To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

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

(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 $\overline{\text{IRQ0}}$ 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.

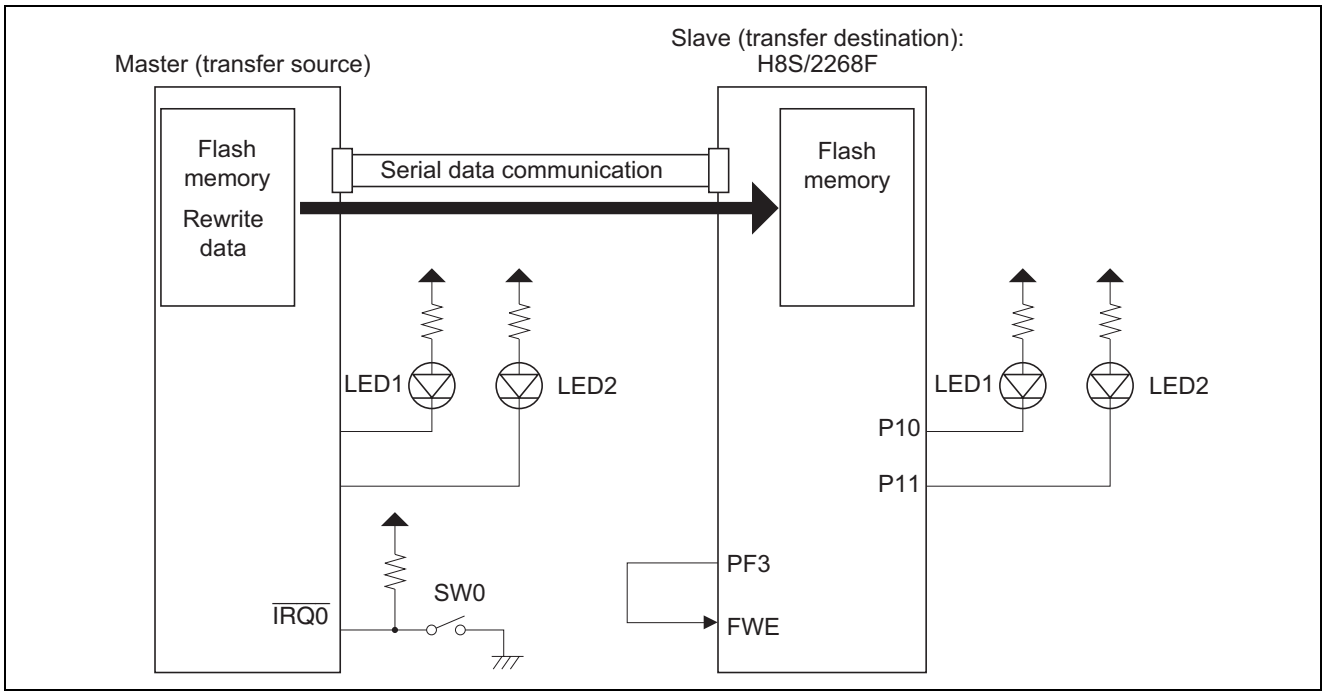


Figure 1 Configuration Example of On-Board Rewrite Circuit

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.

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

Table 1 Applicable Conditions

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

Table 2 Section Settings

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

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

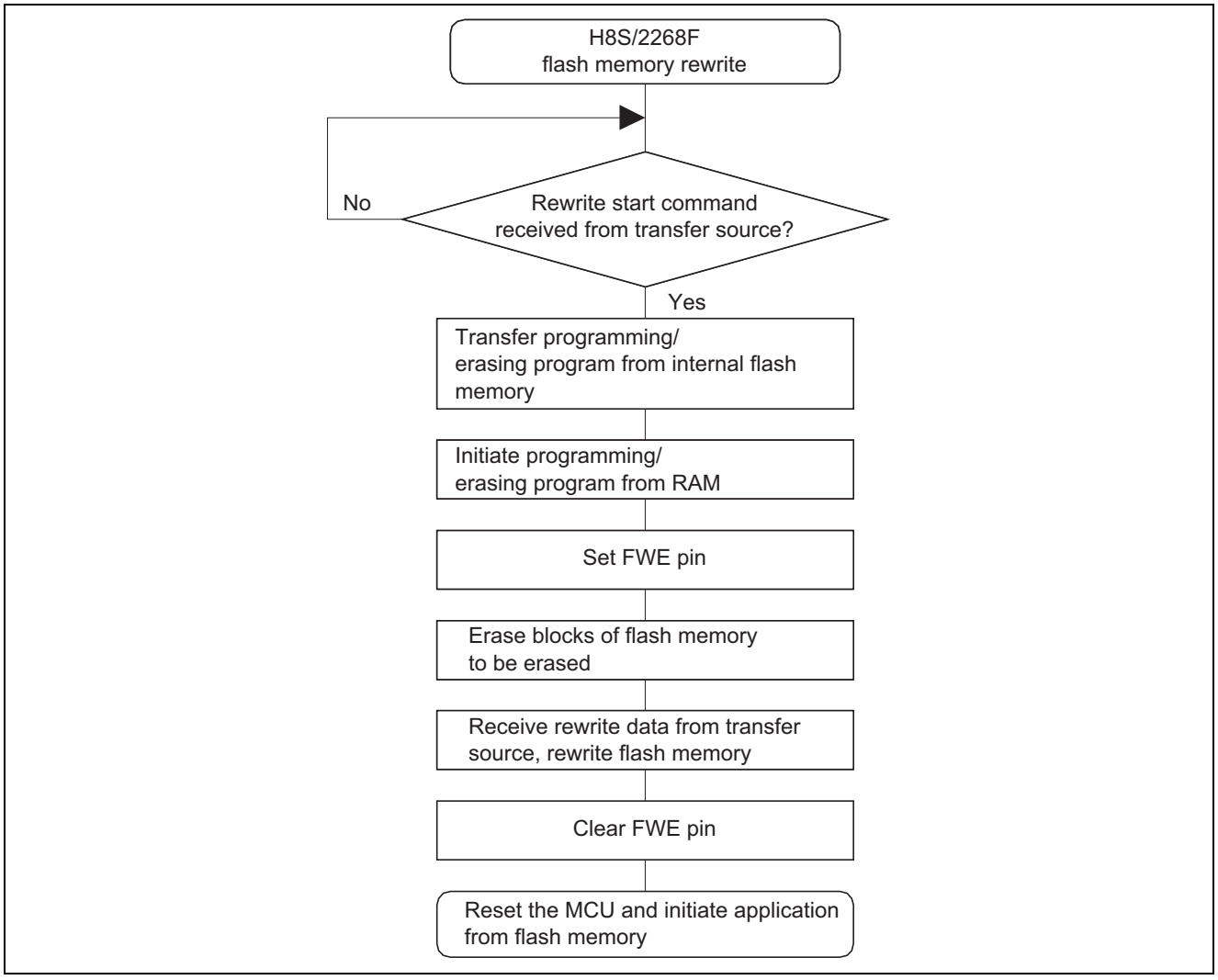


Figure 2 User Program Mode Rewrite Procedure

3.5 Master-Slave Connection Diagram

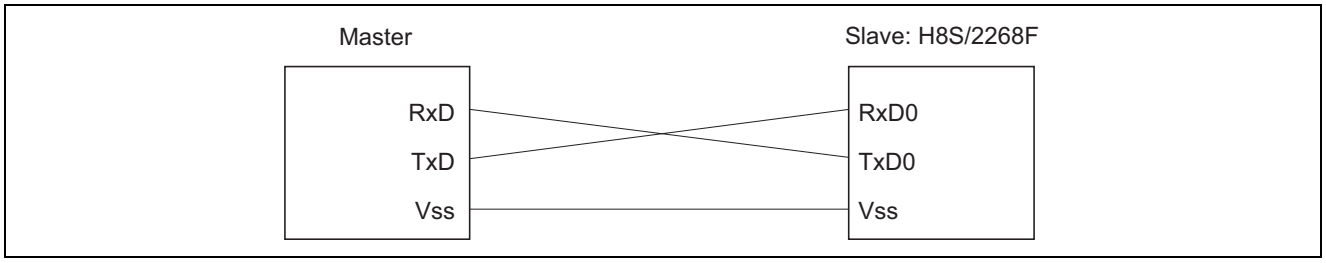


Figure 3 Master-Slave Connection Diagram

3.6 Communication Specifications

Table 3 Communication Specifications

Item	Description
Transfer speed	31,250 bps
Communication type	Asynchronous serial communication
Data bits	8
Stop bits	1
Parity	No

3.7 Communication Commands

Table 4 Communication Commands

Communication Command	Description
H'00	Normal transfer (command name: OK command)
H'01	Transfer error (command name: NG command)
H'11	Transmit start request
H'55	Rewrite start command
H'66	FWE pin setting command
H'77	Erase command
H'88	Programming command

3.8 Memory Mapping

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

Table 5 Flash Memory Erase Blocks

Block (Size)	Address
EB0 (4 Kbytes)	H'000000 to H'000FFF
EB1 (4 Kbytes)	H'001000 to H'001FFF
EB2 (4 Kbytes)	H'002000 to H'002FFF
EB3 (4 Kbytes)	H'003000 to H'003FFF
EB4 (4 Kbytes)	H'004000 to H'004FFF
EB5 (4 Kbytes)	H'005000 to H'005FFF
EB6 (4 Kbytes)	H'006000 to H'006FFF
EB7 (4 Kbytes)	H'007000 to H'007FFF
EB8 (32 Kbytes)	H'008000 to H'00FFFF
EB9 (64 Kbytes)	H'010000 to H'01FFFF
EB10 (64 Kbytes)	H'020000 to H'02FFFF
EB11 (64 Kbytes)	H'030000 to H'03FFFF

Memory maps during normal operation of the H8S/2268F and during the flash memory rewrite operation are shown in figure 4.

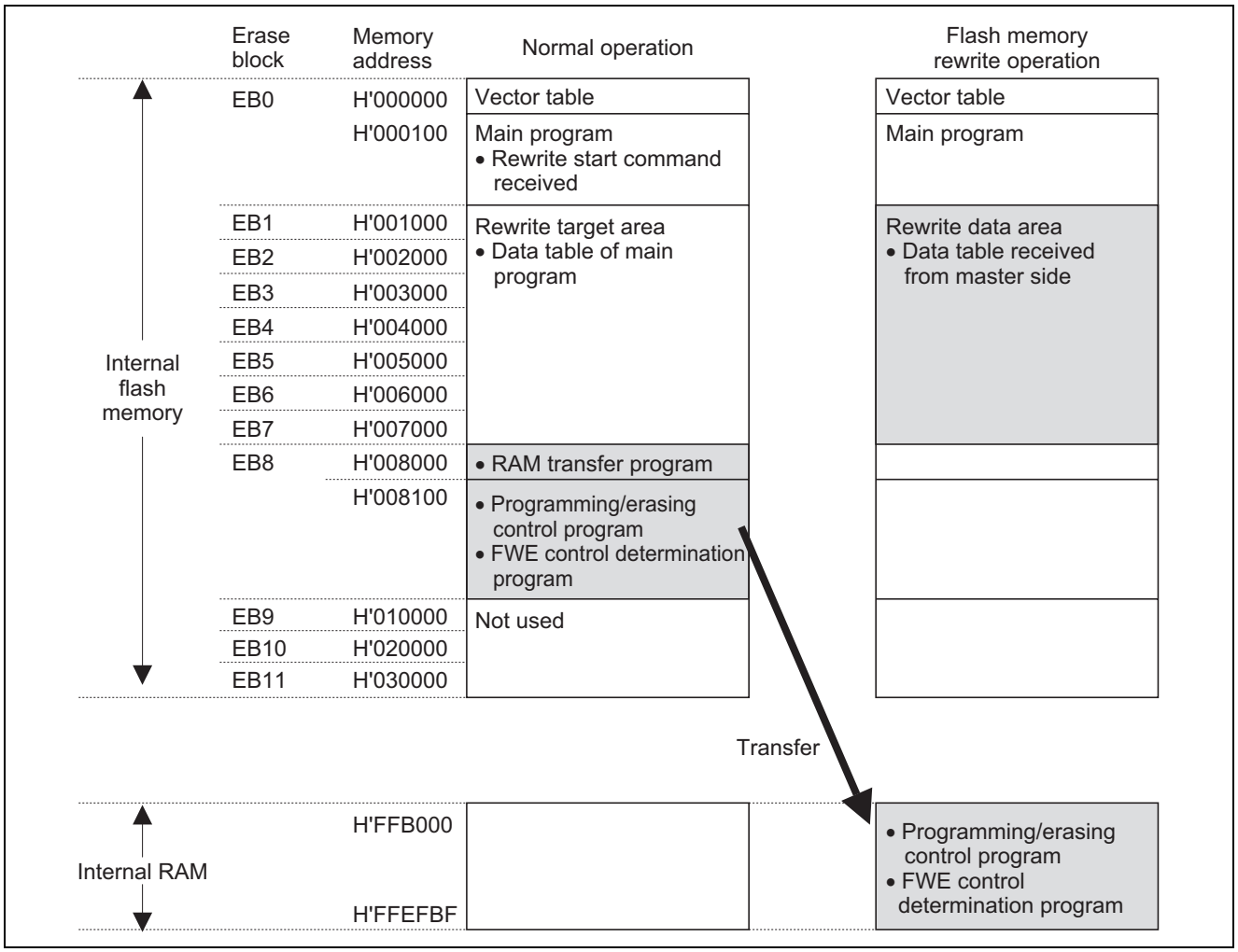


Figure 4 Memory Maps (Slave)

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.

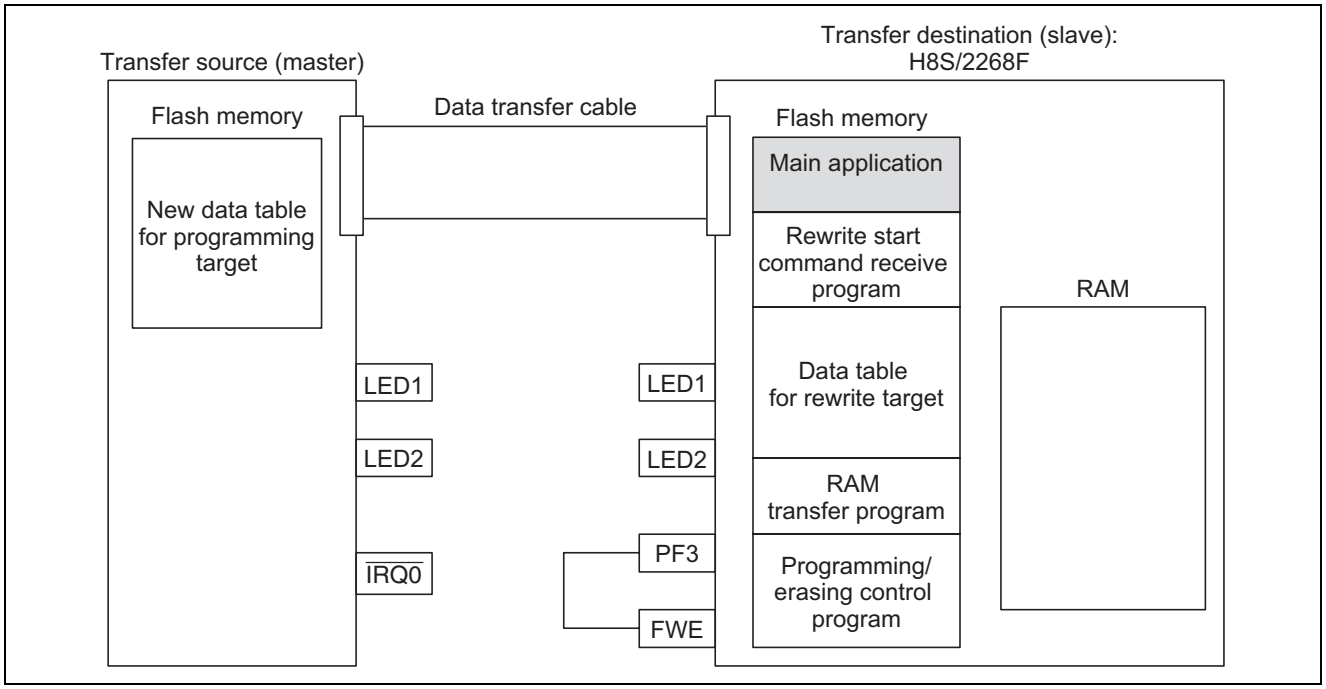


Figure 5 Normal Operation

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 $\overline{\text{IRQ0}}$ signal is detected.
(2) At this point LED1 is off and LED2 is lit on the master.

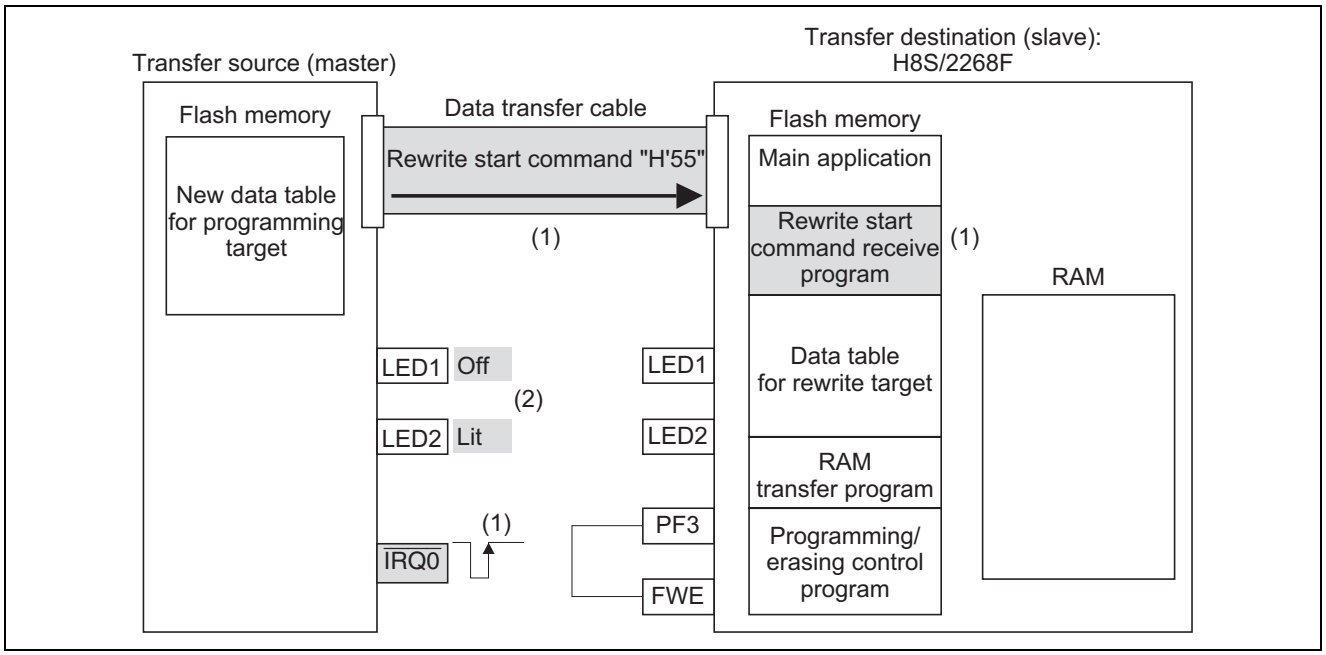


Figure 6 Preparation for On-Board Rewriting

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.

