

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.

H8/300L SLP Series

User-Mode Flash-Memory Programming: 8-Bit Parallel Communications (H8/38024F)

Introduction

Program data in the flash memory of the master H8/38024F is programmed into the flash memory of the slave H8/38024F. The program data is transferred by 8-bit parallel communications.

Target Device

H8/38024F

Contents

1. Specifications	2
2. Detailed Description of Specifications	3
3. Operation Overview	8
4. Sequence Diagrams	12
5. Normal Program on Slave Side	18
6. Program/Erase Control Program on Slave Side	25
7. Program on Master Side	52
8. Program Listing	66

Note: The on-board programming algorithm as described in this application note is not guaranteed. The programming time and other data are not guaranteed either. Use them as reference values when designing the system.

1. Specifications

1. Flash-memory programming is performed in user mode.
2. Program data in flash memory of the master device is programmed into flash memory of the slave device.
3. Eight-bit parallel communications are used for transfer of program data.
4. When switch 0 (SW0) on the master side is turned on, the master transmits a flash-memory programming start command to the slave to start programming of flash memory on the slave side.
5. During programming into flash memory, LED1 is unlit and LED2 is lit. After programming into flash memory is finished, LED1 is lit and LED2 is unlit. This is true for both the master and slave sides.
6. Switch 0 (SW0) is connected to the $\overline{\text{IRQ0}}$ pin of the master device.
7. For both the master and slave sides, LED1 is connected to the P92 output pin, and LED2 to the P93 output pin.
8. P92 is a large-current port.
9. Figure 1.1 shows an example of configuration for on-board programming.

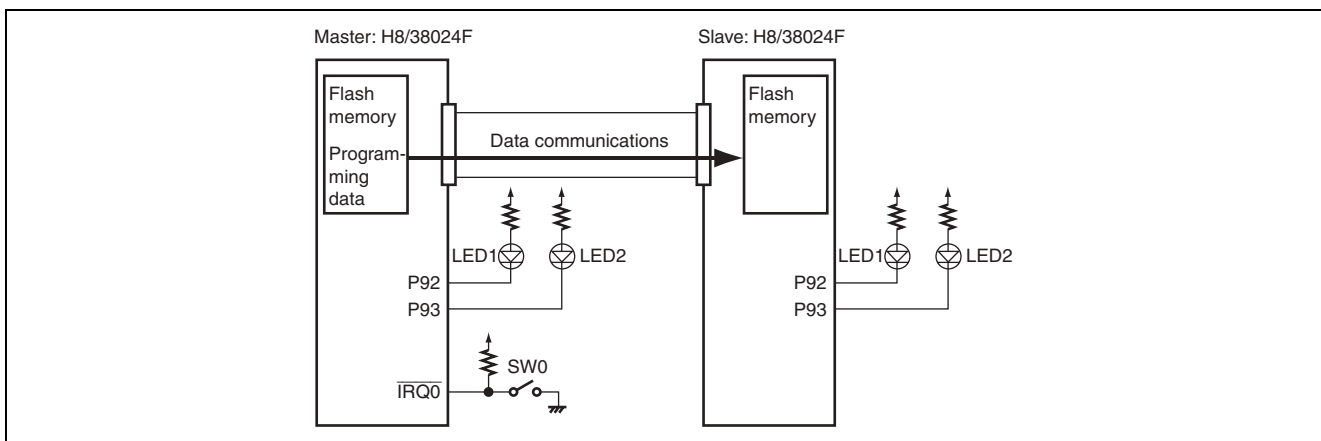


Figure 1.1 Example of Configuration for On-Board Programming

2. Detailed Description of Specifications

2.1 Operating Conditions for On-Board Programming

- Device: HD64F38024 (H8/38024F)
- CPU operation: User mode
- Operating voltage: 3.3 V
- Operating frequency: 5 MHz

2.2 On-Board Programming Mode

- User mode
 This mode assumes that the program/erase control program and RAM transfer program are already programmed in boot mode or programmer mode.

