOP2 2*MHZ/KEISU1+1
#define WLOOP4 4*MHZ/KEISU1+1
#define WLOOP5 5*MHZ/KEISU1+1
#define WLOOP10 10*MHZ/KEISU1+1
#define WLOOP20 20*MHZ/KEISU1+1
#define WLOOP50 50*MHZ/KEISU2+1
#define WLOOP100 100*MHZ/KEISU2+1
#define TIME10 10*MHZ/KEISU1+1 /* WRITE WAIT TIME */
#define TIME30 30*MHZ/KEISU1+1 /* WRITE WAIT TIME */
#define TIME200 200*MHZ/KEISU2+1 /* WRITE WAIT TIME */
#define TIME10000 10000*MHZ/KEISU2+1 /* ERASE WAIT TIME */

***** Fixed number definition *****************************/
unsigned long BLOCKADR[13] ={
0x000000, /* Erase Block Address */
0x000100, /* EB0 4KBYTE */
0x000200, /* EB1 4KBYTE */
0x000300, /* EB2 4KBYTE */
0x000400, /* EB3 4KBYTE */
0x000500, /* EB4 4KBYTE */
0x000600, /* EB5 4KBYTE */
0x000700, /* EB6 4KBYTE */
0x000800, /* EB7 4KBYTE */
0x000900, /* EB8 32KBYTE */
0x000a00, /* EB9 64KBYTE */
0x000b00, /* EBl 64KBYTE */
0x000c00, /* End Block Address */
};
#define MAXBLK1 12
#define OK 0
#define NG 1
#define OW_COUNT 6 /* Over Write Count */

**** Flash Memory Write Main Program */
/** Flash Memory Write Main Program */
void FZMAIN ( void )
{    
    char rtn;
    unsigned char i,tmp;
unsigned char    rcvndt[2];
unsigned long    E_ADR[12];
unsigned char    W_BUF[128];    /* Write Data Area        */
union{
    unsigned char wtdt[8];
    struct{
        unsigned long ad_tmp;
        unsigned long restsize;
    }lw;
}rcv;

trs1byte(OK);           /* SEND OF OK Code        */

tmp = rcv1byte();
if(tmp != 0x66)
goto ERRCASE;
FLPWCR = 0x80;           /* flash power-down modes disabled     */
fwe_check();           /* Set FWE */
trs1byte(OK);           /* SEND OF OK Code        */
RSTCSR = 0x5A5F;          /* LSI Reset if WDT overflows      */
TCSRW_0 = 0xA500;          /* WDT STOP          */

/*------- Erase -------------------------------*/
rcvnbyte(rcvndt, 2);         /* RECEIVE ERASE BLOCK NUMBER      */
if(rcvndt[0] != 0x77)         /* Recive Code = 0x77?       */
goto ERRCASE;
trs1byte(OK);           /* SEND OF OK Code        */
tmp = rcvndt[1] << 2;
rcvnbyte((unsigned char*)E_ADR, tmp);     /* Recive ERASE BLOCK Address      */
for(i = 0; i < rcvndt[1]; i++){
    rtn = blk1_erase(E_ADR[i], 3);       /* 1 block Erase         */
    if(rtn != OK)
goto ERRCASE;
}
trs1byte(OK);           /* SEND OF OK Code        */

/*------- Write Address / Size Recive --------*/
tmp = rcv1byte();
if(tmp != 0x88)
goto ERRCASE;
trs1byte(OK);           /* SEND OF OK Code        */
rcvnbyte(rcv.wtdt, 8);         /* Recive Write Top Address & Size     */
if(rcv.wtdt[3] & 0x7F)
goto ERRCASE;
if(rcv.lw.restsize == 0x0000){
goto ERRCASE;
}
trs1byte(OK);           /* SEND OF OK Code        */