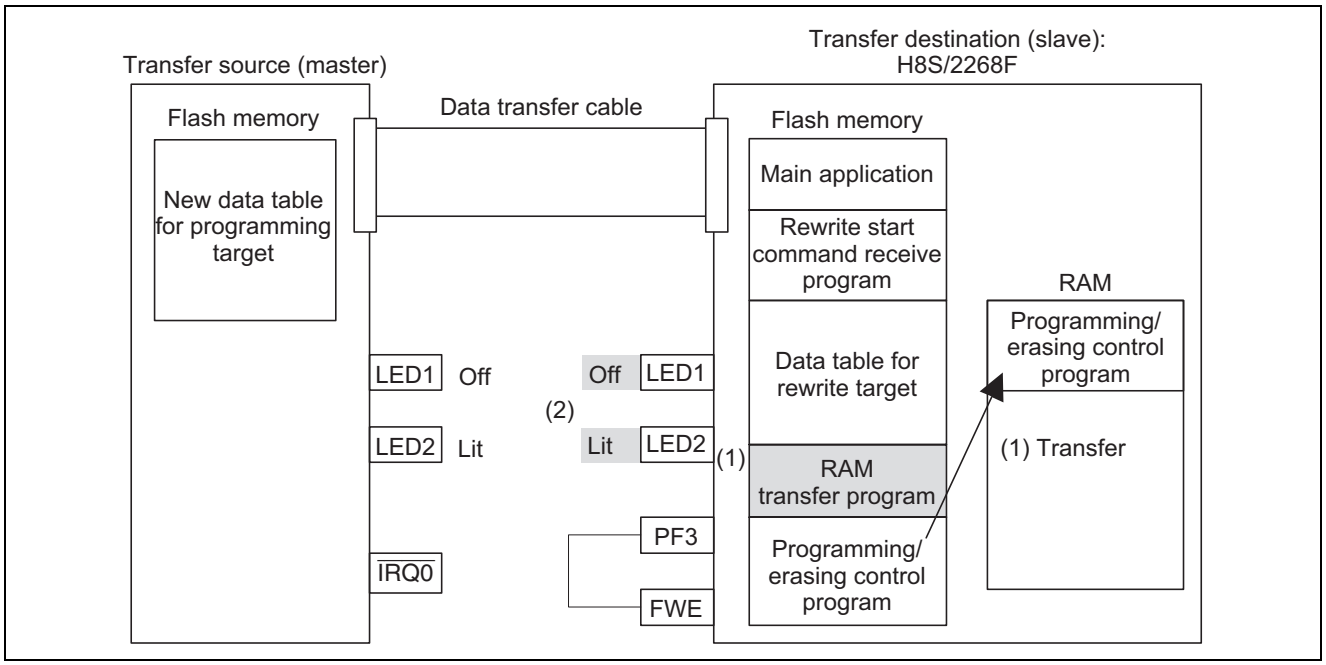


Figure 7 Start of On-Board Rewriting

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.

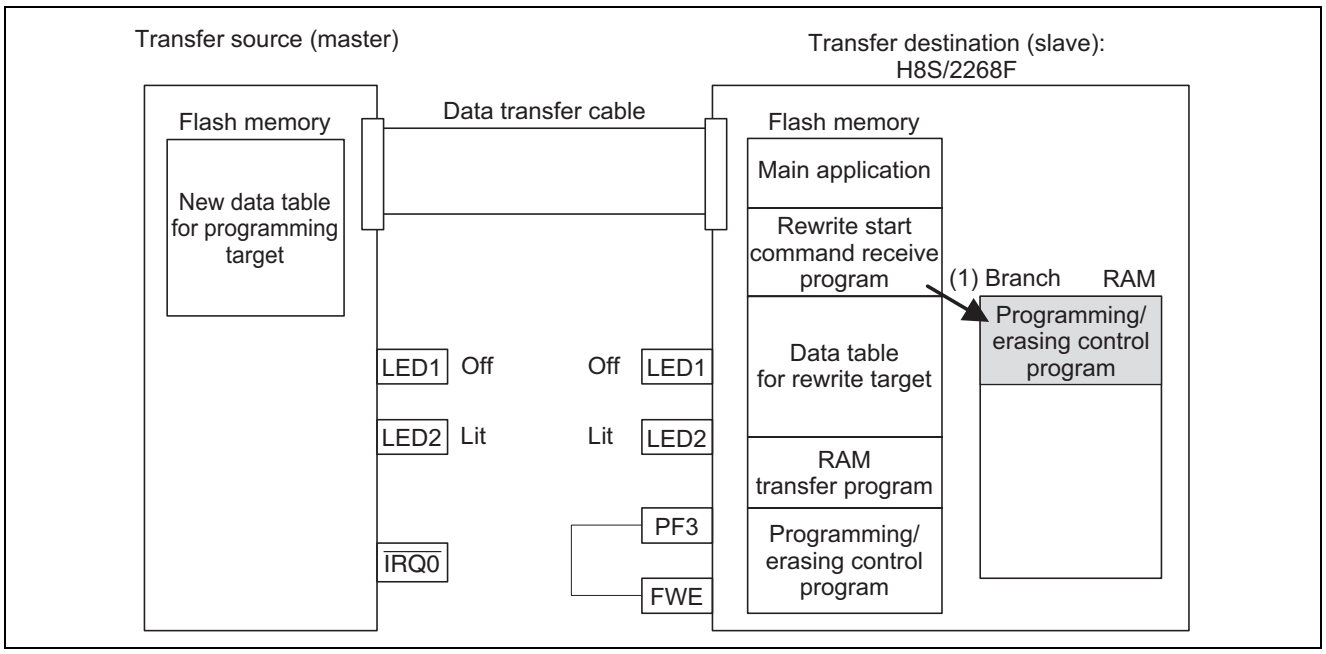


Figure 8 Startup of Programming/Erasing Control Program

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.

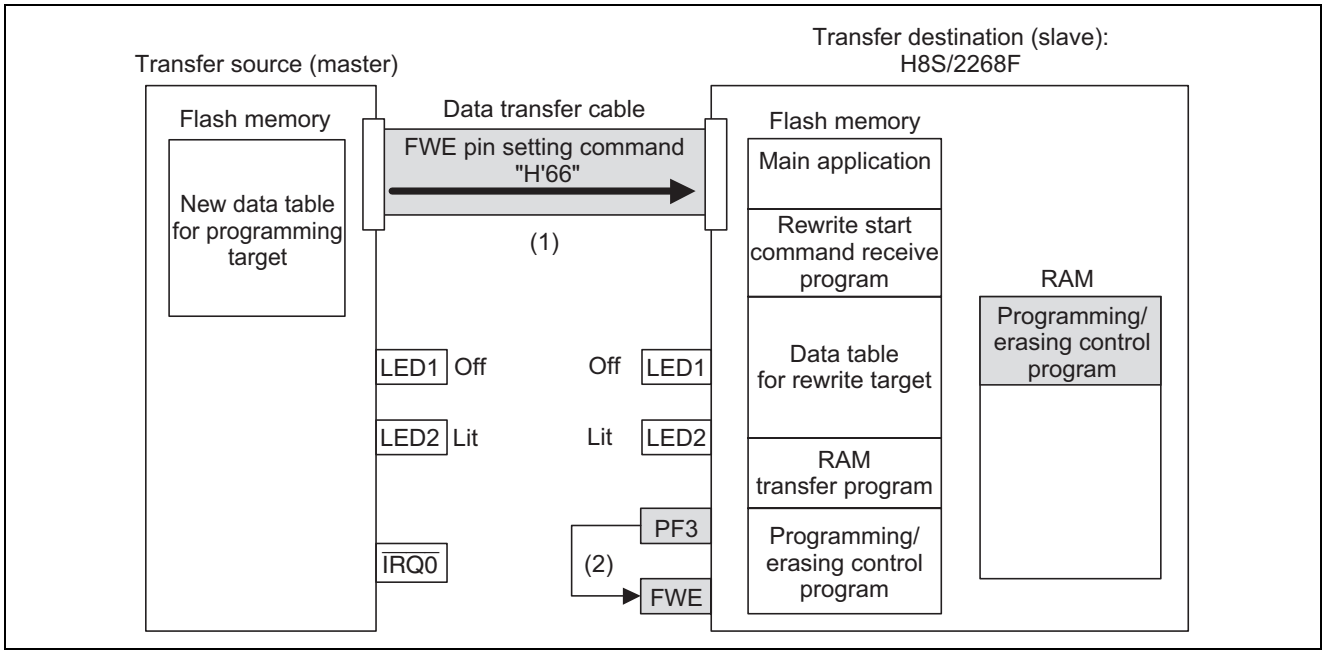


Figure 9 Setting of FWE Pin

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.

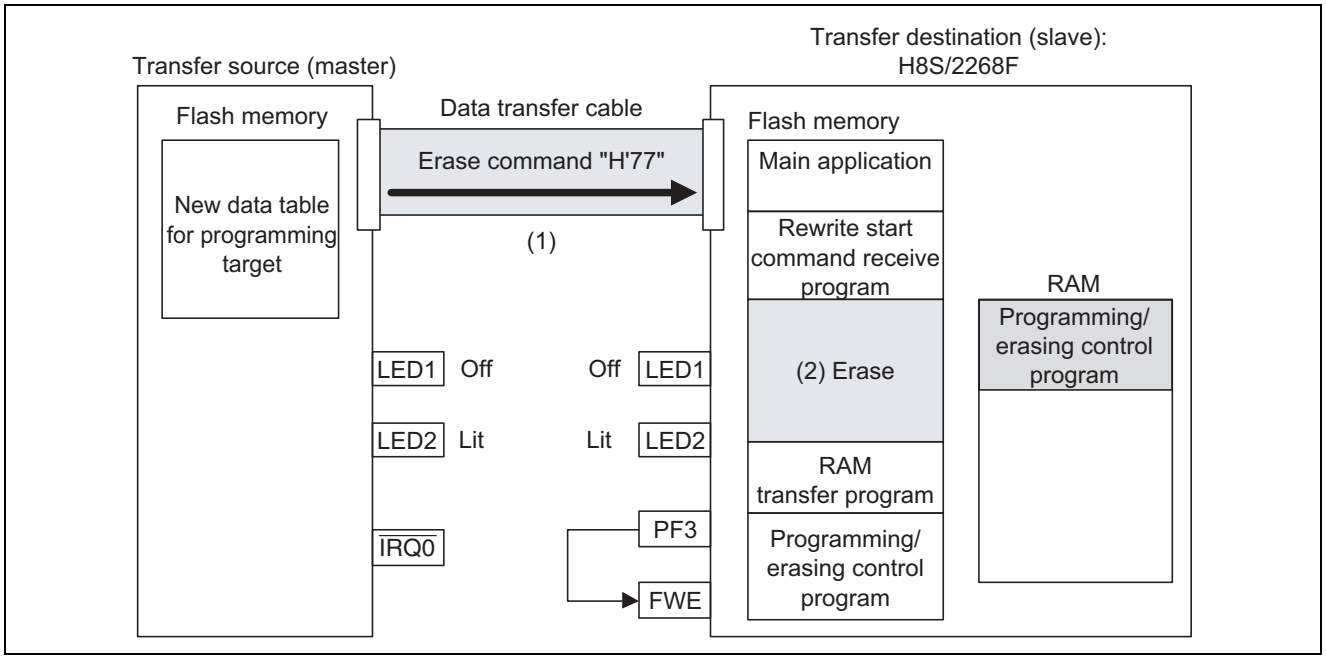


Figure 10 Erasing Flash Memory

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.