2.3 Programming Method

- Program data is received from the transfer source and programmed into flash memory.
- Eight-bit parallel communications are used for communication with the transfer source. The master device is the transfer source, and the slave device is the receiving side.

2.4 Flowchart of Programming Procedure

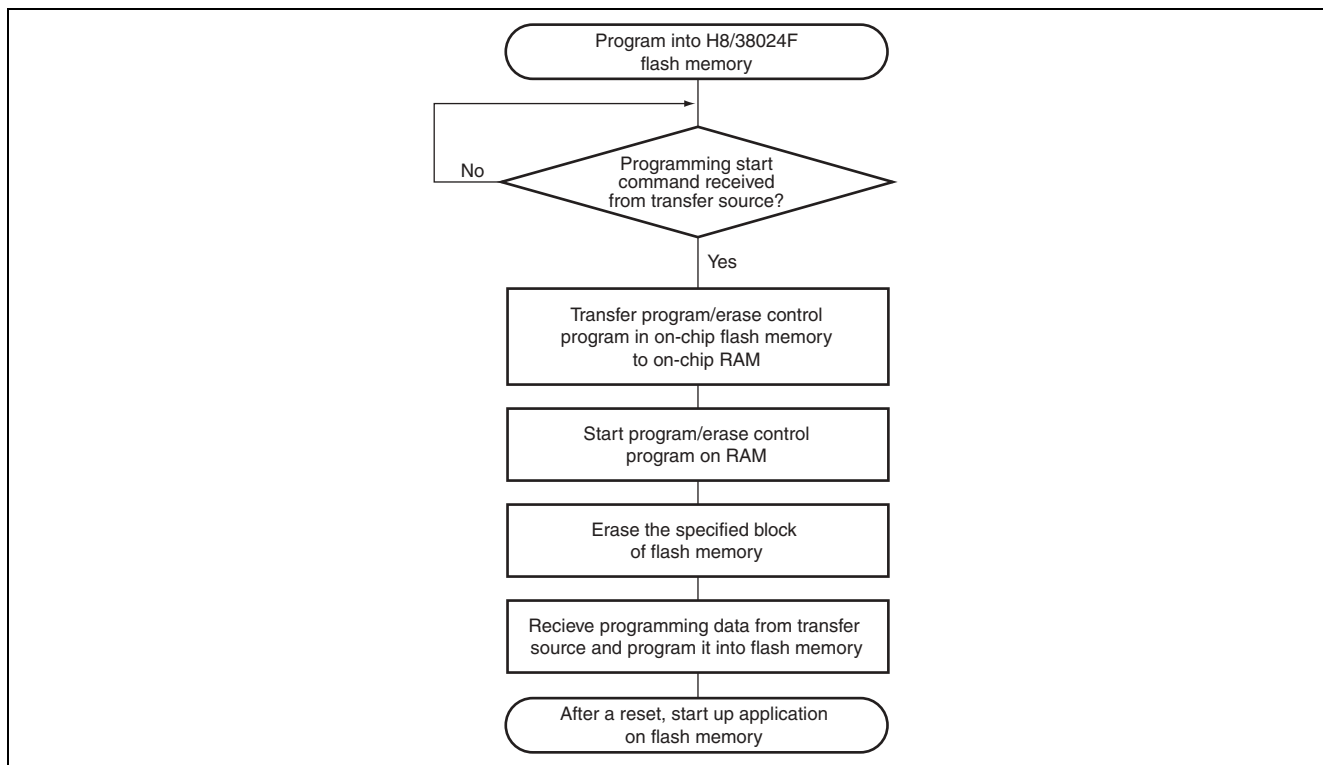


Figure 2.1 User-Mode Programming Procedure

2.5 Connection between Master and Slave

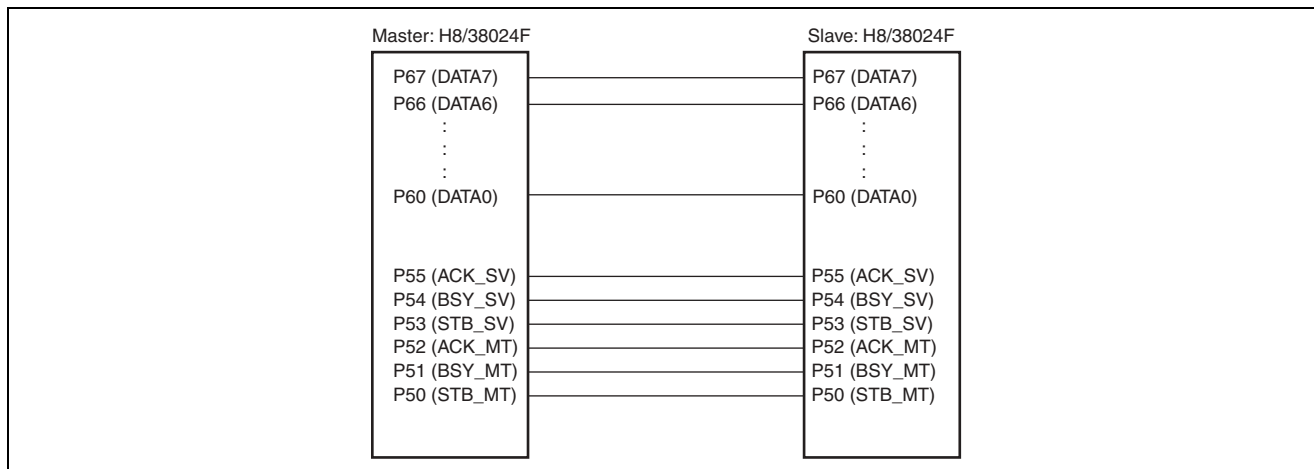


Figure 2.2 Connection between Master and Slave

2.6 Communication Specifications

Table 2.1 Communication Specifications

Communication method	Eight-bit parallel communications
Data bit	8
Number of control lines	6

2.7 Communication Commands

Table 2.2 Communication Commands

Communication command	Description
0x00	Normal communication (command name: OK command)
0x01	Abnormal communication (command name: NG command)
0x11	Transmit start request
0x55	Program start command
0x77	Erase command
0x88	Program command

2.8 List of Signal Lines

The following table lists the names of the signal lines used for 8-bit parallel communications and their pin assignments.

Table 2.3 Signal Lines

Signal name	Corresponding port	Signal direction	Description
DATA7 to DATA0	P63 to P60	From master to slave, or vice versa	Signal line for data
ACK_SV	P55	From slave to master	Receive enable on the slave side ACK_SV = 0: The slave can receive data. ACK_SV = 1: The slave cannot receive data.
BSY_SV	P54	From slave to master	Processing state on the slave side BSY_SV = 0: Internal processing on the slave side ended. BSY_SV = 1: Internal processing on the slave side is in progress.
STB_SV	P53	From slave to master	Signal state on the slave side STB_SV = 0: The slave is not transmitting data. STB_SV = 1: The slave is transmitting data.
ACK_MT	P52	From master to slave	Receive enable on the master side ACK_MT = 0: The master can receive data. ACK_MT = 1: The master cannot receive data.
BSY_MT	P51	From master to slave	Processing state on the master side BSY_MT = 0: Internal processing on the master side ended. BSY_MT = 1: Internal processing on the master side is in progress.
STB_MT	P50	From master to slave	Signal state on the master side STB_MT = 0: The master is not transmitting data. STB_MT = 1: The master is transmitting data.

2.9 Memory Allocation

Table 2.4 lists the erase blocks in the H8/38024F flash memory.