/*------- 128 byte Flash Memory Write ---------*/
while(rcv.lw.restsize != 0){
    tmp = trs1byte();          /* SEND OF Request        */
    if(rcv.lw.restsize <= 128){       /* Receive WriteData from HOST      */
        memset(W_BUF,0xFF,128);       /* INITIALIZE RECEIVE BUFFER (0xFF)    */
        rcvnbyte(M_BUF,(unsigned char)rcv.lw.restsize);    /* "restsize" byte Receive */
    }
rcv.lw.restsize = 0;
}
else{
    rcvbyte(W_BUF, 128);  /* 128byte Receive */
    rcv.lw.restsize -= 128;
}

rtn = fwrite128(W_BUF, (unsigned char*)rcv.lw.ad_tmp, 1000);
if(rtn != OK)
goto ERRCASE;
rcv.lw.ad_tmp = rcv.lw.ad_tmp + 128;

trs1byte(OK);       /* SEND OF OK Code */
P10DR = 0;          /* LED1 ON */
P11DR = 1;          /* LED2 OFF */
TCNTW_0 = 0x5AFF;    /* INITIALIZED WDT COUNT */
TCSR7_0 = 0x5A578;  /* WDT START phi/2 */
while(1);          /* OK End */

/---------- Error Case --------------------------*/
ERRCASE:         /* Error Case */
    trs1byte(NG);
    P10DR = 0;          /* LED1 ON */
    P11DR = 0;          /* LED2 ON */
    while(1);
}

******************************************************************************
/* FWE Check */
******************************************************************************
void fwe_check ( void )
{
    unsigned char tmp;
    IER = 0x00;        /* IRQ3 Disable */
    ADCR = 0x00;       /* ADTRG OFF */
    PFDDR = 0x08;      /* PF3 Output Setting */
    PFDR = 0x08;       /* Set PF3 / Set FEW */
    SCRX = 0x08;       /* FLSHE=1 */
    RAMER = 0x00;      /* RAM Emulation Register OFF */
    do{
        tmp = FWE;
    }while(tmp==0);       /* FWE Set? */

******************************************************************************
/* Flash Memory 1 block Erase */
******************************************************************************
char blk1_erase ( unsigned long ers_ad, unsigned char ET_COUNT )
{
    char rtn;
    unsigned char i;
    unsigned short j;
    unsigned char block_no;
    unsigned long ers_st,ers_ed;

    rtn = blk_check(ers_ad,&ers_st,&ers_ed,&block_no);       /* CHECK BLOCK START ADDRESS */
    if(rtn == OK){
        SWE1 = 1;    /* Set the SWE1 bit */
        for(i = 0; i < WLOOP1; i++);
        /* Need to wait 1 usec */
        rtn = ferasevf((unsigned short*)ers_st, /* Erase Verify */
unsigned short *ers_ed);

for(i = 0; i < ET_COUNT; i++) {
    /* Count Check (Max Erase count) */
    if(!rtn)
        break;
    ferase(block_no);  /* Erase */
    rtn = ferasevf((unsigned short*)ers_st,
                   (unsigned short*)ers_ed);
}

SWE1 = 0; /* Clear the SWE1 bit */
for(j = 0; j < WLOOP100; j++);  /* Need to wait 100 usec */
return(rtn);
}

/************************************************************/
/* Erase Block Check Routin                                  */
/************************************************************/
char blk_check ( unsigned long eck_ad, unsigned long *eck_st, unsigned long *eck_ed, unsigned char *blk_no )
{
    unsigned char i;

    for(i = 0; eck_ad != BLOCKADR[i]; i++) {     /* COMPARE BLOCK_START_ADDRESS */
        if(MAXBLK1 < i)          /* BLOCK NUMBER MAX? */
            return(NG);          /* ERASE BLOCK ADDRESS ERROR */
    }

    *blk_no = i;           /* ERASE BLOCK NUMBER */
    *eck_st = BLOCKADR[i];         /* ERASE START ADDRESS */
    i++;
    *eck_ed = BLOCKADR[i]-1;        /* ERASE END ADDRESS */

    return(OK);
}

/************************************************************/
/* Erase                                                   */
/************************************************************/
void ferase ( unsigned char e_blk_no )
{
    unsigned char i;
    unsigned short j;
    unsigned char tmp;

    tmp = 1;
    if(e_blk_no < 8){
        tmp <<= e_blk_no;
        EBR1 = tmp;           /* Set the EBR1 Erase Block bit */
    }
    else{
        e_blk_no = e_blk_no - 8;
        tmp <<= e_blk_no;
        EBR2 = tmp;           /* Set the EBR2 Erase Block bit */
    }