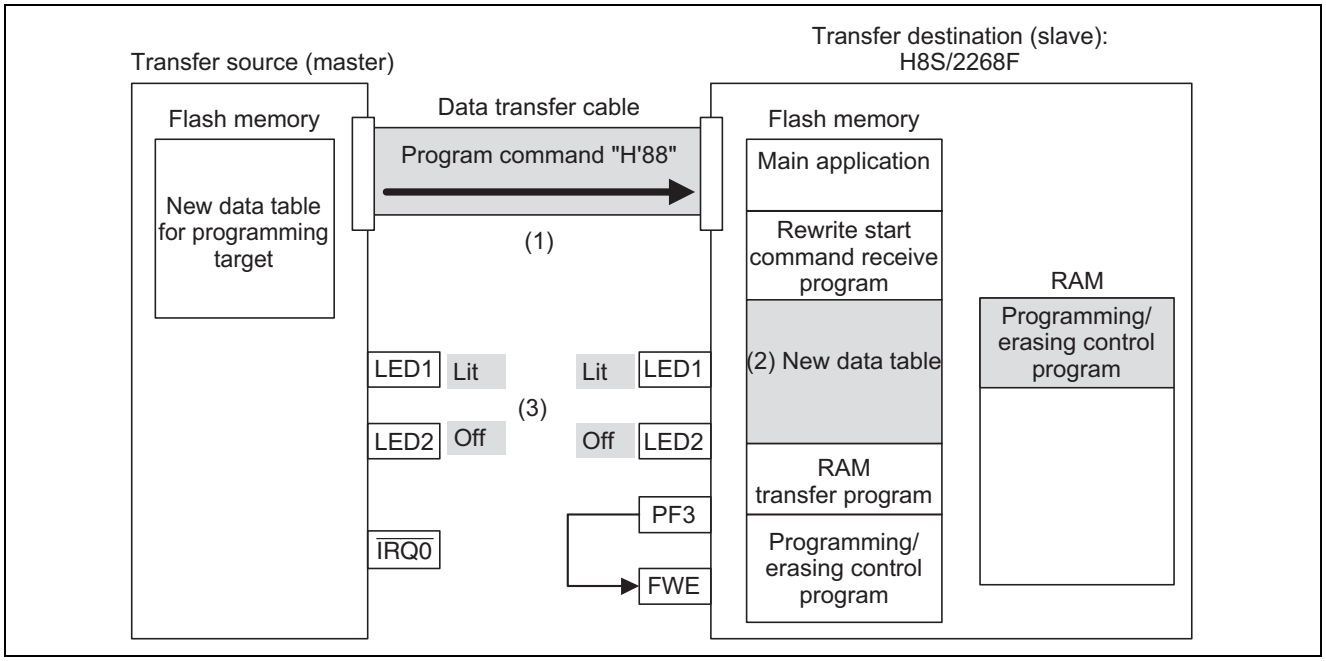


Figure 11 Programming Flash Memory

4.8 Clearing the FWE Pin

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

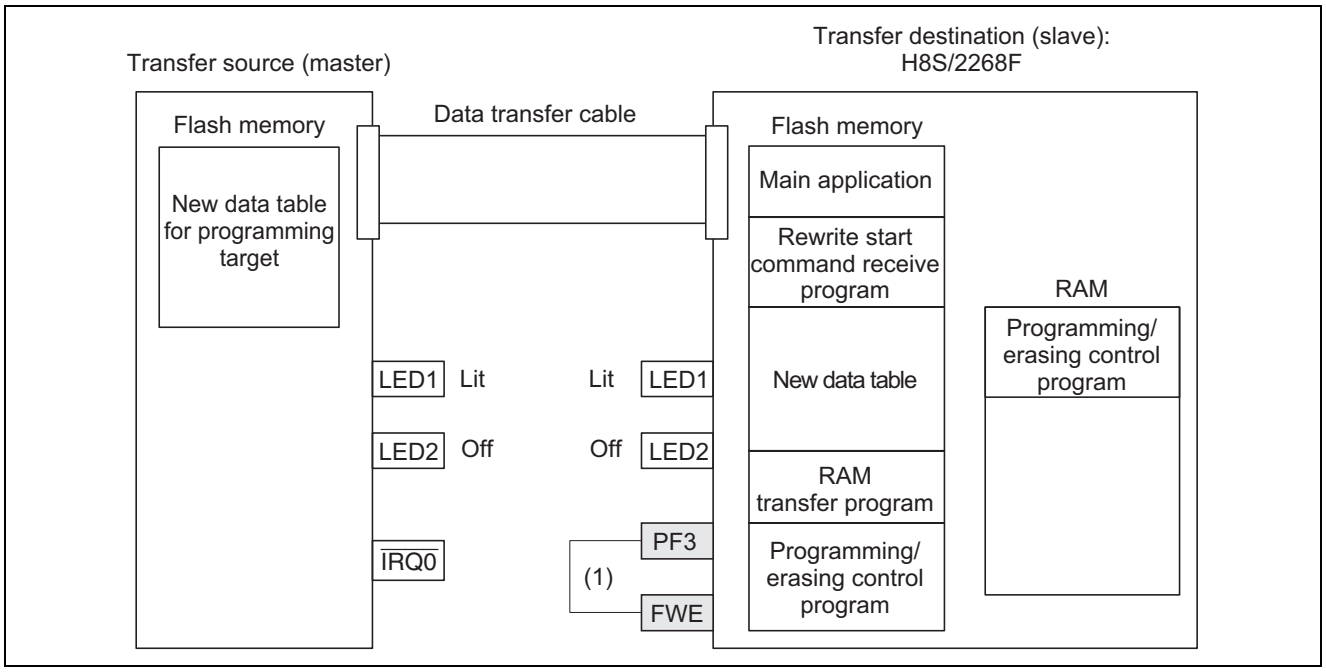


Figure 12 Clearing the FWE Pin

4.9 Initiating the Program

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

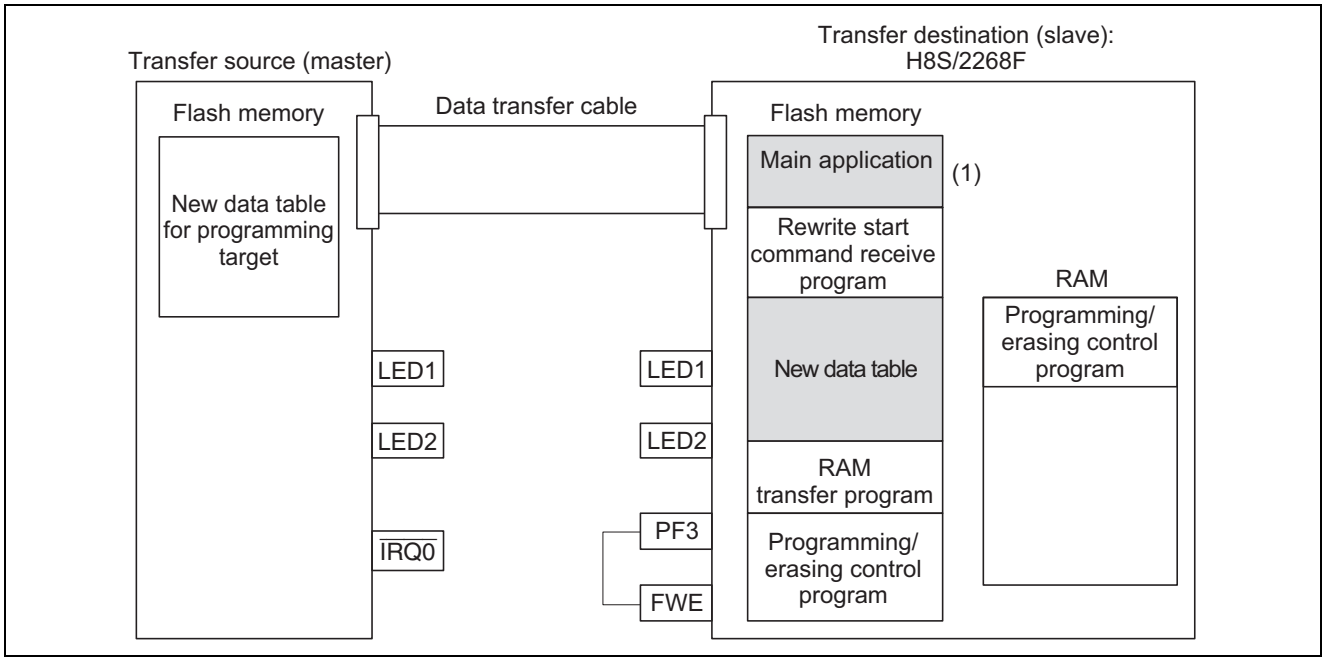


Figure 13 Initiating the Program

5. Sequence Diagram

(1) Normal Operation

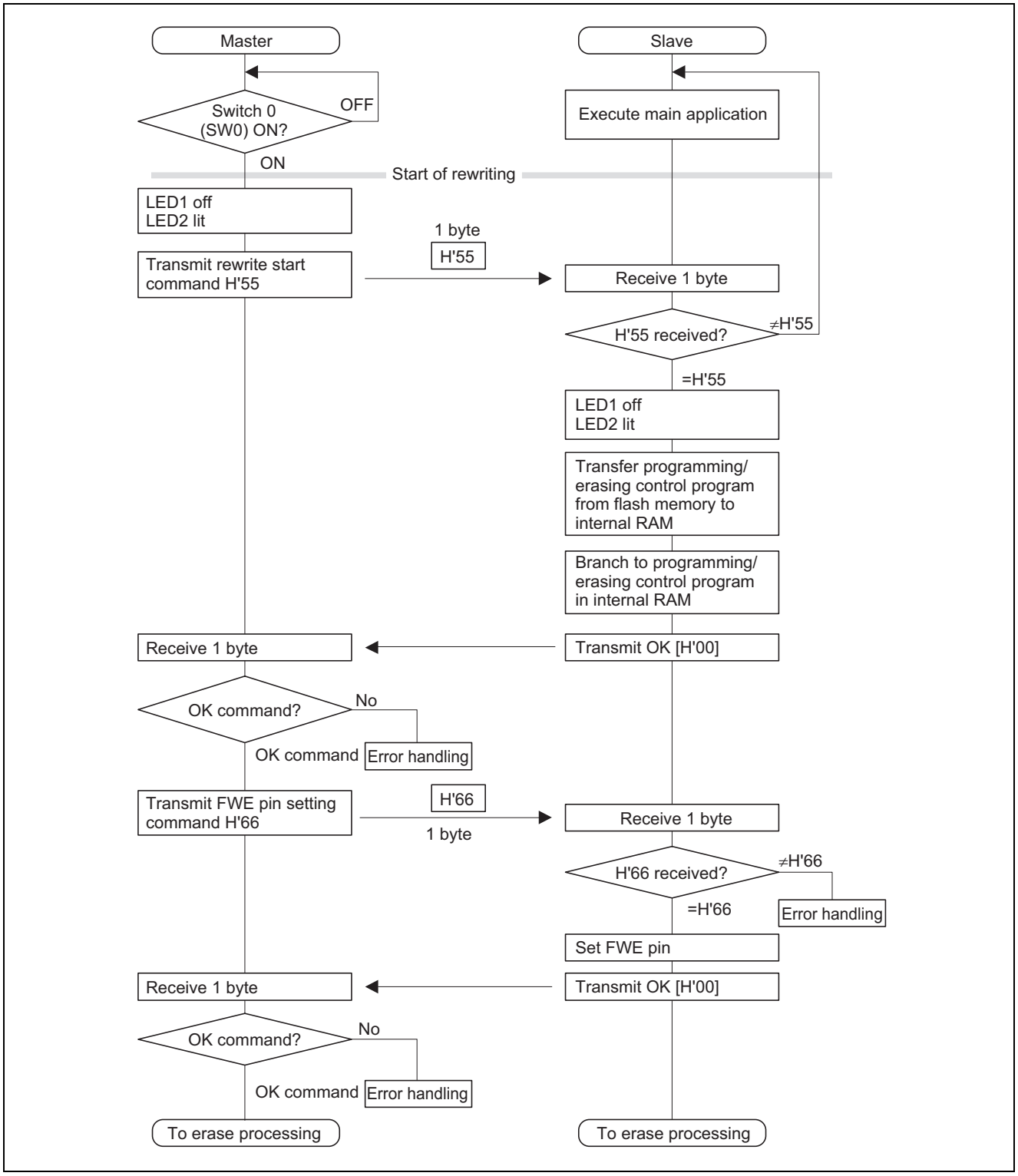


Figure 14 Normal Operation

(2) Erase Processing

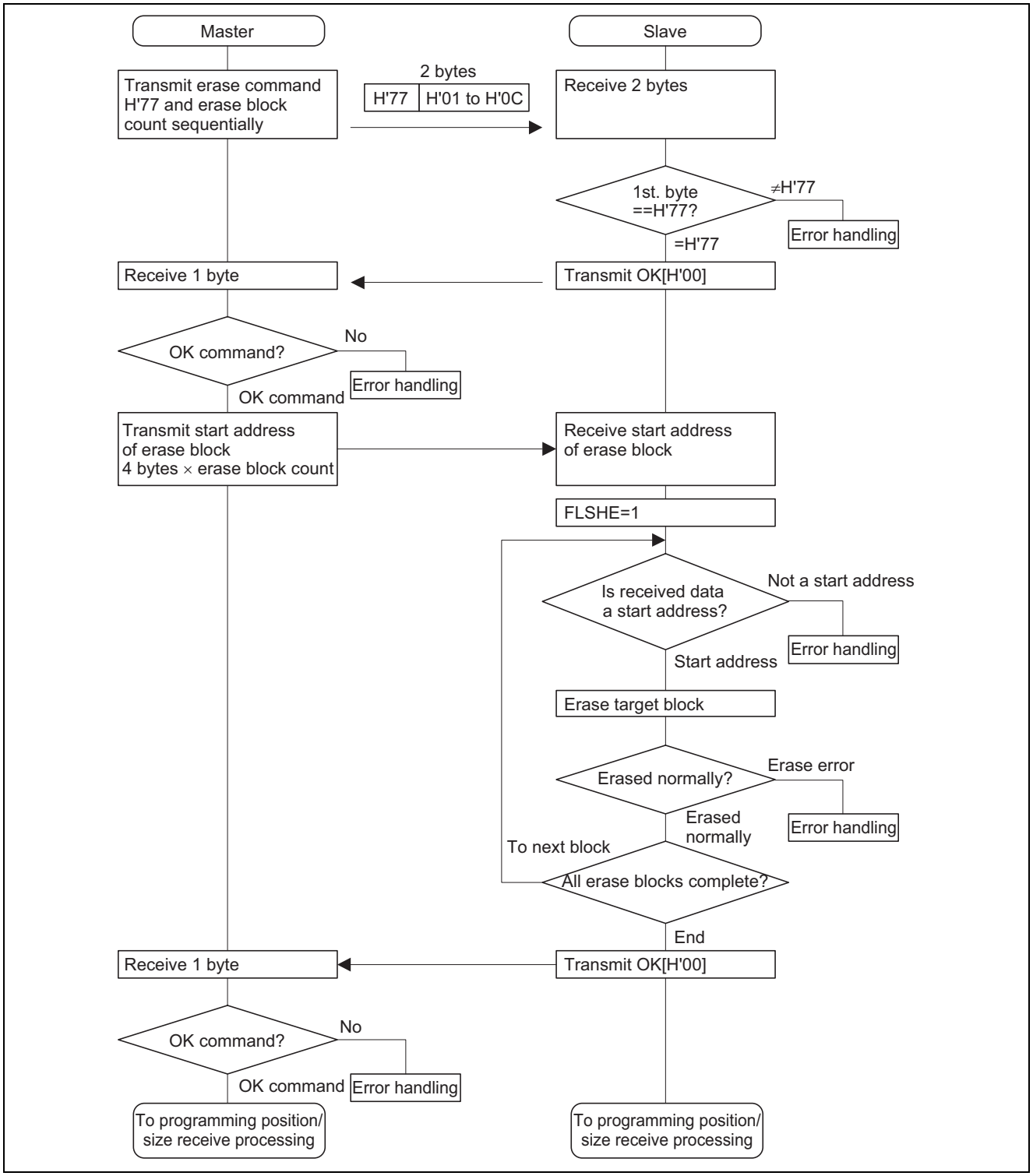


Figure 15 Erase Processing

(3) Programming Position/Size Reception Processing

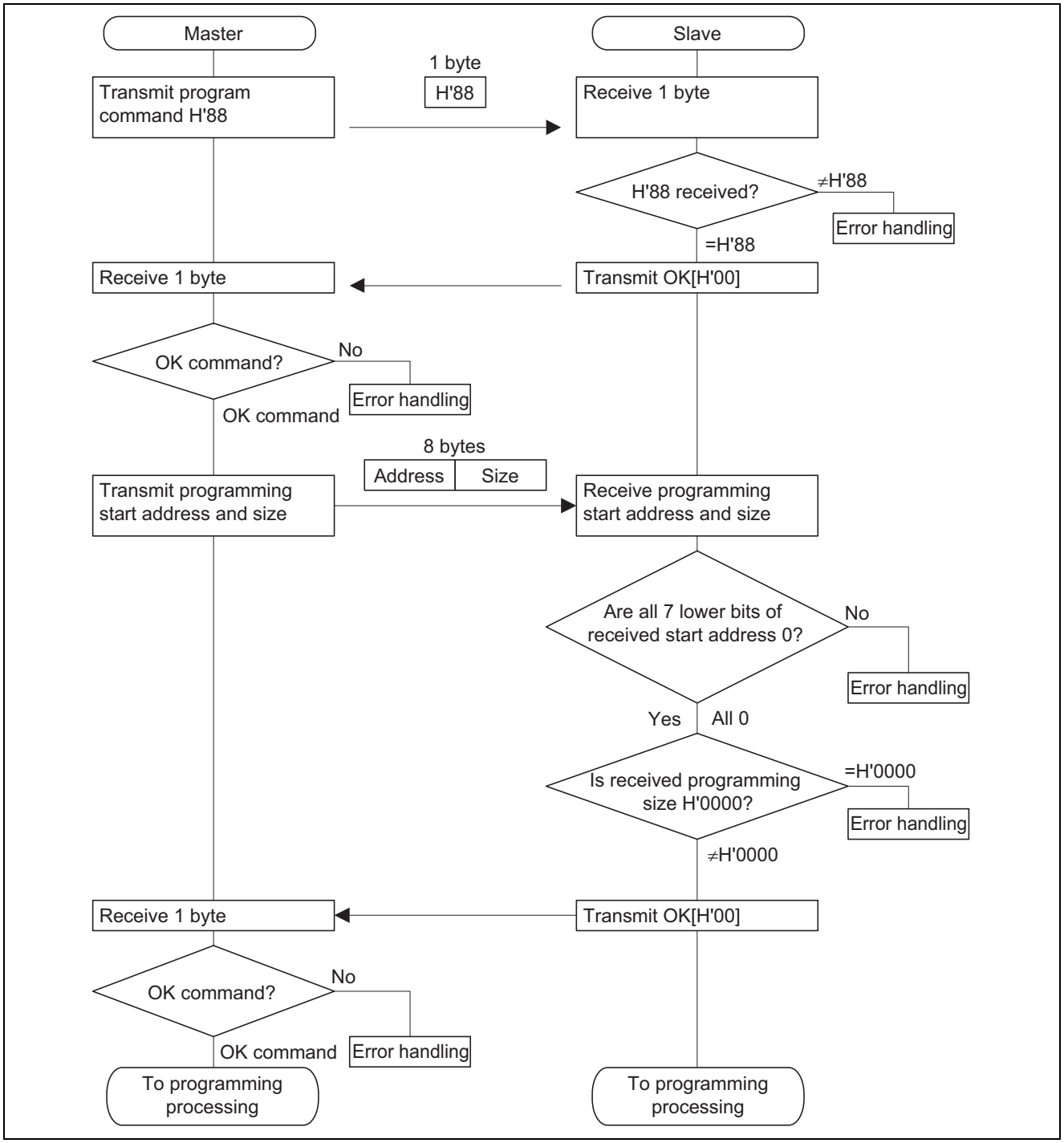


Figure 16 Programming Position/Size Reception Processing

(4) Programming Processing

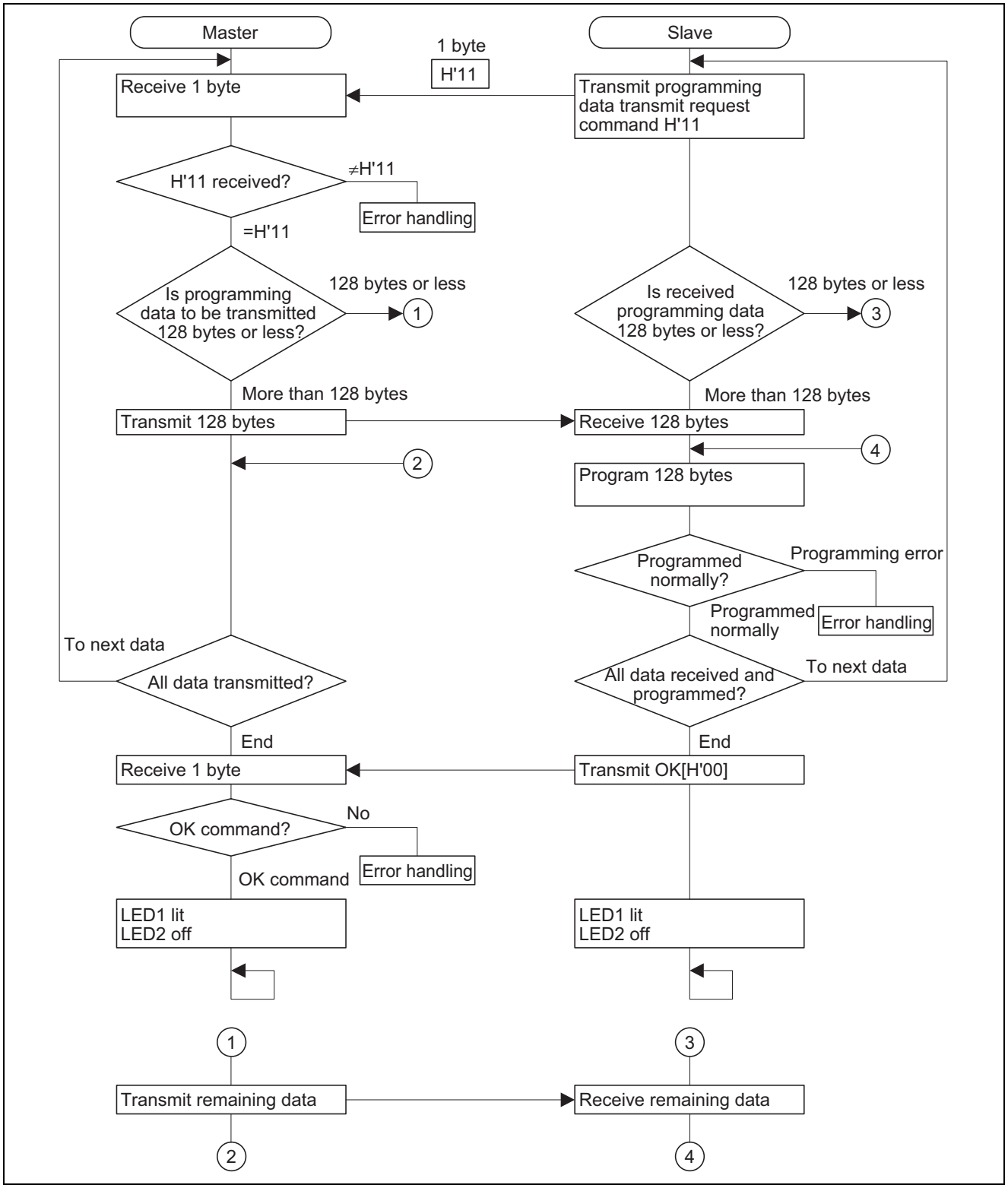


Figure 17 Programming Processing

(5) Error Handling

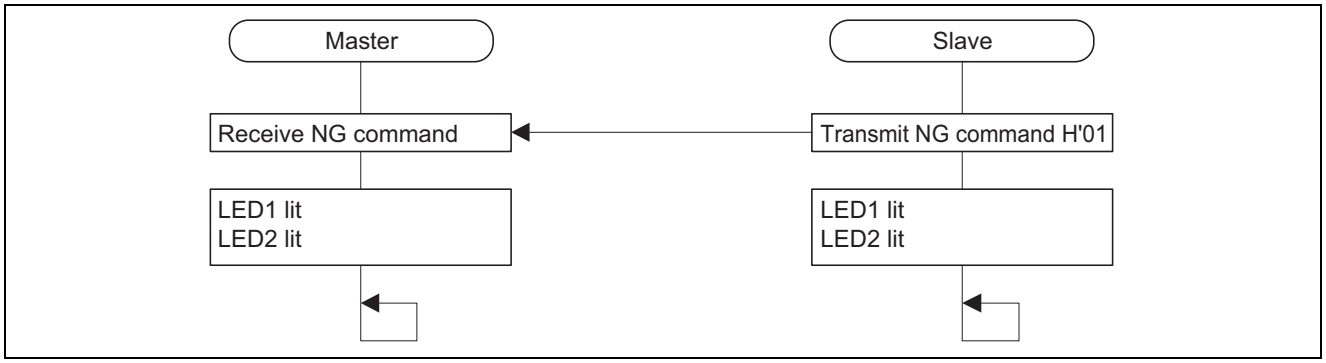


Figure 18 Error Handling

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.

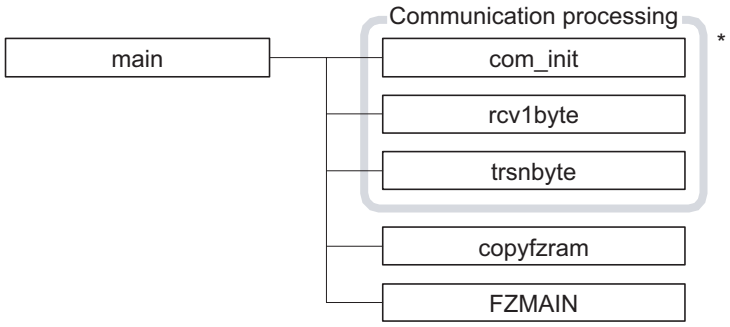


Figure 19 Hierarchy of Slave Main Program

6.2 List of Functions

Table 6 Functions of Slave Main Program

Function	Description
main	Executes main applications, receives rewrite start commands, transfers programming/erasing control program from flash memory to internal RAM
copyfzram	Transfers programming/erasing control program from flash memory to internal RAM
FZMAIN	Programming/erasing control program

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

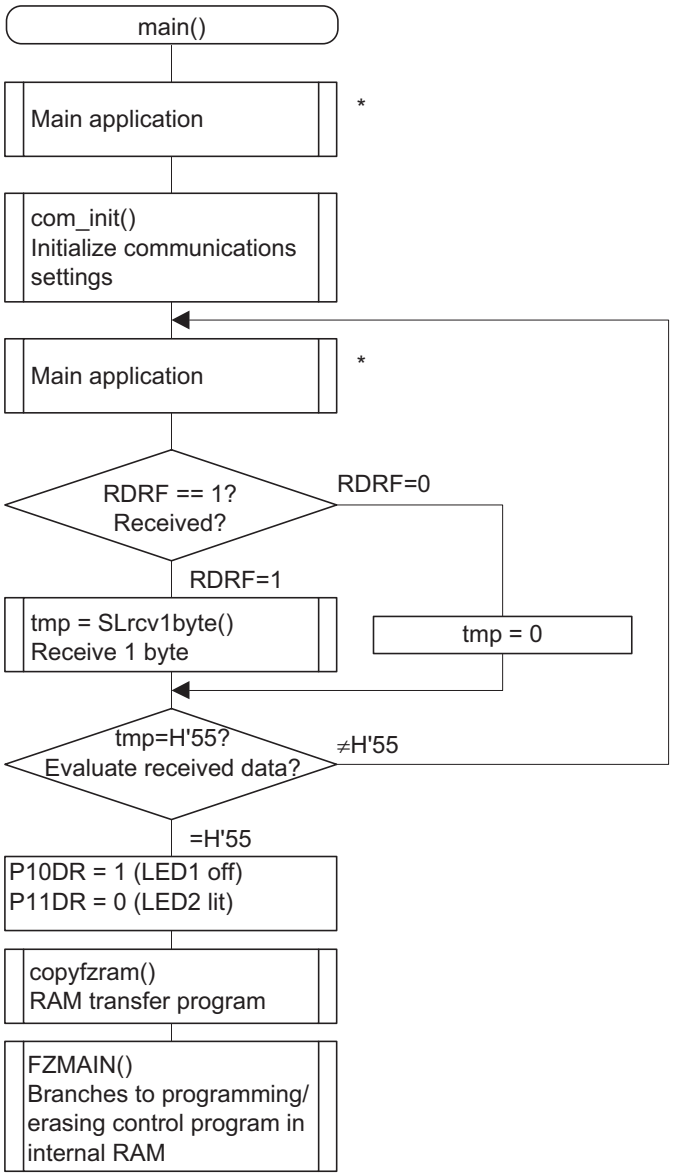
com_init(): Initializes communication settings
SLrcv1byte(): Receives 1 byte of data
copyfzram(): Transfers programming/erasing control program to internal RAM
FZMAIN(): Branches to programming/erasing control program

(f) Internal Registers Used

Table 7 Registers Used by main() Function

Register	Bit Name	Description	Address	Set Value
MSTPCRD		Module stop control register D	H'FFFC60	—
	MSTPD6	Used by sample main application.	Bit 6	—
P7DDR		Port 7 data direction register Used by sample main application.	H'FFFE36	—
P7DR		Port 7 data register Used by sample main application.	H'FFFF06	—
P0RT7		Port 7 register	H'FFFFB6	—
	P70	Port 70 Used by sample main application.	Bit 0	—
P1DDR		Port 1 data direction register • P1DDR = H'03: P11 and P10 set as output pins	H'FFFE30	H'03
P1DR		Port 1 data register	H'FFFF00	—
	P11DR	Port 11 data register • P11DR = 0: P11 output level low • P11DR = 1: P11 output level high	Bit 1	0
	P10DR	Port 10 data register • P10DR = 0: P10 output level low • P10DR = 1: P10 output level high	Bit 0	1
SSR_0		Serial status register_0	H'FFFF7C	—
	RDRF	Receive data register full • RDRF = 0: No received data stored in RDR_0 • RDRF = 1: Received data stored in RDR_0	Bit 6	—

(g) Flowchart



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

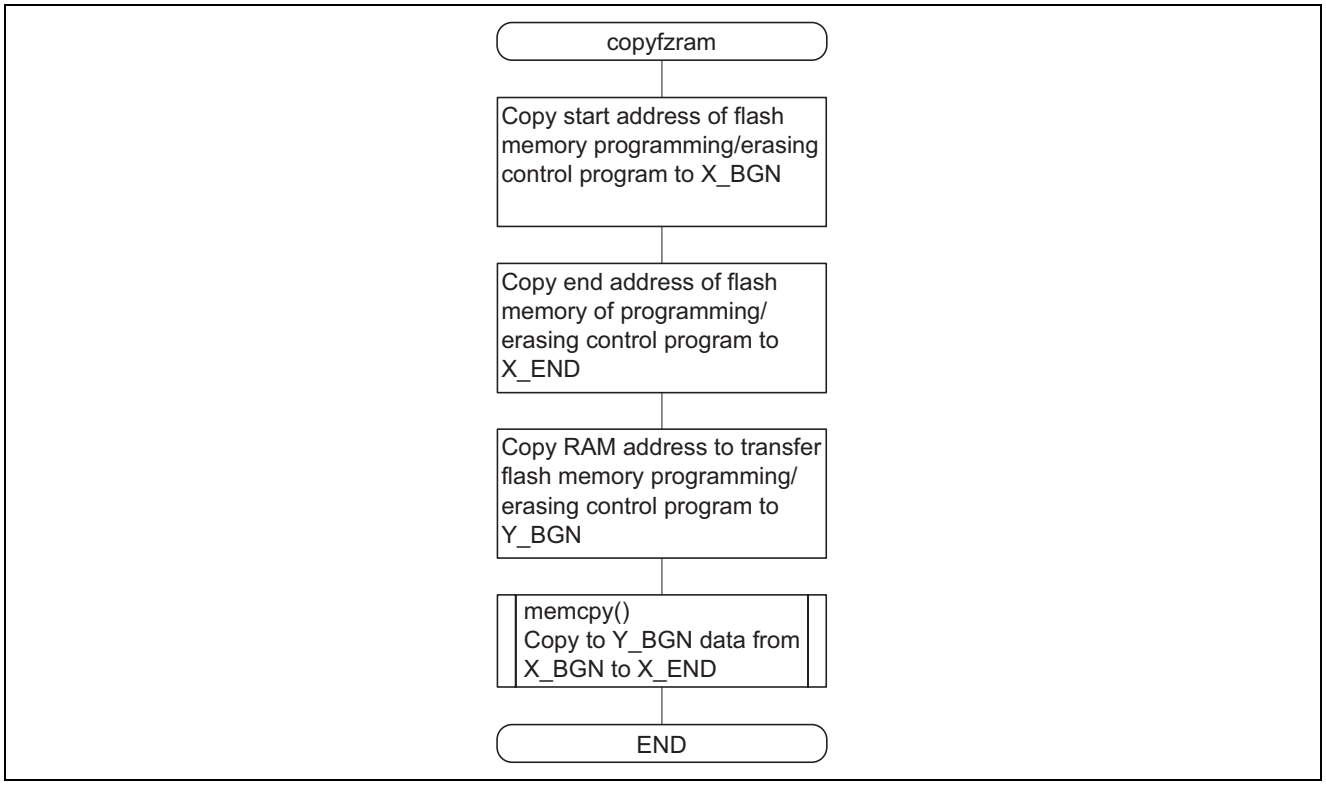
(e) Subroutines Used

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.

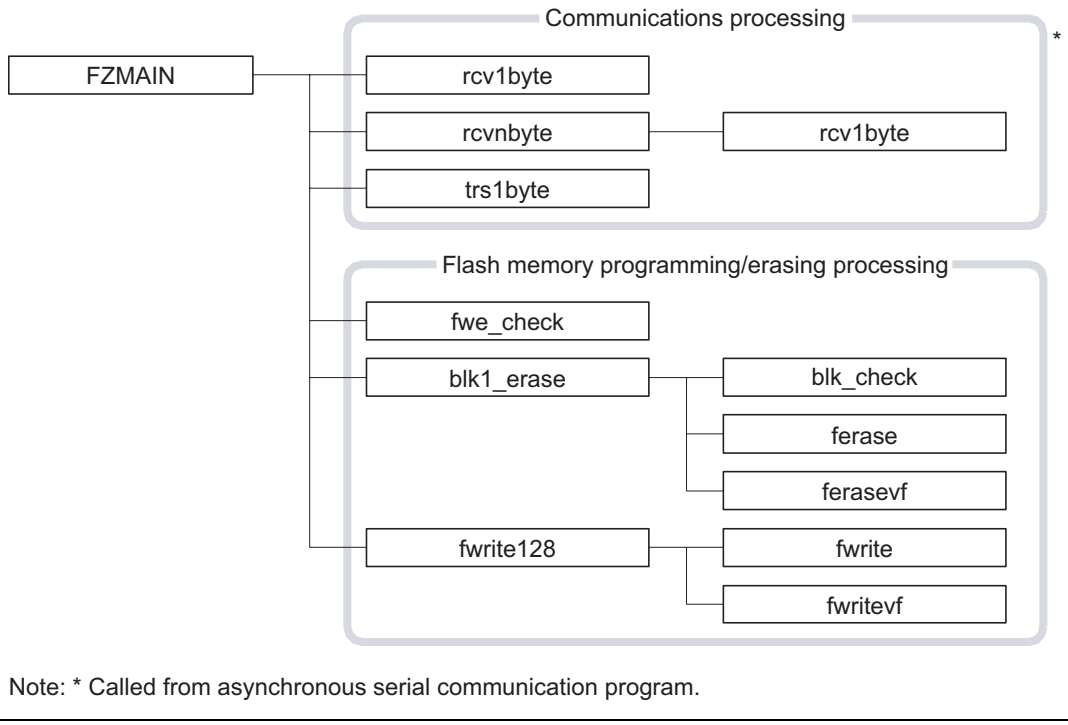


Figure 20 Hierarchy of Programming/Erasing Control Program

7.2 List of Functions

Table 8 Functions of Programming/Erasing Control Program

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

7.3 List of Constants

Table 9 List of Constants

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

Note: MHZ:10 · · · Operating frequency of 10 MHz
KEISU1:3 · · · 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

Data	Stack Memory Used
Programming data	128 bytes
Overwrite data	128 bytes
Additional programming data	128 bytes

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

Register	Bit Name	Description	Address	Set Value
P1DR		Port 1 data register	H'FFFF00	—
	P11DR	Port 11 data register • P11DR = 0: P11 output level low • P11DR = 1: P11 output level high	Bit 1	1
	P10DR	Port 10 data register • P10DR = 0: P10 output level low • P10DR = 1: P10 output level high	Bit 0	0
TCSR_0 *1		Timer control/status register_0	H'FFFF74	H'00
	OVF	Overflow flag • OVF = 0: No TCNT_0 overflow • OVF = 1: TCNT_0 overflow occurred	Bit 7	0
	WT/ $\overline{IT}$	Timer mode select • WT/ $\overline{IT}$ = 0: Interval timer • WT/ $\overline{IT}$ = 1: Watchdog timer	Bit 6	0
	TME	Timer enable • TME = 0: TCNT_0 count start • TME = 1: TCNT_0 count halt	Bit 5	0
	CKS2 CKS1 CKS0	Clock select 2 to 0 • CKS2 = 0, CKS1 = 0, CKS0 = 0: ϕ/2 clock input selected for TCNT_0	Bit 2 Bit 1 Bit 0	CKS2 = 0 CKS1 = 0 CKS0 = 0
TCNT_0 *2		Timer counter • 8-bit up-counter	H'FFFF74	H'FF
RSTCSR *3		Reset control/status register	H'FFFF76	H'5F
	RSTE	Reset enable • RSTE = 0: No MCU internal reset when TCNT_0 overflow occurs • RSTE = 1: MCU internal reset when TCNT_0 overflow occurs	Bit 6	1

Table 11 Registers Used by FZMAIN() Function (cont.)

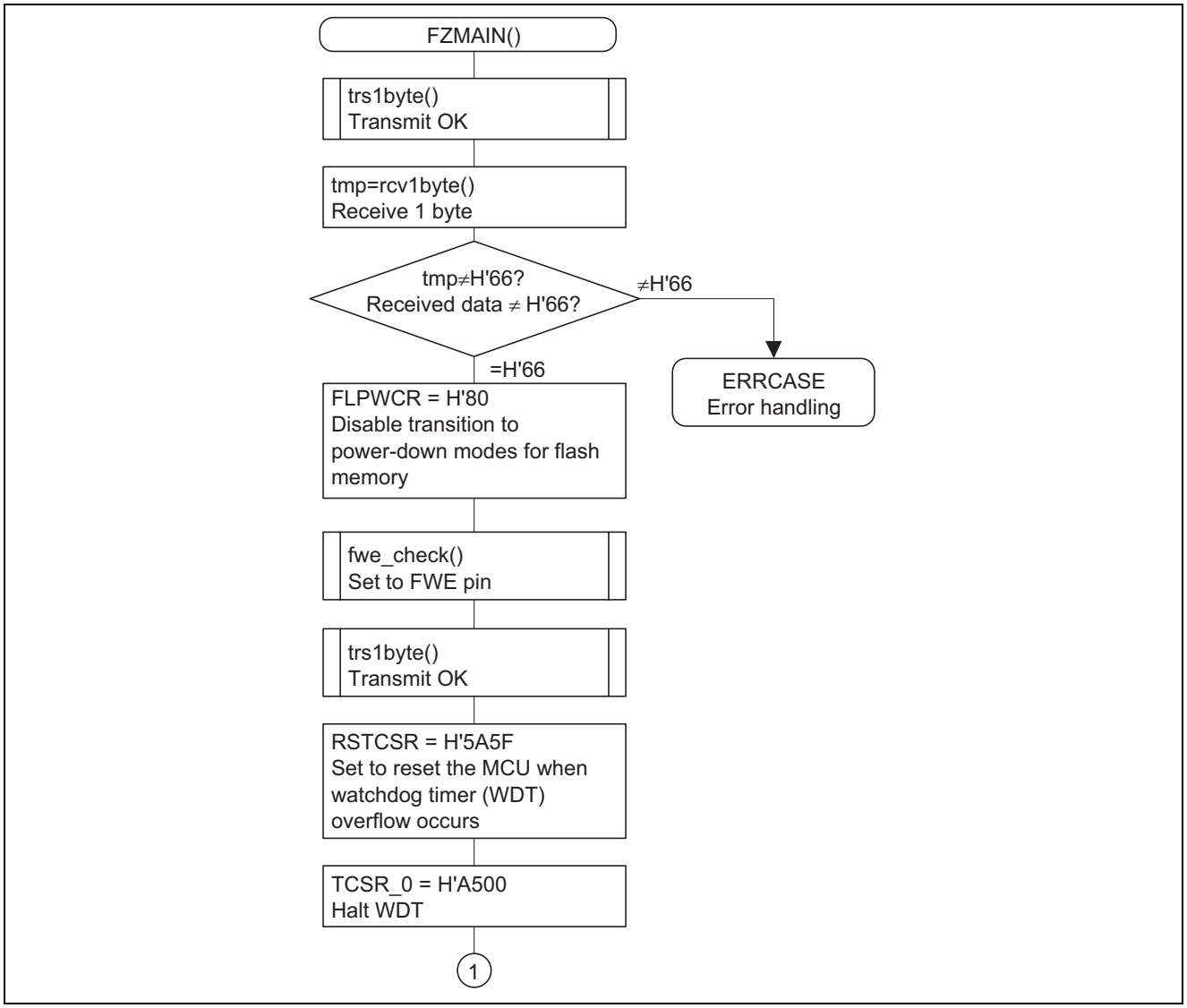
Register	Bit Name	Description	Address	Set Value
FLPWCR		Flash memory power control register	H'FFFFAC	H'80
	PDWND	Power-down disable • PDWND = 0: Transition to power-down modes for flash memory enabled • PDWND = 1: Transition to power-down modes for flash memory disabled	Bit 7	1

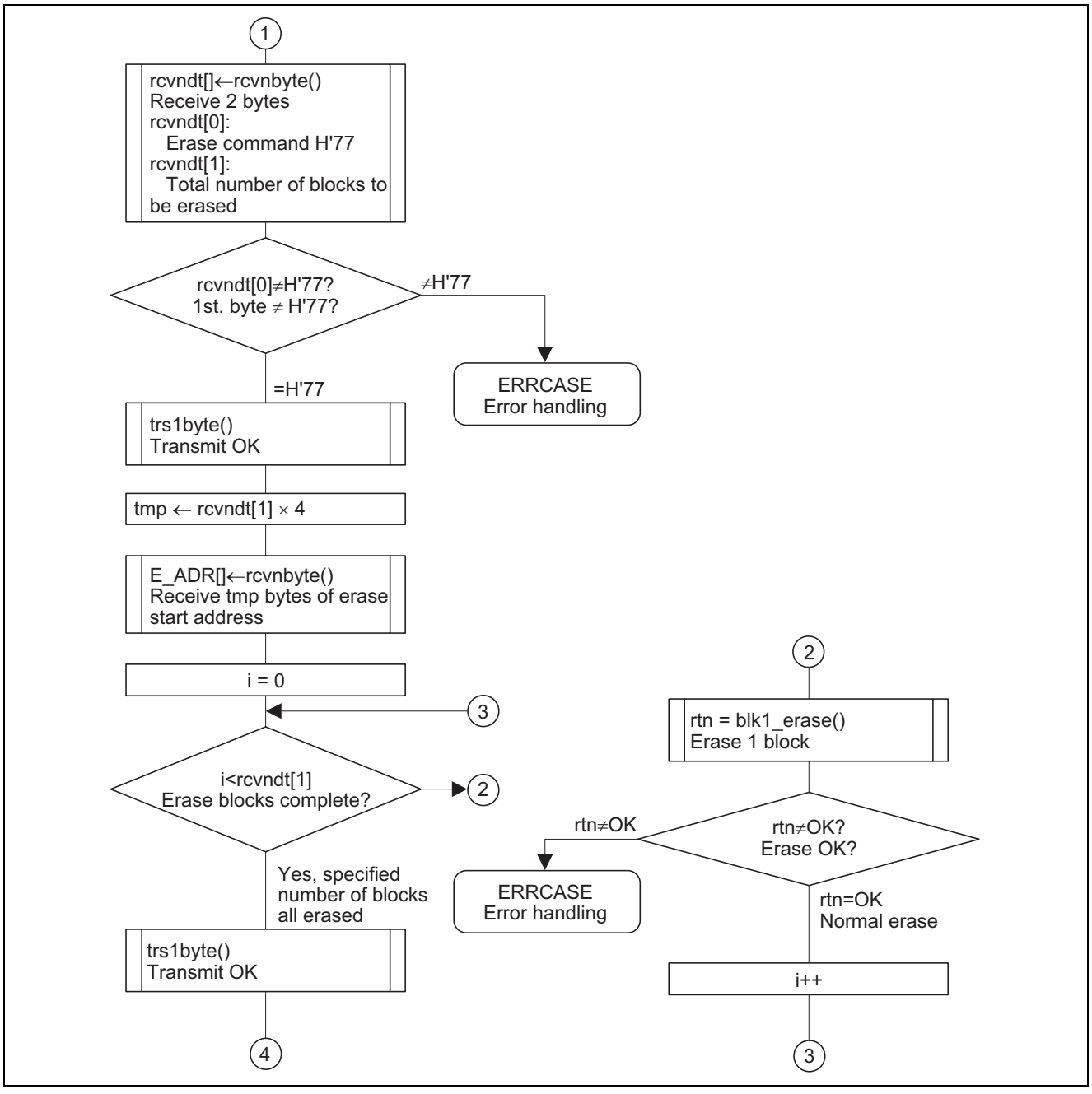
Notes: 1. The method for writing to TCSR_0 is different from that for general registers.
- Writing is accomplished by word transfer with H'FFFF74 as the target.
- The value of the upper byte is H'A5 and the lower byte is the programming data.
- In this function, the value written is as follows:
  TCSR_0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

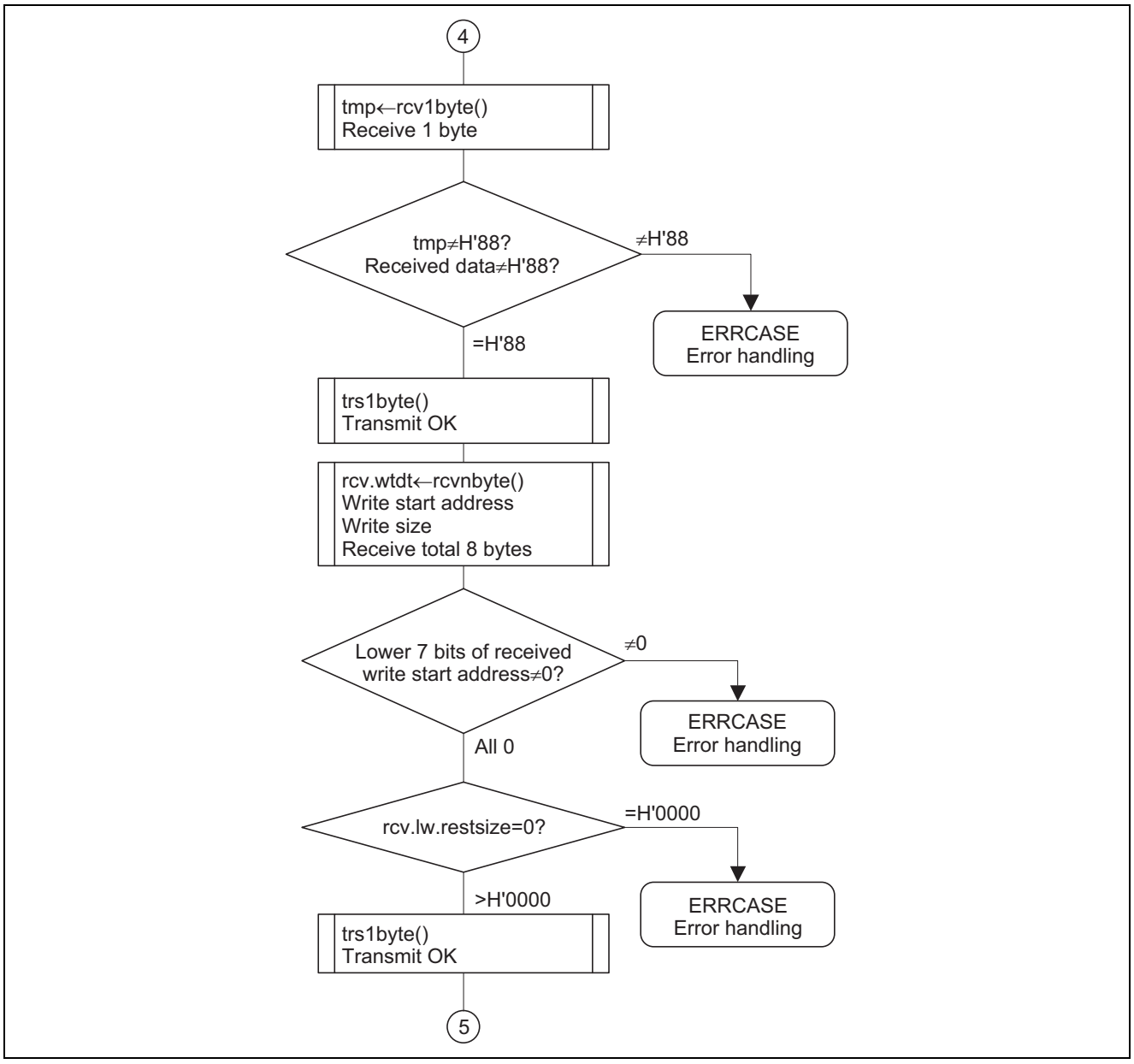
2. The method for writing to TCNT_0 is different from that for general registers.
- Writing is accomplished by word transfer with H'FFFF74 as the target.
- The value of the upper byte is H'5A and the lower byte is the programming data.
- In this function, the value written is as follows:
  TCNT_0 = H'5AFF