    TCSRW_0 = 0xA57F;          /* WDT START phi/13072 */
    ESU1 = 1;            /* Set the ESU1 bit */
    for(j = 0; j < WLOOP100; j++);  /* Need to wait 100 usec */
    E1 = 1;            /* Set the E1 bit (ERASE) */
    for(j = 0; j < TIME10000; j++);  /* Need to wait 10 msec */
    E1 = 0;            /* Clear the E1 bit */
    for(i = 0; i < WLOOP10; i++);  /* Need to wait 10 usec */
    ESU1 = 0;            /* Clear the ESU1 bit */
for(i = 0; i < WLOOP10; i++); /* Need to wait 10 usec */
TCSRQ_0 = 0xA500; /* WDT STOP */
EBR1 = 0;
EBR2 = 0;
}/************************************************************/
/* Erase Verify */
}/************************************************************/
char ferasevf (unsigned short *evf_st, unsigned short *evf_ed)
{
    char rtn;
    unsigned char i;
    unsigned short j;
    unsigned char *ead;
    EV1 = 1; /* Set the EV bit */
    for(i = 0; i < WLOOP20; i++); /* Need to wait 20 usec */
    rtn = OK;
    ead = (unsigned char*)evf_st;
    for(i = 0; i < WLOOP2; i++); /* Need to wait 2 usec */
    if(evf_st[j] != 0xFFFF){ /* Verify */
        rtn = NG; /* NG flag set */
        break;
    }
    EV1 = 0; /* Clear the EV bit */
    for(i = 0; i < WLOOP4; i++); /* Need to wait 4 usec */
    return(rtn); /* OK flag set */
}/************************************************************/
/* Flash Memory 128 byte Write */
}/******************************************************************************/
char fwrite128 (unsigned char *wt_buf, unsigned char *wt_adr, unsigned short WT_COUNT)
{
    char rtn;
    unsigned char i;
    unsigned short j;
    unsigned short TM;
    unsigned short TM2;
    unsigned char *OWBUFF[128]; /* Over Write Data Area */
    unsigned char *BUFF[128]; /* Retry Write Data Area */
    memcpy(BUFF,wt_buf,128); /* W_BUF -> BUFF BLOCK COPY */
    SWE1 = 1; /* Set the SWE1 bit */
    for(i = 0; i < WLOOP1; i++); /* Need to wait 1 usec */
    rtn = fwritevf((unsigned short *)OWBUFF, /* 1st Program Verify */
        (unsigned short *)BUFF,
        (unsigned short *)wt_buf,
        (unsigned short *)wt_adr);
    if(rtn == NG){ /* 1st Verify END */
        TM = TIME30; /* Input P Palse(30 usec) */
        for(j = 0; j < WT_COUNT; j++){
            fwrite(BUFF,wt_adr,TM); /* Input P Palse(10,30,200 usec) */
            rtn = fwritevf((unsigned short *)OWBUFF, /* 1st Program Verify */
                (unsigned short *)BUFF,
                (unsigned short *)wt_buf,
                (unsigned short *)wt_adr);
        }
    }
if(j < OW_COUNT){
    fwrite(OWBUFF,wt_adr,TIME10);
} else{
    TM = TIME200;
}
if(rtn != NG){
    break;
}
}
SWE1 = 0; /* Clear the SWE1 bit */
for(j = 0; j < WLOOP100; j++) { /* Need to wait 100 usec */
    return(rtn);
}