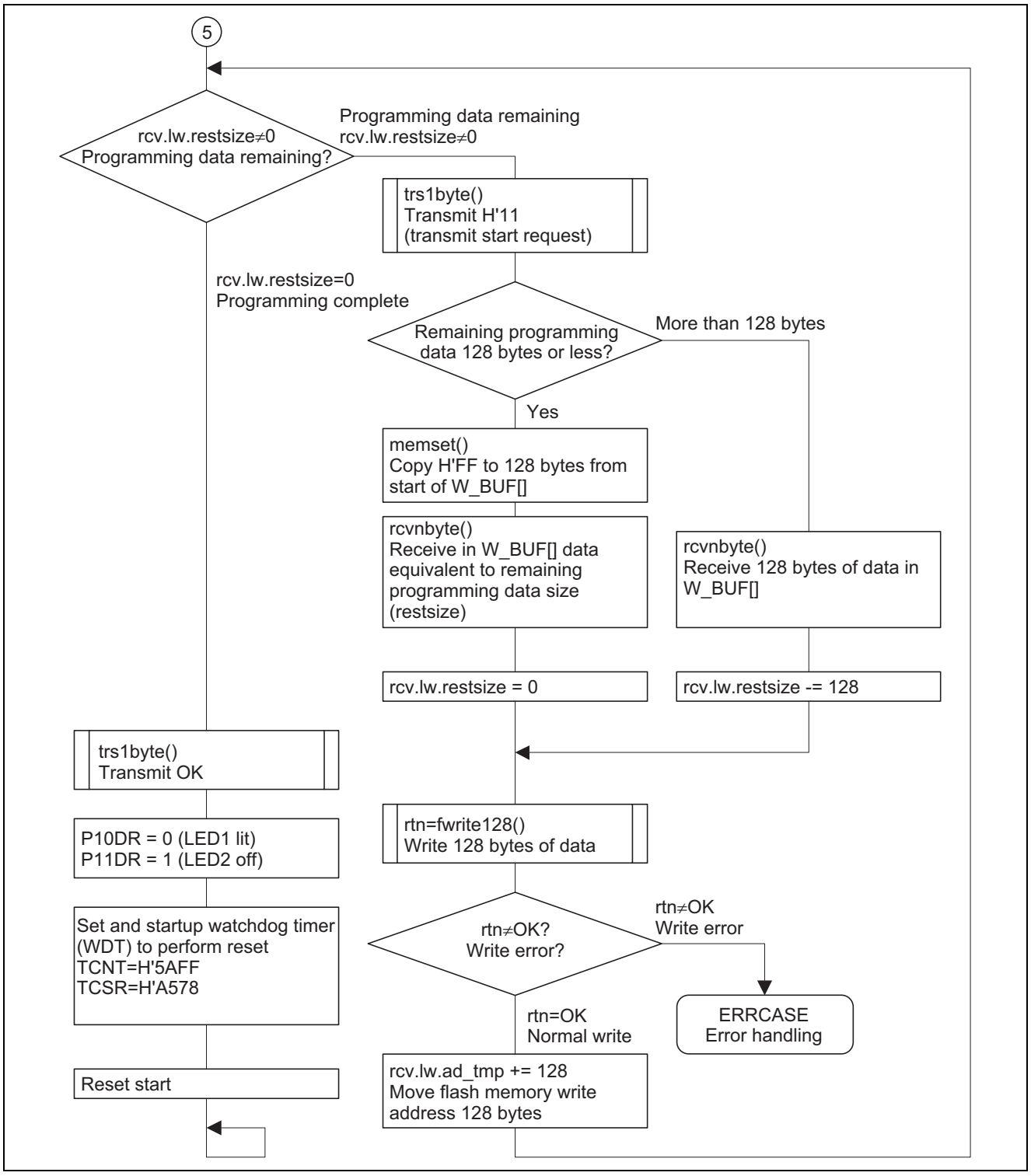
3. The method for writing to RSTCSR is different from that for general registers.
- Writing is accomplished by word transfer with H'FFFF76 as the target.
- When writing to the RSTE bit, the value of the upper byte is H'5A and the lower byte is the programming data.
- In this function, the value written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

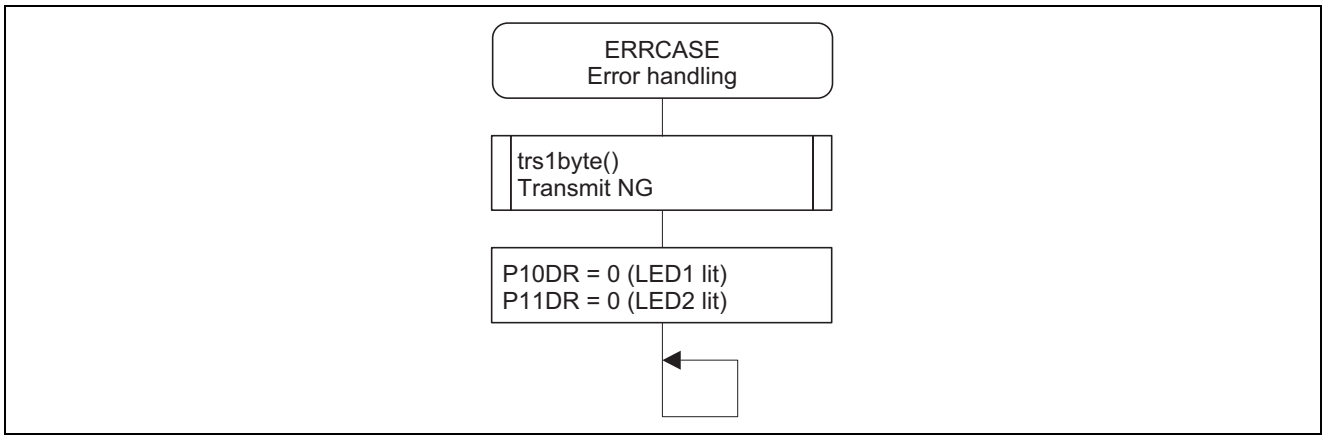
(g) Flowcharts




1
rcvndt[]←rcvnbyte()
Receive 2 bytes
rcvndt[0]:
Erase command H'77
rcvndt[1]:
Total number of blocks to be erased
rcvndt[0]≠H'77?
1st. byte ≠ H'77?
≠H'77
ERRCASE
Error handling
=H'77
trs1byte()
Transmit OK
tmp ← rcvndt[1] × 4
E_ADR[]←rcvnbyte()
Receive tmp bytes of erase start address
i = 0
3
i<rcvndt[1]
Erase blocks complete?
2
Yes, specified number of blocks all erased
trs1byte()
Transmit OK
4
2
rtn = blk1_erase()
Erase 1 block
rtn≠OK?
Erase OK?
rtn≠OK
ERRCASE
Error handling
rtn=OK
Normal erase
i++
3


4
tmp←rcv1byte()
Receive 1 byte
tmp≠H'88?
Received data≠H'88?
≠H'88
ERRCASE
Error handling
=H'88
trs1byte()
Transmit OK
rcv.wtdt←rcvnbyte()
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
trs1byte()
Transmit OK
5


5
rcv.lw.restsize≠0
Programming data remaining?
Programming data remaining
rcv.lw.restsize≠0
trs1byte()
Transmit H'11
(transmit start request)
rcv.lw.restsize=0
Programming complete
Remaining programming
data 128 bytes or less?
More than 128 bytes
Yes
memset()
Copy H'FF to 128 bytes from
start of W_BUF[]
rcvnbyte()
Receive in W_BUF[] data
equivalent to remaining
programming data size
(restsize)
rcvnbyte()
Receive 128 bytes of data in
W_BUF[]
rcv.lw.restsize = 0
rcv.lw.restsize -= 128
trs1byte()
Transmit OK
rtn=fwrite128()
Write 128 bytes of data
P10DR = 0 (LED1 lit)
P11DR = 1 (LED2 off)
rtn≠OK?
Write error?
rtn≠OK
Write error
Set and startup watchdog timer
(WDT) to perform reset
TCNT=H'5AFF
TCSR=H'A578
rtn=OK
Normal write
ERRCASE
Error handling
Reset start
rcv.lw.ad_tmp += 128
Move flash memory write
address 128 bytes



(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

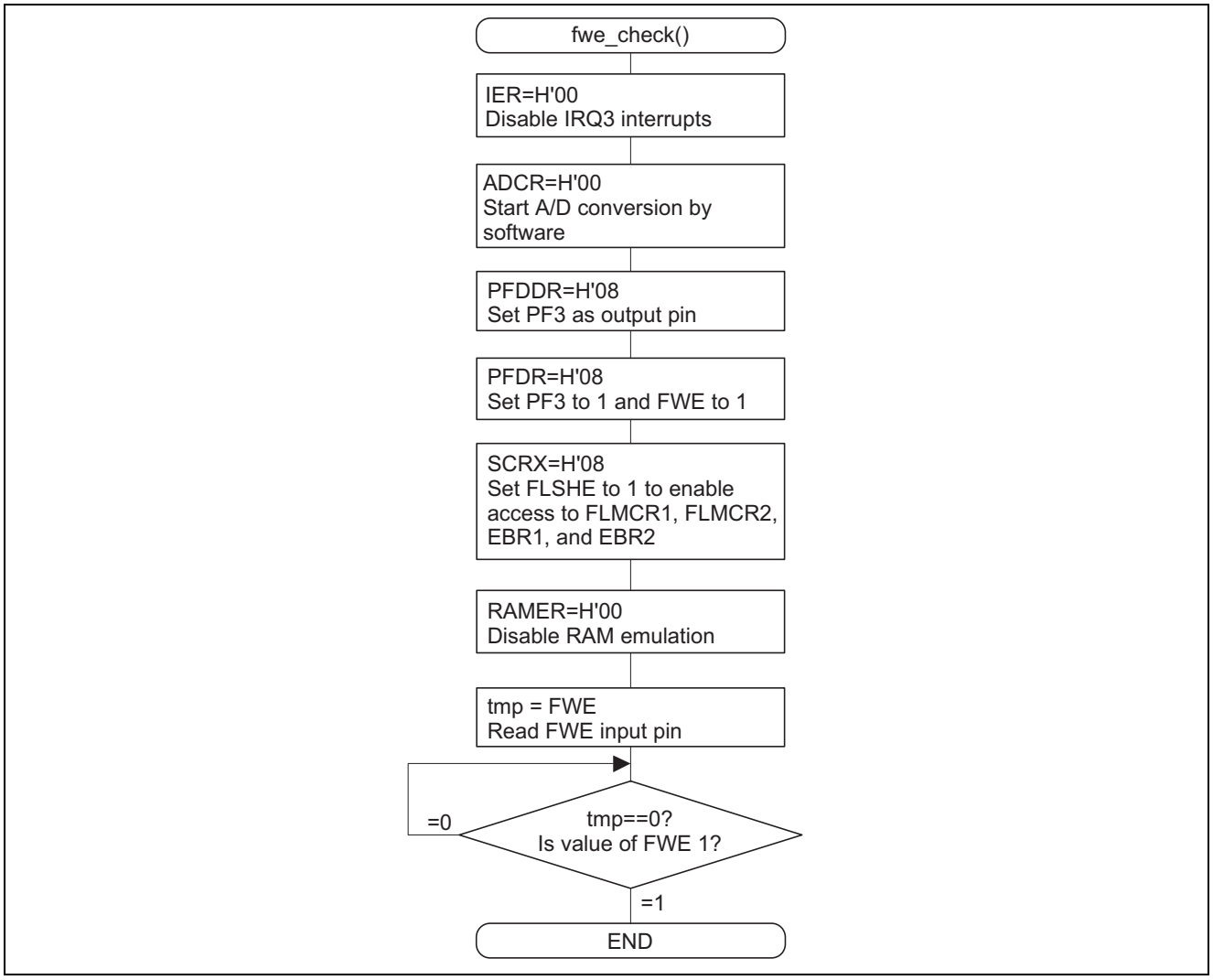
Table 12 Registers Used by fwe_check() Function

Register	Bit Name	Description	Address	Set Value
SCRX		Serial control register X	H'FFFDB4	H'08
	FLSHE	Flash memory control register enable • FLSHE = 0: Disables access to flash memory control registers (FLMCR1, FLMCR2, EBR1, EBR2) • FLSHE = 1: Enables access to flash memory control registers (FLMCR1, FLMCR2, EBR1, EBR2)	Bit 3	1
IER		IRQ enable register	H'FFFF14	H'00
	IRQ3E	IRQ3 enable • IRQ3E = 0: Disables IRQ3 interrupt requests • IRQ3E = 1: Enables IRQ3 interrupt requests	Bit 3	0
PFDDR		Port F data direction register	H'FFFF3E	H'08
	PF3DDR	Port F3 data direction register • PF3DDR = 0: Set PF3 to input • PF3DDR = 1: Set PF3 to output	Bit 3	1

Table 12 Registers Used by fwe_check() Function (cont.)

Register	Bit Name	Description	Address	Set Value
RAMER		RAM emulation register	H'FFFEDB	H'00
	RAMS	RAM select • RAMS = 0: Disables RAM emulation • RAMS = 1: Enables RAM emulation	Bit 3	0
PFDR		Port F data register	H'FFFF0E	H'08
	PF3DR	Port F3 data register • PF3DR = 0: PF3 output level low • PF3DR = 1: PF3 output level high	Bit 3	1
ADCR		A/D control register	H'FFFF99	H'00
	TRGS1	Timer trigger select 1 and 0	Bit 7	TRGS1 = 0
	TRGS0	• TRGS1 = 0, TRGS0 = 0: Starts A/D conversion by software	Bit 6	TRGS0 = 0
FLMCR1		Flash memory control register 1	H'FFFEA8	—
	FWE	Flash write enable • FWE = 0: FWE input pin outputs low level • FWE = 1: FWE input pin outputs high level	Bit 7	—

(g) Flowchart


fwe_check()
IER=H'00
Disable IRQ3 interrupts
ADCR=H'00
Start A/D conversion by software
PFDDR=H'08
Set PF3 as 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

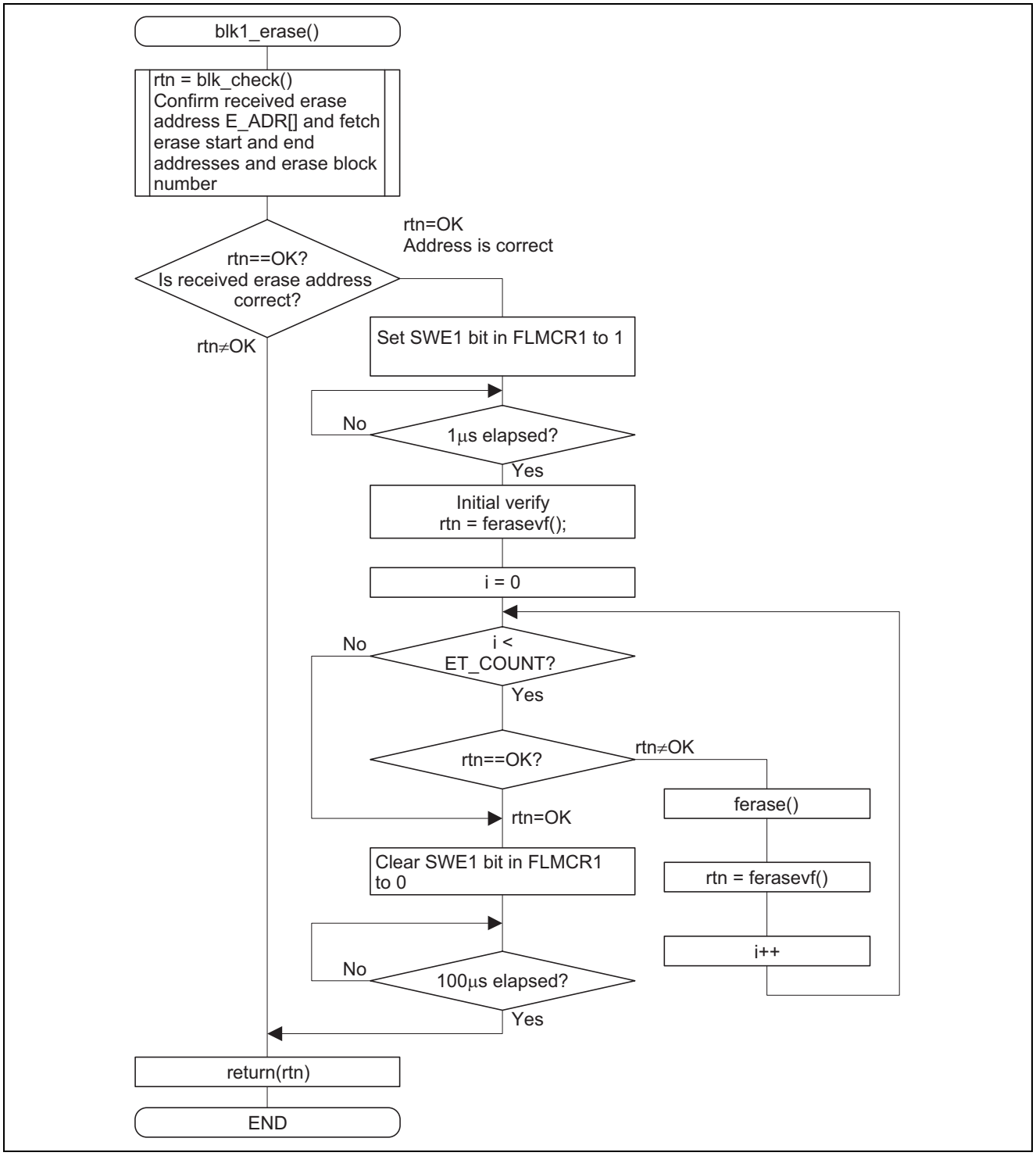
blk_check(): Determines the bit number of the block to be erased from the erase start address
ferase(): Erases designated blocks
ferasevf(): Verifies erase of designated blocks

(f) Internal Registers Used

Table 13 Registers Used by blk1_erase() Function

Register	Bit Name	Description	Address	Set Value
FLMCR1		Flash memory control register 1	H'FFFFA8	—
	SWE1	Software write enable • SWE1 = 0: Disable flash memory programming/erasing • SWE1 = 1: Enables flash memory programming/erasing	Bit 6	1

(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?
Is received erase address correct?
rtn=OK
Address is correct
rtn≠OK
Set SWE1 bit in FLMCR1 to 1
No
1µs elapsed?
Yes
Initial verify
rtn = ferasevf();
i = 0
No
i < ET_COUNT?
Yes
rtn==OK?
rtn≠OK
ferase()
rtn=OK
Clear SWE1 bit in FLMCR1 to 0
rtn = ferasevf()
i++
No
100µs elapsed?
Yes
return(rtn)
END

(4) blk_check() Function

(a) Specifications

```
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

```
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        */
    };
```

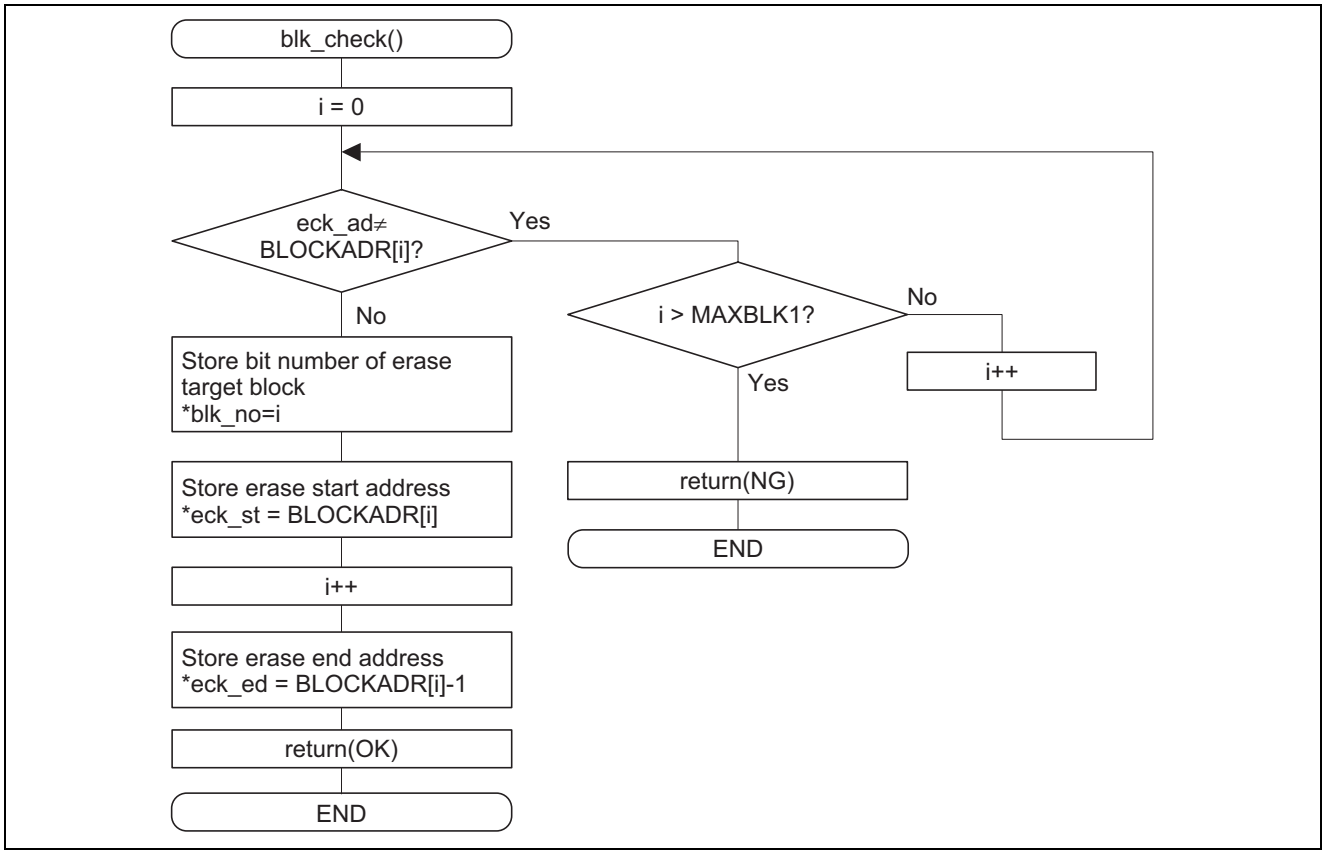
(e) Subroutines Used

None

(f) Internal Registers Used

None

(g) Flowchart



(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

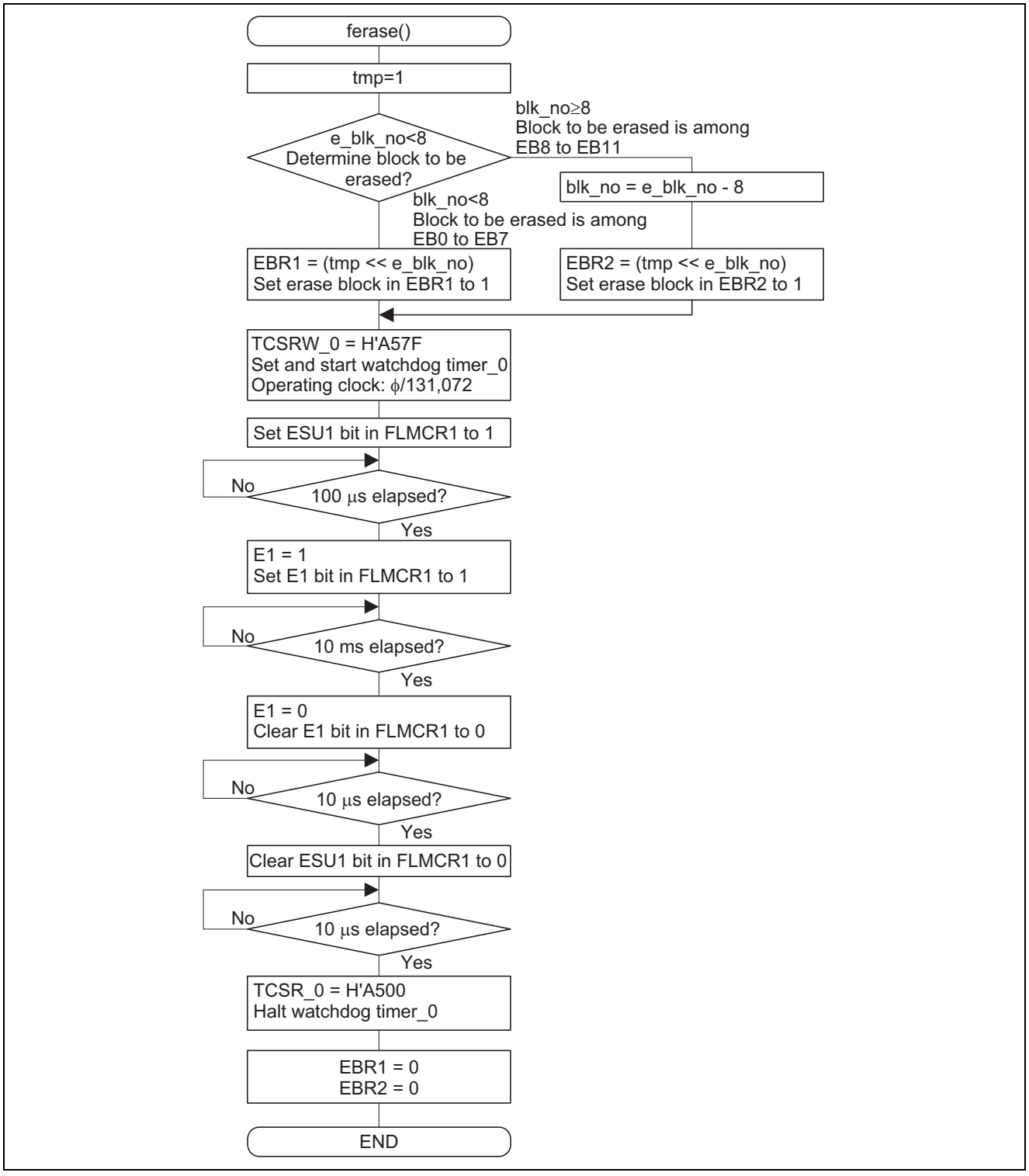
Table 14 Registers Used by ferase() Function

Register	Bit Name	Description	Address	Set Value
FLMCR1		Flash memory control register 1	H'FFFFA8	—
	ESU1	Erase setup • ESU1 = 0: Clears erase setup state • ESU1 = 1 when FWE1 = 1 and SWE1 = 1: Enters erase setup state	Bit 5	1
	E1	Erase • E1 = 0: Clears the erase mode • E1 = 1 when SWE1 = 1 and ESU1 = 1: : Enters the erase mode	Bit 1	1
EBR1	EB7 : : EB0	Erase block register 1 • Setting a bit from EB7 to EB0 to 1 enables erasing of the corresponding block of flash memory	H'FFFFAA	—
EBR2	EB11 EB10 EB9 EB8	Erase block register 2 • Setting a bit from EB11 to EB8 to 1 enables erasing of the corresponding block of flash memory	H'FFFFAB	—
TCSR_0 *1		Timer control/status register_0	H'FFFF74	H'7F
	OVF	Overflow flag • OVF = 0: No TCNT_0 overflow • OVF = 1: TCNT_0 overflow	Bit 7	0
	WT/ $\overline{IT}$	Timer mode select • WT/ $\overline{IT}$ = 0: Interval timer • WT/ $\overline{IT}$ = 1: Watchdog timer	Bit 6	1
	TME	Timer enable • TME = 0: TCNT_0 count start • TME = 1: TCNT_0 count halt	Bit 5	1
	CKS2 CKS1 CKS0	Clock select 2 to 0 • CKS2 = 1, CKS1 = 1, CKS0 = 1: ϕ/131,072 clock input selected for TCNT_0	Bit 2 Bit 1 Bit 0	CKS2 = 1 CKS1 = 1 CKS0 = 1

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'A57F

(g) Flowchart


ferase()
tmp=1
e_blk_no<8
Determine block to be erased?
blk_no≥8
Block to be erased is among EB8 to EB11
blk_no = e_blk_no - 8
blk_no<8
Block to be erased is among EB0 to EB7
EBR1 = (tmp << e_blk_no)
Set erase block in EBR1 to 1
EBR2 = (tmp << e_blk_no)
Set erase block in EBR2 to 1
TCSRW_0 = H'A57F
Set and start watchdog timer_0
Operating clock: ϕ/131,072
Set ESU1 bit in FLMCR1 to 1
No
100 μs 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 μs elapsed?
Yes
Clear ESU1 bit in FLMCR1 to 0
No
10 μs elapsed?
Yes
TCSR_0 = H'A500
Halt watchdog timer_0
EBR1 = 0
EBR2 = 0
END

(6) ferasevf() Function

(a) Specifications

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

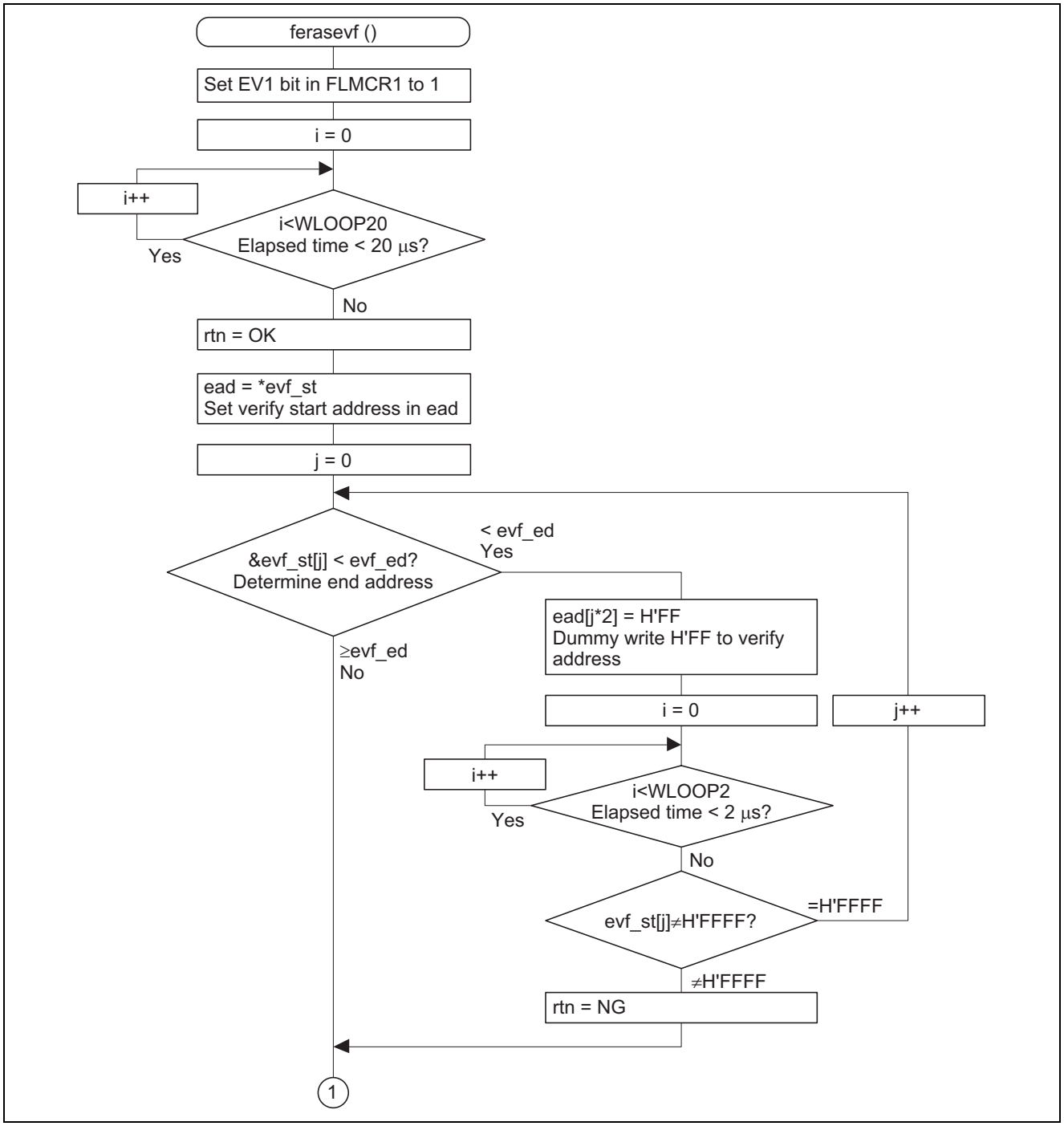
None

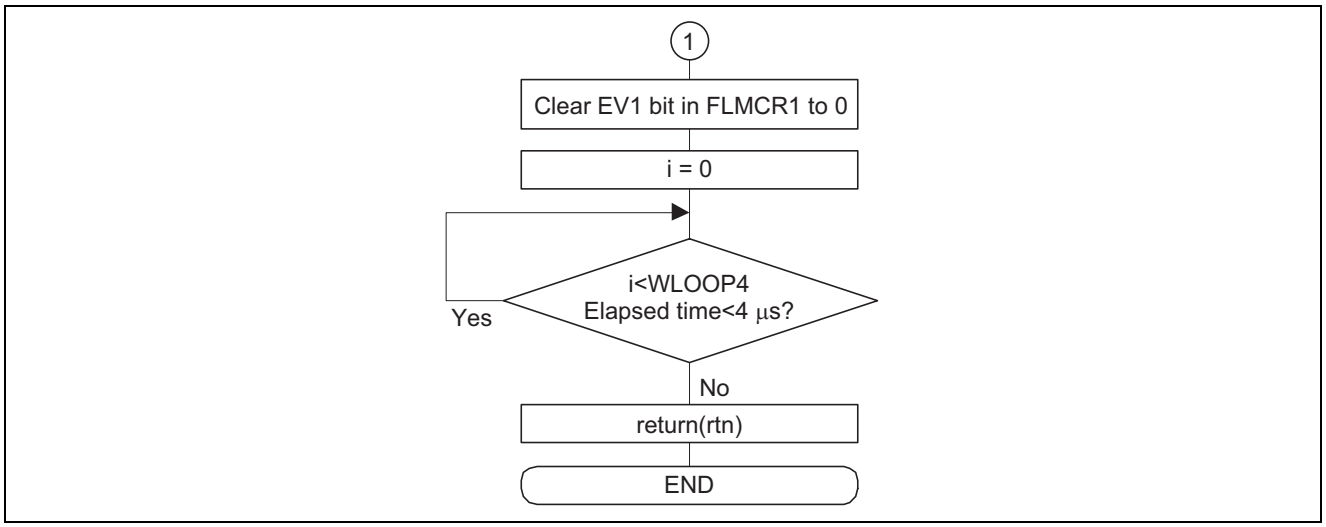
(f) Internal Registers Used

Table 15 Registers Used by ferasevf() Function

Register	Bit Name	Description	Address	Set Value
FLMCR1		Flash memory control register 1	H'FFFFA8	—
	EV1	Erase verify • EV1 = 0: Cancels the erase verify mode • EV1 = 0: Enters the erase verify mode	Bit 3	1

(g) Flowcharts





(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

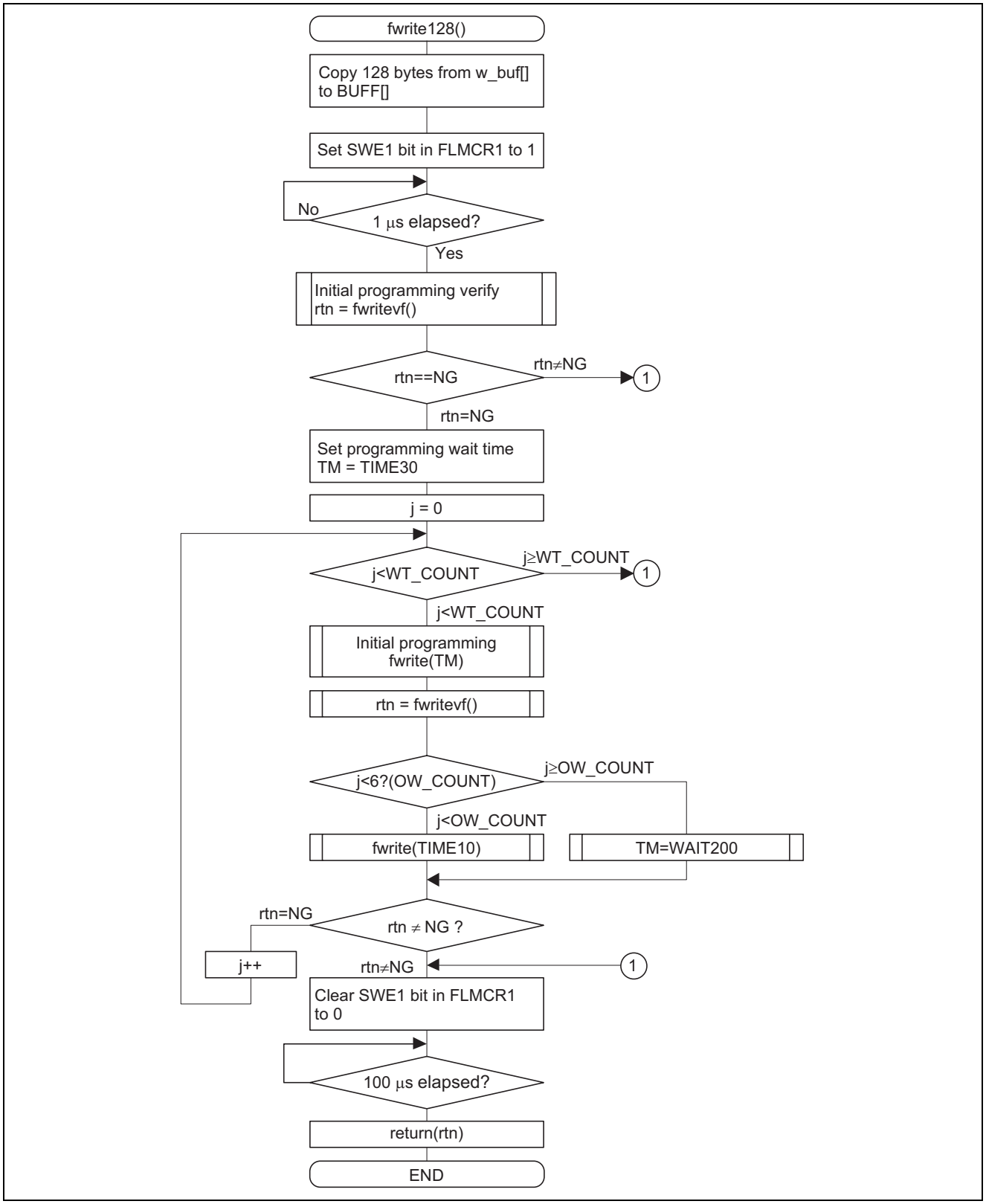
fwrite: Writes to the target address
fwritevf: Verifies the target address, creates overwrite data

(f) Internal Registers Used

Table 16 Registers Used by fwrite128() Function

Register	Bit Name	Description	Address	Set Value
FLMCR1		Flash memory control register 1	H'FFFFA8	—
	SWE1	Software write enable • SWE1 = 0: Disables flash memory programming/erasing • SWE1 = 1: Enables flash memory programming/erasing	Bit 6	1

(g) Flowchart


fwrite128()
Copy 128 bytes from w_buf[] to BUFF[]
Set SWE1 bit in FLMCR1 to 1
No
1 µs elapsed?
Yes
Initial programming verify
rtn = fwritevf()
rtn==NG
rtn≠NG
1
rtn=NG
Set programming wait time
TM = TIME30
j = 0
j<WT_COUNT
j≥WT_COUNT
1
j<WT_COUNT
Initial programming
fwrite(TM)
rtn = fwritevf()
j<6?(OW_COUNT)
j≥OW_COUNT
j<OW_COUNT
fwrite(TIME10)
TM=WAIT200
rtn=NG
rtn ≠ NG ?
j++
rtn≠NG
1
Clear SWE1 bit in FLMCR1
to 0
100 µs elapsed?
return(rtn)
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

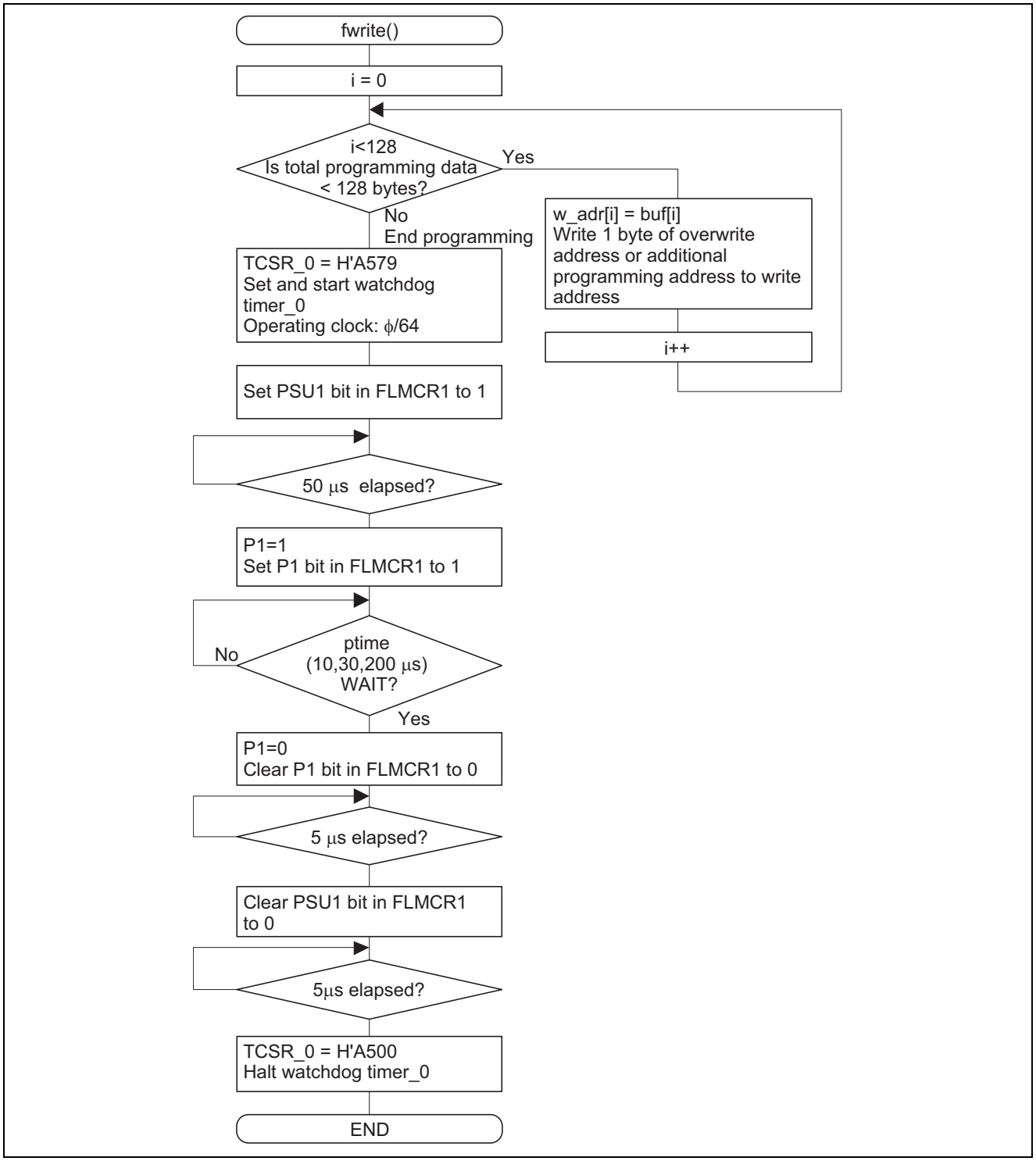
Table 17 Registers Used by fwrite() Function

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

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

(g) Flowchart


fwrite()
i = 0
i<128
Is total programming data
< 128 bytes?
Yes
No
End programming
w_adr[i] = buf[i]
Write 1 byte of overwrite
address or additional
programming address to write
address
i++
TCSR_0 = H'A579
Set and start watchdog
timer_0
Operating clock: ϕ/64
Set PSU1 bit in FLMCR1 to 1
50 µs elapsed?
P1=1
Set P1 bit in FLMCR1 to 1
No
ptime
(10,30,200 µs)
WAIT?
Yes
P1=0
Clear P1 bit in FLMCR1 to 0
5 µs elapsed?
Clear PSU1 bit in FLMCR1
to 0
5µs elapsed?
TCSR_0 = H'A500
Halt watchdog timer_0
END

(9) fwritevf() Function

(a) Specifications