/************************************************************/
/* Flash Memory Write                                       */
/************************************************************/
void fwrite ( unsigned char *buf, unsigned char *w_adr, unsigned short ptime )
{
    unsigned char i;
    unsigned short j;
    for(i = 0; i < 128; i++){ /* 128 byte repeat */
        w_adr[i] = buf[i]; /* Rewrite data dummy write */
    }
    TCSRW_0 = 0xA579; /* WDT START phi/64 */
    PSU1 = 1; /* Set the PSU1 bit */
    for(j = 0; j < WLOOP50; j++) { /* Need to wait 50 usec */
        P1 = 1; /* Set the P1 bit */
        for(j = 0; j < ptime; j++) { /* Writing Time 10/30/200 usec */
            P1 = 0; /* Clear the P1 bit */
            for(i = 0; i < WLOOP5; i++) { /* Need to wait 5 usec */
                PSU1 = 0; /* Clear the PSU1 bit */
                for( i = 0; i < WLOOP5; i++) { /* Need to wait 5 usec */
                    TCSRW_0 = 0xA500; /* WDT STOP */
                }
            }
        }
    }
}

/************************************************************/
/* Flash Memory Verify                                     */
/************************************************************/
char fwritevf ( unsigned short *owbuff, unsigned short *buff , unsigned s hort *wvf_buf, unsigned s hort *wvf_adr )
{
    char rtn;
    unsigned char i;
    unsigned char j;
    unsigned short tmp;
    unsigned char *wad;
    PV1 = 1; /* Set the PV1 bit */
    for(i = 0; i < WLOOP4; i++); /* Need to wait 4 usec */
    wad = (unsigned char*)wvf_adr;
    for(j = 0; j < 128/2; j++){ /* Dummy Write */
        wad[j*2] = 0xFF; /* Need to wait 2 usec */
        for(i = 0; i < WLOOP2; i++) { /* Need to wait 2 usec */
            owbuff[j] = buff[j] | wvf_adr[j];
        }
    }
}
tmp = ~wvf_adr[j];
buff[j] = tmp | wvf_buf[j];

tmp = tmp & wvf_buf[j]; /* Error Check */
if(tmp != 0)
    break;
}

PV1 = 0; /* PV1 bit Clear */
for(i = 0; i < WLOOP2; i++); /* Need to wait 2 usec */

if(tmp == 0){
    rtn = OK;
    for(j = 0; j < 128/2; j++){
        /* 128 byte OK? */
        if(buff[j] != 0xFFFF){ /* Error Check */
            rtn = NG;
            break;
        }
    }
} else{
    rtn = WNG; /* Write Error */
}

return(rtn);