```
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

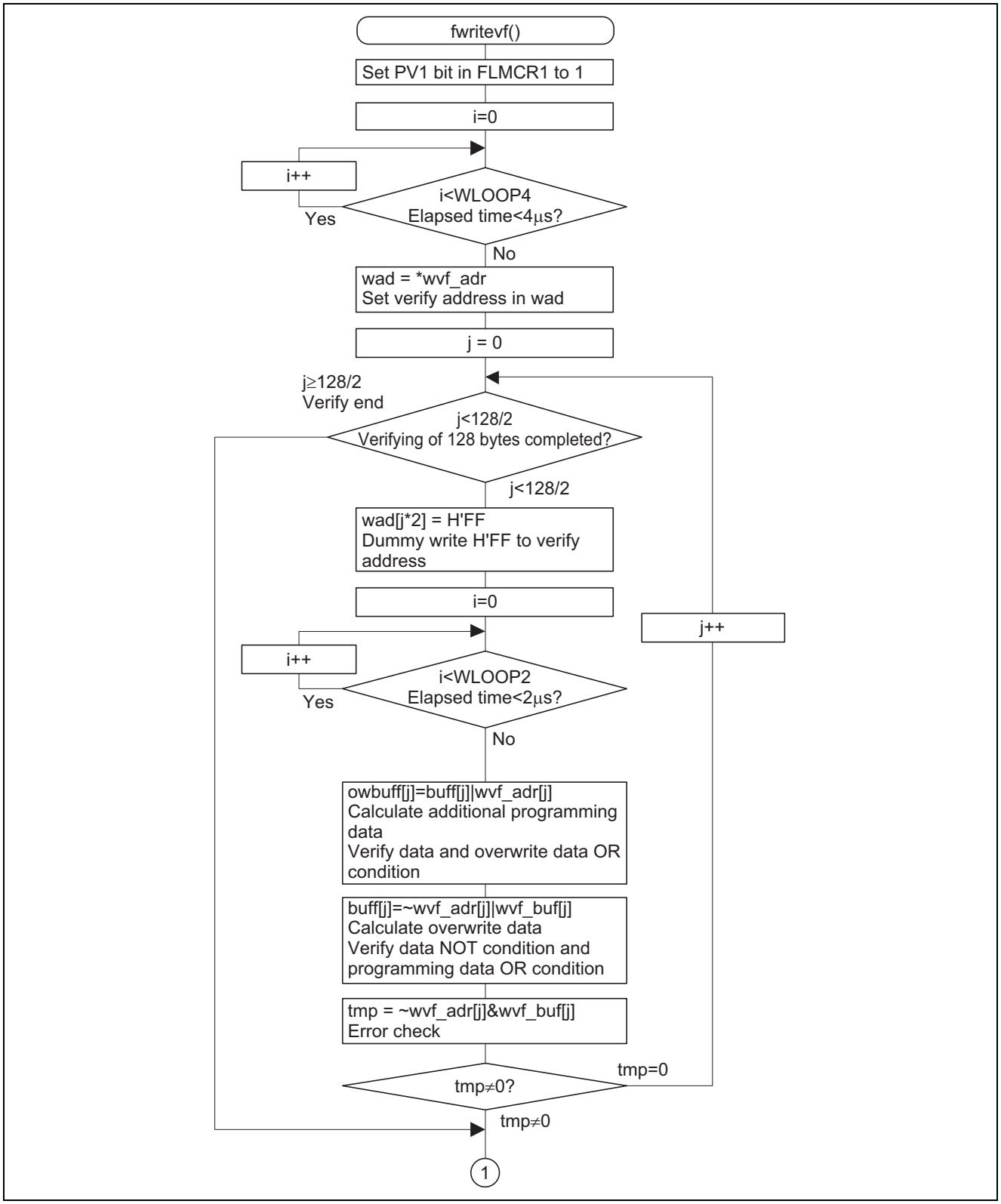
None

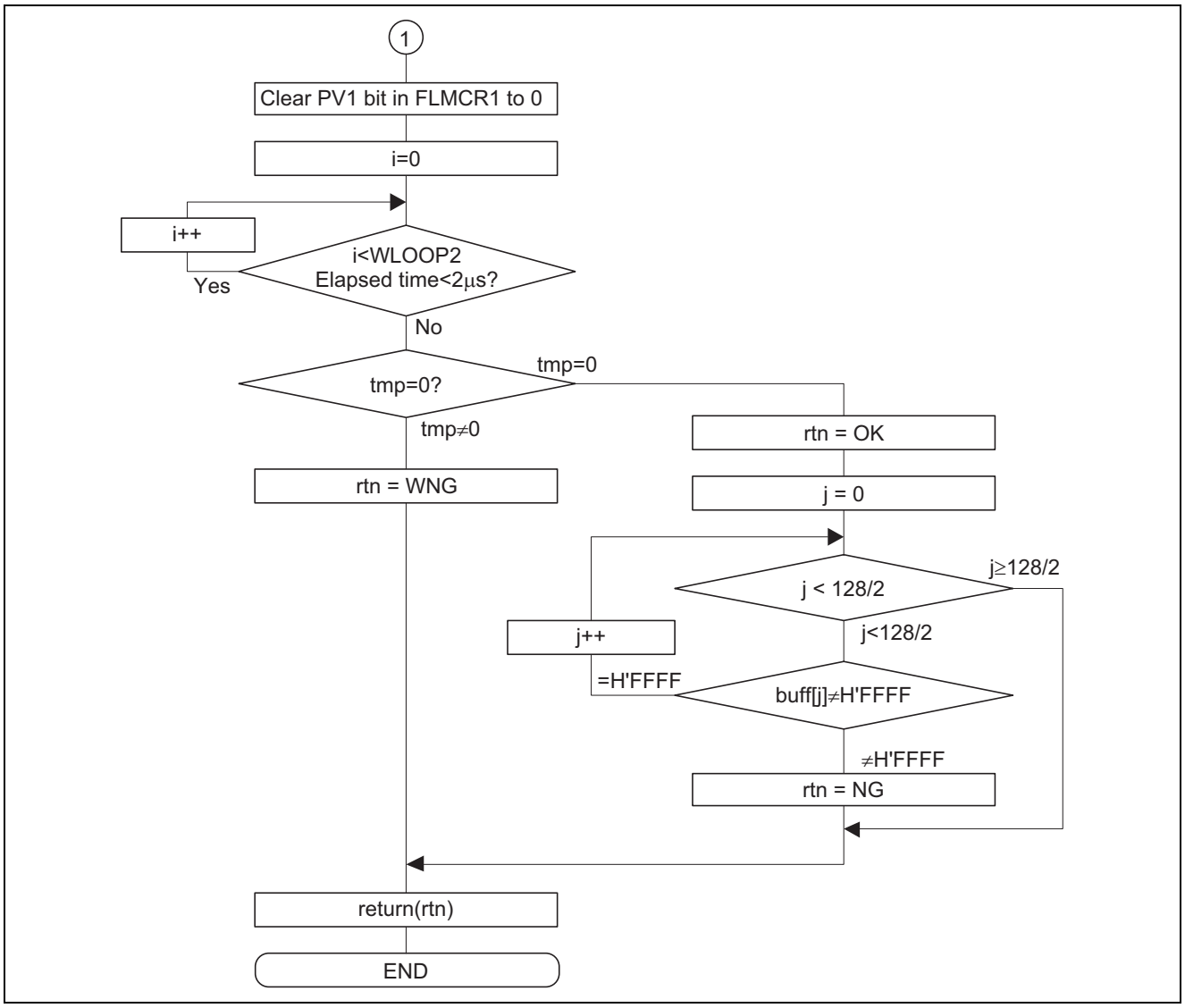
(f) Internal Registers Used

Table 18 Registers Used by fwritevf() Function

Register	Bit Name	Description	Address	Set Value
FLMCR1		Flash memory control register 1	H'FFFFA8	—
	PV1	Program-verify • PV1 = 0: Cancels the program-verify mode • PV1 = 1: Enters the program-verify mode	Bit 2	1

(g) Flowcharts




1
Clear PV1 bit in FLMCR1 to 0
i=0
i++
i<WLOOP2
Elapsed time<2μs?
Yes
No
tmp=0?
tmp=0
tmp≠0
rtn = WNG
rtn = OK
j = 0
j < 128/2
j≥128/2
j++
j<128/2
=H'FFFF
buff[j]≠H'FFFF
≠H'FFFF
rtn = NG
return(rtn)
END

8. Asynchronous Serial Communication Program

8.1 Hierarchy

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

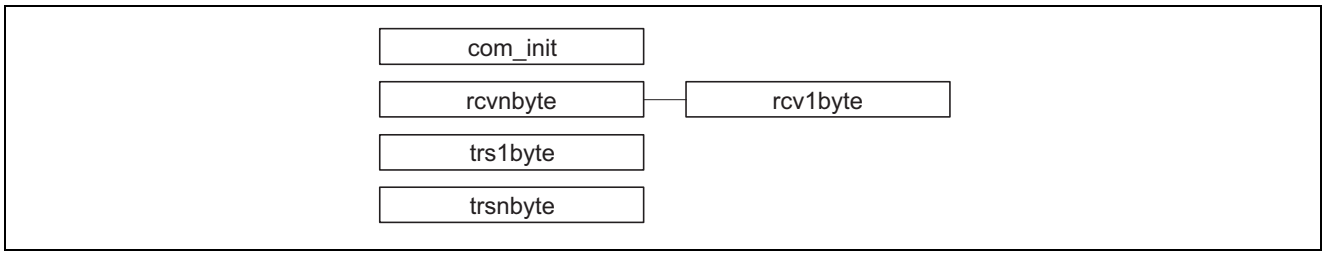


Figure 21 Hierarchy of Asynchronous Serial Communication Program

8.2 List of Functions

Table 19 Functions of Asynchronous Serial Communication Program

Description	Function
com_init	Initializes asynchronous serial communication
rcv1byte	Receives 1 byte of data
rcvnbyte	Receives n bytes of data
trs1byte	Transmits 1 byte of data
trsnbyte	Transmits n bytes of data

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

Register	Bit Name	Description	Address	Set Value
MSTPCRB		Module stop control register B	H'FFFDE9	H'7F
	MSTPB7	Serial communication interface 0 • MSTPB7 = 0: Clear module stop mode for SCI_0 • MSTPB7 = 1: Enter module stop mode for SCI_0	Bit 7	0
SMR_0		Serial mode register 0	H'FFFF78	H'00
	C/ $\overline{A}$	Communication mode • C/ $\overline{A}$ = 0: Asynchronous communication mode • C/ $\overline{A}$ = 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/ $\overline{E}$	Parity mode • O/ $\overline{E}$ = 0: Even parity for appending and checking of parity bits • O/ $\overline{E}$ = 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 CKS0	Clock select 1 and 0 • CKS1 = 0, CKS0 = 0: ϕclock selected as clock source for internal baud rate generator	Bit 1 Bit 0	CKS1 = 0 CKS0 = 0

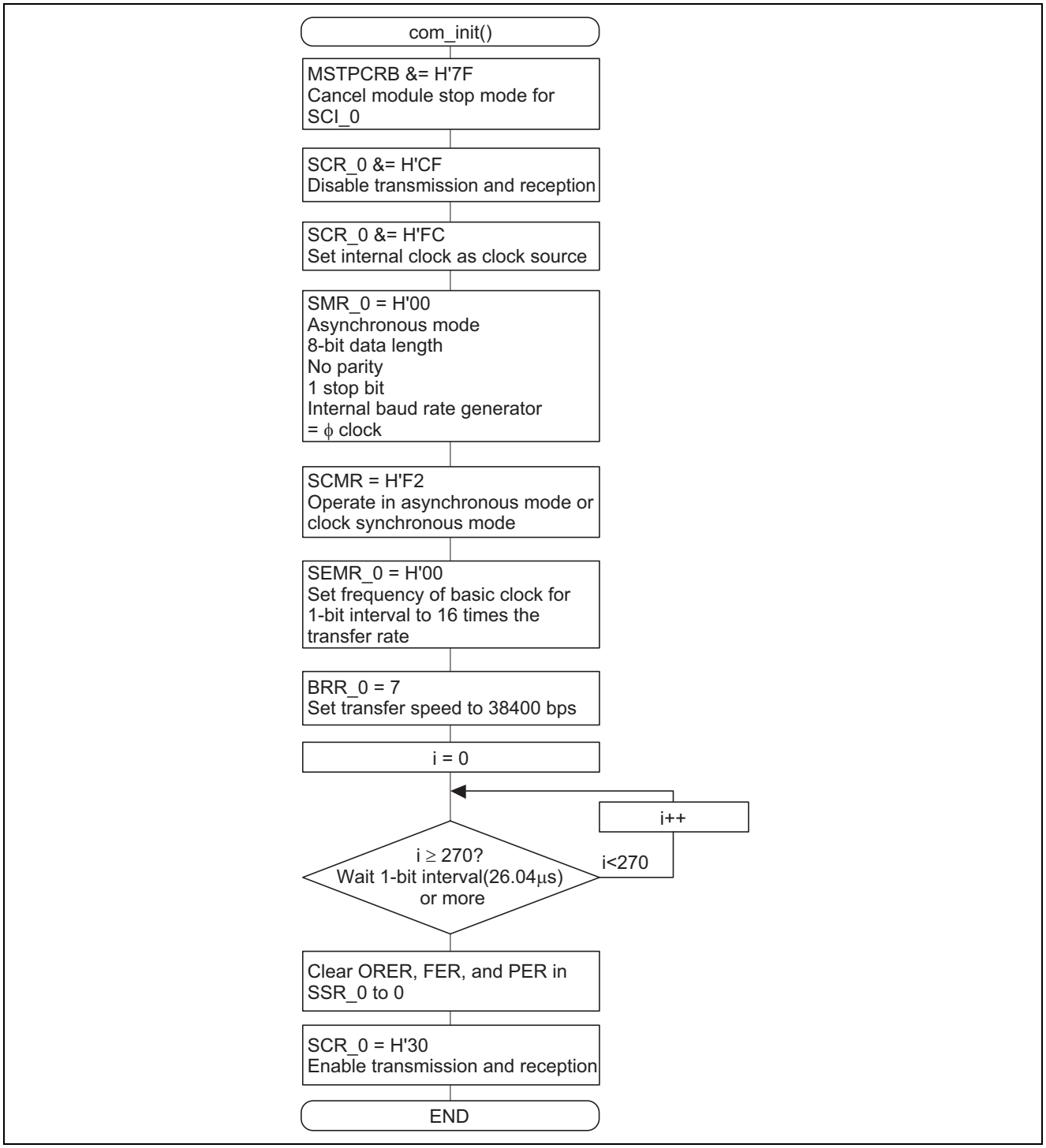
Table 20 Registers Used by com_init() Function (cont.)

Register	Bit Name	Description	Address	Set Value
BRR_0		Bit rate register 0 • BRR = H'09: Selects 31250-bps transmit bit rate matching operating clock selected by CKS1 and CKS0 in SMR_0	H'FFFF79	H'09
SCR_0		Serial control register	H'FFFF7A	—
	TE	Transmit enable • TE = 0: Disables transmit operation • TE = 1: Enables transmit operation	Bit 5	0
	RE	Receive enable • RE = 0: Disables receive operation • RE = 1: Enables receive operation	Bit 4	0
	CKE1	Clock enable 1 and 0	Bit 1	CKE1 = 0
	CKE0	• CKE1 = 0, CKE0 = 0: Selects internal clock as clock source in asynchronous communication mode and sets SCK0 as I/O port	Bit 0	CKE0 = 0
SSR_0		Serial status register 0	H'FFFF7C	—
	TDRE	Transmit data register empty • TDRE = 0: Transmit data written to TDR_0 has not been transferred to TSR_0 • TDRE = 1: Transmit data has not been written to TDR_0 or transmit data written to TDR_0 has been transferred to TSR_0	Bit 7	—
	RDRF	Receive data register full • RDRF = 0: No received data stored in RDR_0 • RDRF = 1: Received data stored in RDR_0	Bit 6	—
	ORER	Overrun error • ORER = 0: Indicates reception is in progress or has completed • ORER = 1: Indicates an overrun error occurred during reception	Bit 5	0
	FER	Framing error • FER = 0: Indicates reception is in progress or has completed • FER = 1: Indicates a framing error occurred during reception	Bit 4	0
	PER	Parity error • PER_0 = 0: Indicates reception is in progress or has completed • PER_0 = 1: Indicates a parity error occurred during reception	Bit 3	0
	TEND	Transmit end • TEND_0 = 0: Indicates transmission is in progress • TEND_0 = 1: Indicates transmission has ended	Bit 2	—

Table 20 Registers Used by com_init() Function (cont.)

Register	Bit Name	Description	Address	Set Value
SCMR		Smart card mode register	H'FFFF7E	H'F2
	SMIF	Smart card interface mode select • SMIF = 0: Normal asynchronous mode or clock synchronous mode • SMIF = 1: Smart card interface mode	Bit 0	0
SEMR_0		Serial expansion mode register 0	H'FFFDF8	H'00
	ABCS	Asynchronous basic clock select • ABCS = 0: Frequency of basic clock for 1-bit interval is 16 times the transfer rate in asynchronous mode • ABCS = 1: Frequency of basic clock for 1-bit interval is 8 times the transfer rate in asynchronous mode	Bit 3	0
	ACS2	Asynchronous clock source select • ACS2 = 0, ACS1 = 0, ACS0 = 0: External clock input selected as asynchronous clock source	Bit 2	ACS2=0
	ACS1		Bit 1	ACS1=0
	ACS0		Bit 0	ACS0=0

(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
= ϕ 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++
i ≥ 270?
Wait 1-bit interval(26.04µs) or more
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

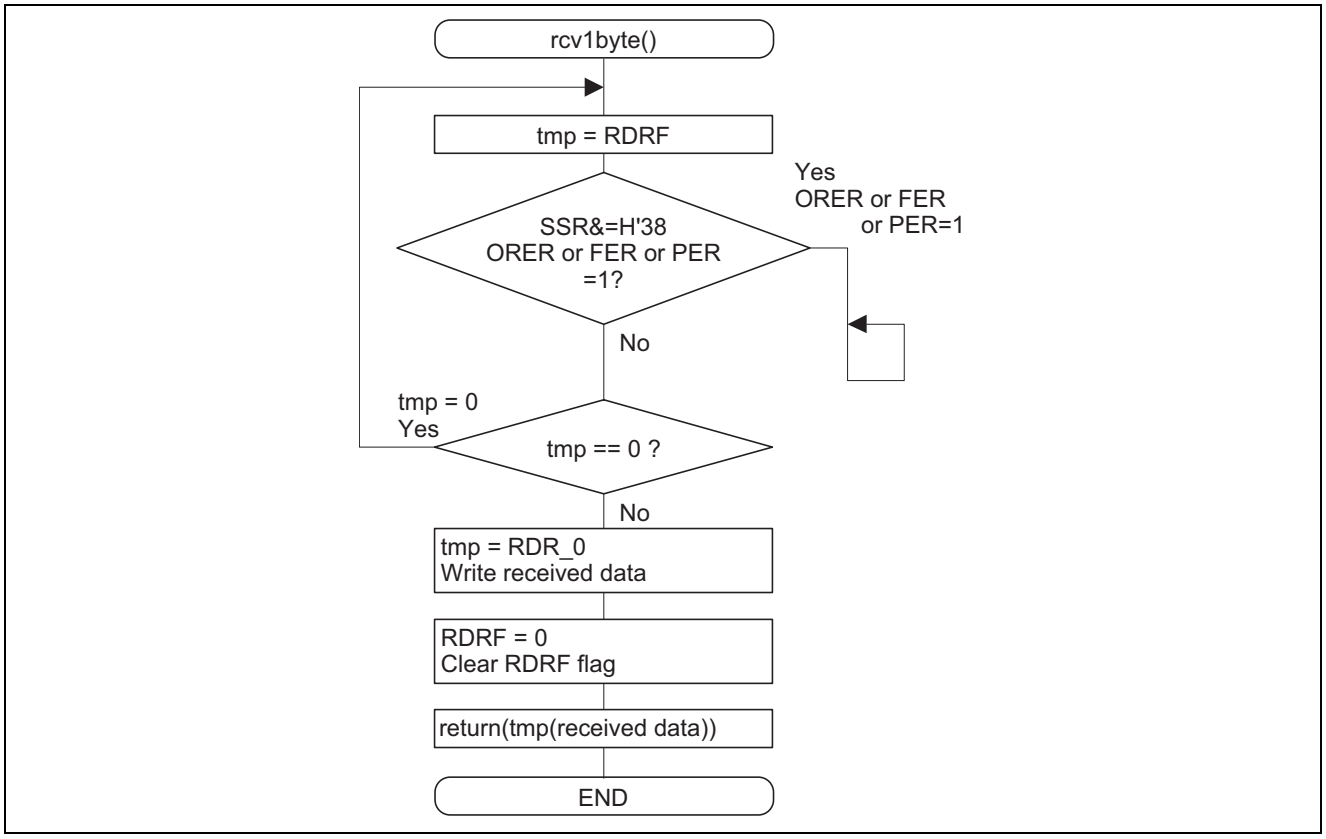
None

(f) Internal Registers Used

Table 21 Registers Used by rcv1byte() Function

Register	Bit Name	Description	Address	Set Value
SSR_0		Serial status register 0	H'FFFF7C	—
	RDRF	Receive data register full • RDRF = 0: No received data stored in RDR_0 • RDRF = 1: Received data stored in RDR_0	Bit 6	—
	ORER	Overrun error • ORER = 0: Indicates reception is in progress or has completed • ORER = 1: Indicates an overrun error occurred during reception	Bit 5	—
	FER	Framing error • FER = 0: Indicates reception is in progress or has completed • FER = 1: Indicates a framing error occurred during reception	Bit 4	—
	PER	Parity error • PER_0 = 0: Indicates reception is in progress or has completed • PER_0 = 1: Indicates a parity error occurred during reception	Bit 3	—
RDR_0		Receive data register 0 • 8-bit register that stores received data	H'FFFF7D	—

(g) Flowchart


rcv1byte()
tmp = RDRF
Yes
ORER or FER
or PER=1
SSR&=H'38
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

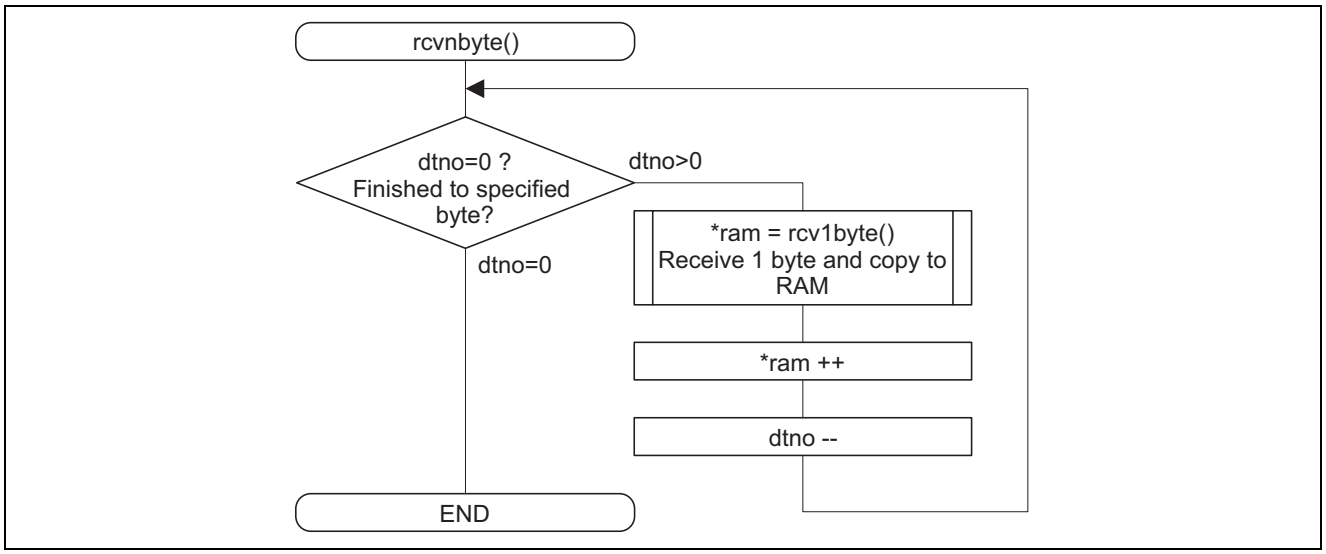
(e) Subroutines Used

rcv1byte: Receives 1 byte of asynchronous serial data

(f) Internal Registers Used

None

(g) Flowchart



(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

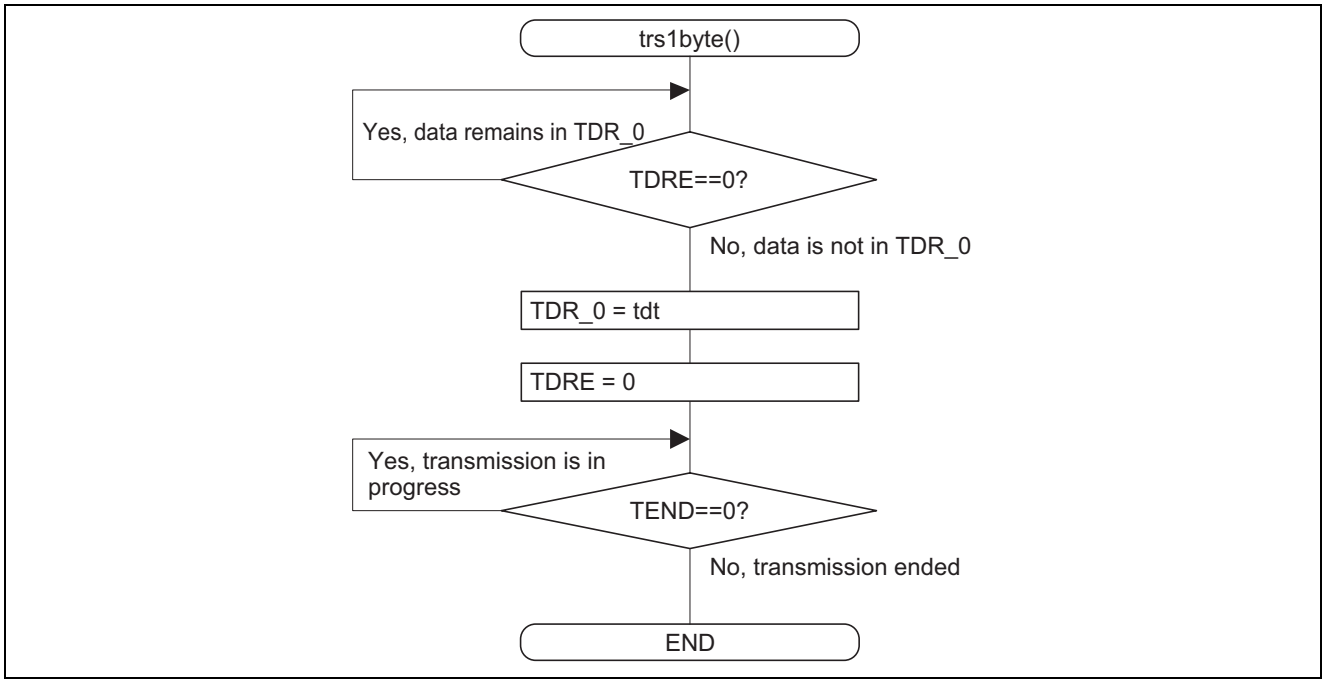
None

(f) Internal Registers Used

Table 22 Registers Used by trs1byte() Function

Register	Bit Name	Description	Address	Set Value
TDR_0		Transmit data register 0 • 8-bit register that stores transmit data	H'FFFF7B	—
SSR_0		Serial status register 0	H'FFFF7C	—
	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

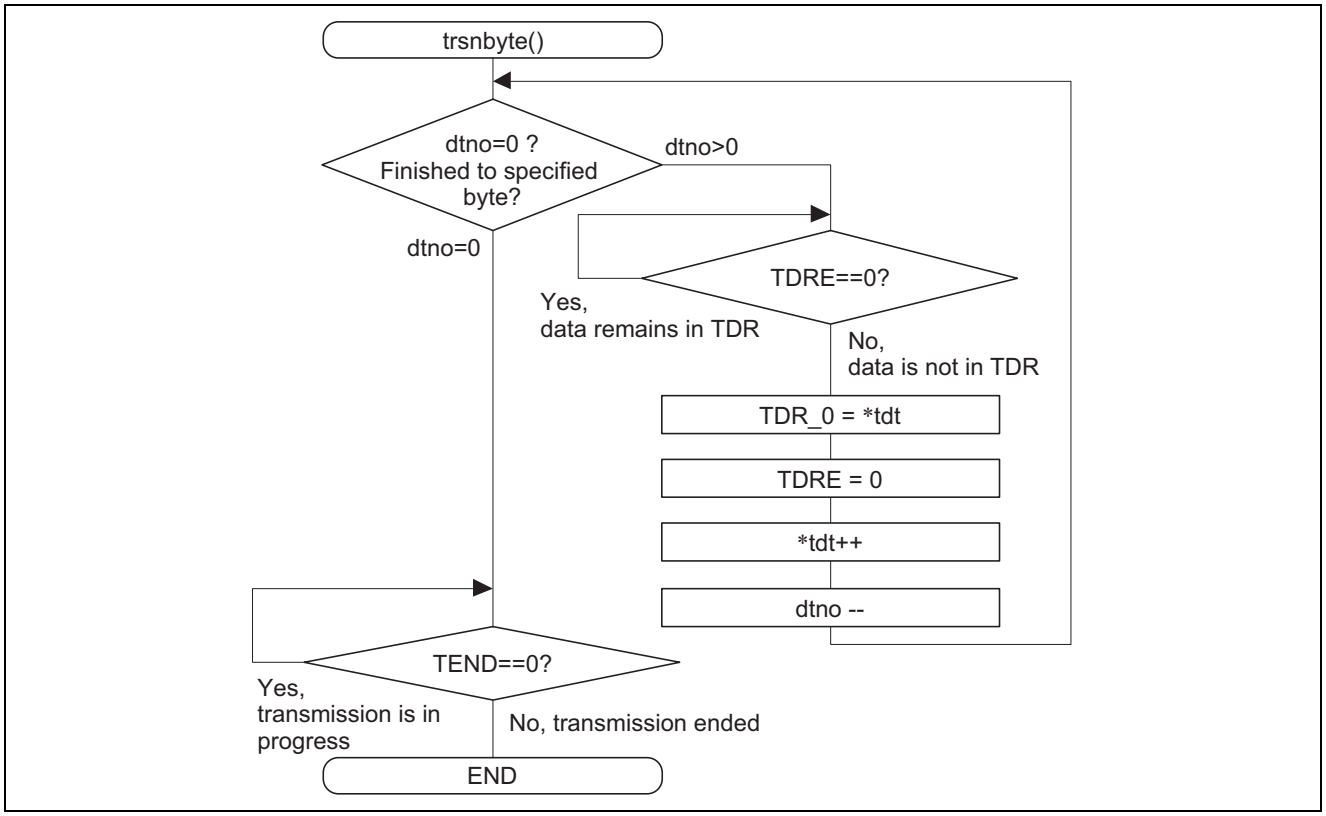
None

(f) Internal Registers Used

Table 23 Registers Used by trsnbyte() Function

Register	Bit Name	Description	Address	Set Value
TDR_0		Transmit data register 0 • 8-bit register that stores transmit data	H'FFFF7B	—
SSR_0		Serial status register 0	H'FFFF7C	—
	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


trsnbyte()
dtno=0 ?
Finished to specified byte?
dtno>0
dtno=0
TDRE==0?
Yes,
data remains in TDR
No,
data is not in TDR
TDR_0 = *tdt
TDRE = 0
*tdt++
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 *)0xFFFE30       /* Port 1 Data Direction Register   */
#define P1DR                 *(volatile unsigned char *)0xFFFF00       /* Port 1 Data Register             */
#define P1DR_BIT             (*(volatile struct BIT *)0xFFFF00)        /* Port 1 Data Register             */
#define P11DR                P1DR_BIT.b1                               /* Port 11                          */
#define P10DR                P1DR_BIT.b0                               /* Port 10                          */
#define P7DDR                *(volatile unsigned char *)0xFFFE36       /* Port 7 data direction register   */
#define P7DR                 *(volatile unsigned char *)0xFFFF06       /* Port 7 data register             */
#define PORT7                *(volatile unsigned char *)0xFFFFB6       /* Port 7 register                  */
#define PORT7_BIT            (*(volatile struct BIT *)0xFFFFB6)        /* Port 7 register                  */
#define P70                  PORT7_BIT.b0                              /* Port 70                          */
```

```
/************************************************************/
/* Function define                                          */
/************************************************************/
extern void FZMAIN ( void );
void main ( void );
void copyfzram ( void );
extern void com_init ( void );
extern void trsnbyte ( unsigned char *tdt, unsigned char dtno );
extern unsigned char rcv1byte ( void );
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                                                     /* 00 Reset                                     */
};

#pragma entry main(sp=0x00FFEFC0)
#pragma section                                          /* P                                            */
/************************************************************/
/* 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();                                              /* Comunication Initialize                      */

swcnt = 0;                                               /* User Application Program Sample              */
do{
  if(swcnt == 1){
    trsnbyte(&SAMPLEDT1[0], 10);
  }
  else if(swcnt == 2){
    trsnbyte(&SAMPLEDT2[0], 10);
  }
  else if(swcnt == 3){
    trsnbyte(&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 = rcv1byte();
```

```
}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 SECTOIN 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("FZEND");
  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 *)0xFFFEDB       /* RAM Emulation Register                */
#define FLPWCR           *(volatile unsigned char *)0xFFFFAC       /* Flash Memory Power Control Register   */
#define SCRX             *(volatile unsigned char *)0xFFFDB4       /* Serial Control Register X             */
#define SCRX_BIT         (*(volatile struct BIT *)0xFFFDB4)       /* Serial Control Register X             */
#define FLSHE            SCRX_BIT.b3                              /* Flash Memory Control Register Enable  */
#define TCSRW_0          *(volatile unsigned short *)0xFFFF74      /* Timer Control/Status Register W       */
#define TCNTW_0          *(volatile unsigned short *)0xFFFF74      /* Timer Counter W                       */
#define RSTCSR           *(volatile unsigned short *)0xFFFF76      /* Timer Control/Status Register W       */
#define P1DDR            *(volatile unsigned char *)0xFFFE30       /* Port 1 Data Direction Register        */
#define P1DR             *(volatile unsigned char *)0xFFFF00       /* Port 1 Data Register                  */
#define P1DR_BIT         (*(volatile struct BIT *)0xFFFF00)       /* Port 1 Data Register                  */
#define P11DR            P1DR_BIT.b1                              /* Port 11                               */
#define P10DR            P1DR_BIT.b0                              /* Port 10                               */
#define IER              *(volatile unsigned char *)0xFFFE14       /* IRQ Enable Register                   */
#define ADCR             *(volatile unsigned char *)0xFFFF99       /* A/D Control Register                  */
#define PFDDR            *(volatile unsigned char *)0xFFFE3E       /* Port F Data Direction Register        */
#define PFDR             *(volatile unsigned char *)0xFFFF0E       /* Port F Data Register                  */
```

```
/************************************************************/
/* 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 ferasevf ( unsigned short *evf_st, unsigned short *evf_ed );
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 *owbu ff, unsigned short *buff , un signed s hort *wvf_buf, unsigned s hort
*wvf_adr );
extern unsigned char rcv1byte ( void );
extern void rcvnbyte ( unsigned char *ram, unsigned char dtno );
extern void trs1byte ( unsigned char tdt );

/************************************************************/
/* ROM define                                               */
/************************************************************/

/****** 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] ={                            /* Erase Block Address                  */
0x000000,                                                /* EB0 4KBYTE                           */
0x001000,                                                /* EB1 4KBYTE                           */
0x002000,                                                /* EB2 4KBYTE                           */
0x003000,                                                /* EB3 4KBYTE                           */
0x004000,                                                /* EB4 4KBYTE                           */
0x005000,                                                /* EB5 4KBYTE                           */
0x006000,                                                /* EB6 4KBYTE                           */
0x007000,                                                /* EB7 4KBYTE                           */
0x008000,                                                /* EB8 32KBYTE                          */
0x010000,                                                /* EB9 64KBYTE                          */
0x020000,                                                /* EB10 64KBYTE                         */
0x030000,                                                /* EB11 64KBYTE                         */
0x040000                                                 /* End Block Address                    */
};

#define MAXBLK1      12
#define OK           0
#define NG           1
#define WNG          2
#define OW_COUNT 6                                       /* Over Write Count                     */
/************************************************************/
/* 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();                                     /* Recive 1byte Data -> RAM Area            */
  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();                                     /* Recive 1byte Data -> RAM Area            */
  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){
    trs1byte(0x11);                                     /* SEND OF Request                          */

    if(rcv.lw.restsize <= 128){                         /* Receive WriteData from HOST              */
        memset(W_BUF,0xFF,128);                         /* INITIALIZE RECEIVE BUFFER (0xFF)         */
        rcvnbyte(W_BUF,(unsigned char)rcv.lw.restsize); /* "restsize" byte Receive                  */
```

```
      rcv.lw.restsize = 0;
      }
      else{
          rcvnbyte(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;                                    /* INITIZLIZED WDT COUNT                    */
    TCSRW_0 = 0xA578;                                    /* 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,             /* Erase Verify                                 */
                (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                    */

  TCSRW_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( j = 0; &evf_st[j] < evf_ed; j++){
    ead[j*2] = 0xFF;                       /* Perform dummy write                                   */
    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 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,
                       (unsigned short *)BUFF,
                       (unsigned short *)wt_buf,
                       (unsigned short *)wt_adr);
```

```
        if(j < OW_COUNT){                                   /* Count Check(additive Write Count)          */
            fwrite(OWBUFF,wt_adr,TIME10);                   /* Input P Palse(10 usec)                     */
        }
        else{
            TM = TIME200;                                   /* Input P Palse(200 usec)                    */
        }

        if(rtn != NG){
            break;                                          /* NG Write Over Error                        */
        }
    }
  }

  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 *owbu ff, unsigned short *buff , un signed 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++){
    wad[j*2] = 0xFF;                                        /* Dummy Write                                */
    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

```
/************************************************************/
/*                                                          */
/* H8S/2268F                                                */
/* SCI Program                                              */
/*                                                          */
/* External Clock : 10MHz                                   */
/* Internal Clock : 10MHz                                   */
/* Sub Clock : 32.768kHz                                    */
/*                                                          */
/************************************************************/
#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 3                     */
#define SCR_0_BIT     (*(volatile struct BIT *)0xFFFF7A)     /* Serial Control Register 3                     */
#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 TDRE_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 Erorr                                 */
#define FER_0         SSR_0_BIT.b4                           /* Framing Erorr                                 */
#define PER_0         SSR_0_BIT.b3                           /* Parity Erorr                                  */
#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 *)0xFFFDF8    /* Serial Expansion Mode Register                */
#define MSTPCRB       *(volatile unsigned char *)0xFFFDE9    /* Module Stop Control Registers C               */

/************************************************************/
/* Communication Initialize                                 */
/************************************************************/
void com_init ( void )
{
  unsigned short i;

  MSTPCRB &= 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 = 0xF2;                                               /* Don't use Smart Card                          */
  SEMR_0 = 0x00;                                             /* 1bit-interval base clock is                   */
                                                             /* 16times 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--){
    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

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Mar.09.05	—	First edition issued
2.00	Aug.29.06	3, 25	New addition and content correction

Keep safety first in your circuit designs!

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

Notes regarding these materials

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