#pragma section FZEND
9.3 Asynchronous Serial Communication Program

```c
#pragma section ASSCI
#include <machine.h>

/* Symbol Definition */
*/

struct BIT {
    unsigned char b7:1; /* bit7 */
    unsigned char b6:1; /* bit6 */
    unsigned char b5:1; /* bit5 */
    unsigned char b4:1; /* bit4 */
    unsigned char b3:1; /* bit3 */
    unsigned char b2:1; /* bit2 */
    unsigned char b1:1; /* bit1 */
    unsigned char b0:1; /* bit0 */
};

#define SMR_0  *(volatile unsigned char *)0xFFFF78 /* Serial Mode Register */
#define BRR_0  *(volatile unsigned char *)0xFFFF79 /* Bit Rate Register */
#define SCR_0  *(volatile unsigned char *)0xFFFF7A /* Serial Control Register */
#define SCR_0_BIT  (*(volatile struct BIT *)0xFFFF7A) /* Serial Control Register */
#define TE_0   SCR_0_BIT.b5       /* Transmit Enable */
#define RE_0   SCR_0_BIT.b4       /* Receive Enable */
#define CKE1_0  SCR_0_BIT.b1       /* Clock Enable 1 */
#define CKE0_0  SCR_0_BIT.b0       /* Clock Enable 0 */
#define TDR_0  *(volatile unsigned char *)0xFFFF7B /* Transmit Data Register */
#define SSR_0  *(volatile unsigned char *)0xFFFF7C /* Serial Status Register */
#define SSR_0_BIT  (*(volatile struct BIT *)0xFFFF7C) /* Serial Status Register */
#define TDR_E_0  SSR_0_BIT.b7 /* Transmit Data Register Empty */
#define RDRF_0  SSR_0_BIT.b6 /* Receive Data Register Full */
#define ORER_0  SSR_0_BIT.b5 /* Overrun Error */
#define FER_0  SSR_0_BIT.b4 /* Framing Error */
#define PER_0  SSR_0_BIT.b3 /* Parity Error */
#define TEND_0  SSR_0_BIT.b2 /* Transmit End */
#define RDR_0  *(volatile unsigned char *)0xFFFF7D /* Receive data Register */
#define SCMR  *(volatile unsigned char *)0xFFFF7E /* Smart Card Mode Register */
#define SEMR_0  *(volatile unsigned char *)0xFFFFDF /* Serial Expansion Mode Register */
#define MSTFCRB  *(volatile unsigned char *)0xFFFFDE9 /* Module Stop Control Registers */

void com_init ( void )
{
    unsigned short i;

    MSTFCRB |= 0x7F; /* module stop mode is cleared */
    SCR_0 |= 0xCF; /* TE,RE=0 */
    SCR_0 |= 0xFC; /* CKE1,CKE0=0 */
    SMR_0 = 0x00; /* Initialize Serial Mode Register */
    SCMR = 0x02; /* Don't use Smart Card */
    SEMR_0 = 0x00; /* Ibit-interval base clock is */
    /* 1times the transfer rate. */
    BRR_0 = 7; /* 38400 bps phi=10MHz */
}
```
for(i = 0; i < 270; i++);        /* Dummy Loop ,26.04us over Wait */
i = SSR_0;
SSR_0 &= 0xC7;           /* ORER,FER,PER=0 */
SCR_0 = 0x30;           /* TE=1,RE=1 */
}

/**************************************************************/
/* Receive 1 byte                                         */
/**************************************************************/
unsigned char rcv1byte ( void )
{
  unsigned char tmp;

do{
    tmp = RDRF_0;
    if(SSR_0 & 0x38)     /* ORER/FER/PER = 1 ? */
      while(1);          /* Receive Error */
  }while(tmp == 0);          /* End Serial Receiving */
  tmp = RDR_0;           /* Read Receive data */
  RDRF_0 = 0;           /* Clear RDRF bit */
  return(tmp);
}

/**************************************************************/
/* Receive N byte                                         */
/**************************************************************/
void rcvnbyte ( unsigned char *ram, unsigned char dtno )
{
  while(dtno--){           /* dtno = 0 ? */
    *ram = rcv1byte();         /* 1byte Receive Data -> RAM */
    *ram++;
  }
}

/**************************************************************/
/* Transmit 1 byte                                        */
/**************************************************************/
void trs1byte ( unsigned char tdt )
{
  while(TDRE_0 == 0);          /* End Serial Transmitting */
  TDR_0 = tdt;
  TDRE_0 = 0;
  while(TEND_0 == 0);          /* End Serial Transmitting */
}

/**************************************************************/
/* Transmit N byte                                        */
/**************************************************************/
void trsnbyte ( unsigned char *tdt, unsigned char dtno )
{
  while(dtno--){           /* dtno = 0 ? */
    while(TDRE_0 == 0);      /* End Serial Transmitting */
    TDR_0 = *tdt;
    TDRE_0 = 0;
    *tdt++;
  }
  while(TEND_0 == 0);        /* End Serial Transmitting */
}
Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Mar.09.05</td>
<td>—</td>
<td>—</td>
<td>First edition issued</td>
</tr>
<tr>
<td>2.00</td>
<td>Aug.29.06</td>
<td>3, 25</td>
<td>3, 25</td>
<td>New addition and content correction</td>
</tr>
</tbody>
</table>
Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. 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 Corp. or a third party.

2. Renesas Technology Corp. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. by various means, including the Renesas Technology Corp. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.

5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. for further details on these materials or the products contained therein.

© 2006. Renesas Technology Corp., All rights reserved.