Table 2.4 Erase Blocks in Flash Memory

Block (size)	Address
EB0 (1 Kbyte)	0x0000 to 0x03FF
EB1 (1 Kbyte)	0x0400 to 0x07FF
EB2 (1 Kbyte)	0x0800 to 0x0BFF
EB3 (1 Kbyte)	0x0C00 to 0x0FFF
EB4 (28 Kbytes)	0x1000 to 0x7FFF

Figure 2.3 shows the memory maps for the normal operation and program operation of the H8/38024F.

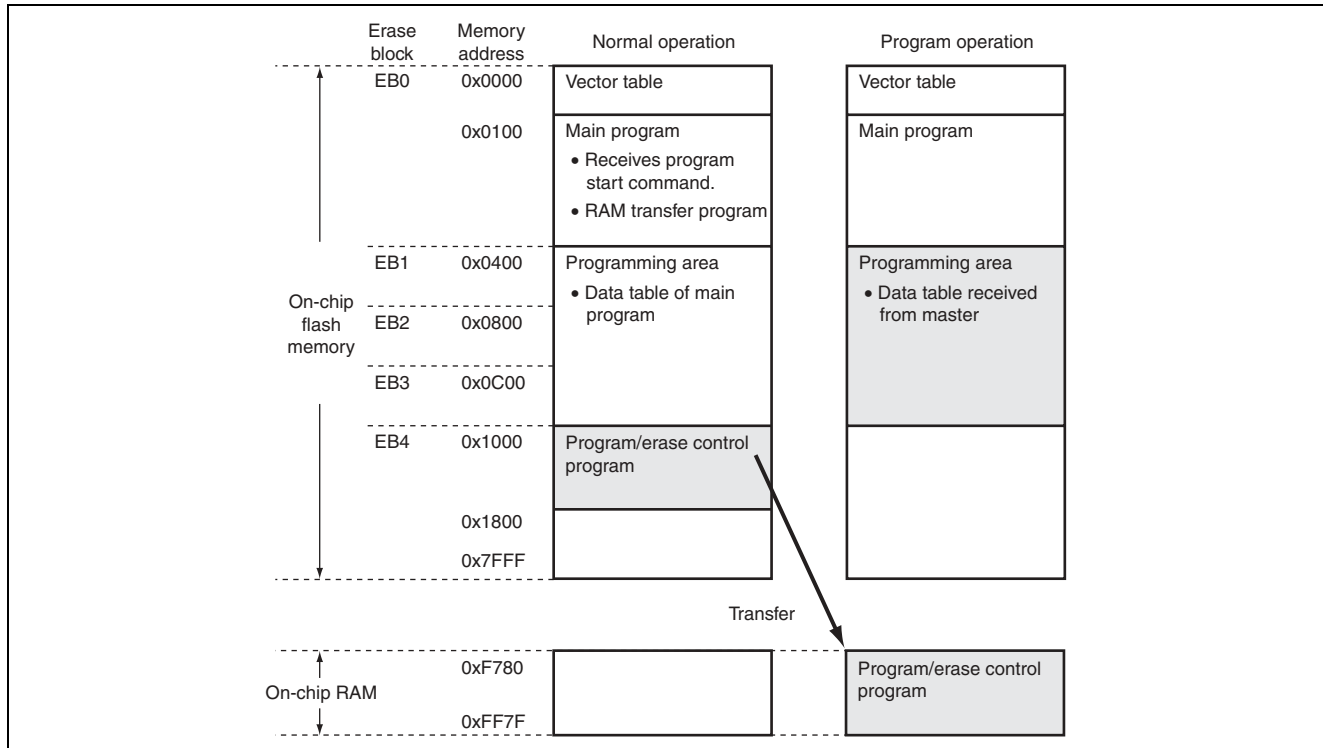


Figure 2.3 Memory Maps

2.10 Compile and Link Options

1. Compile options

Table 2.5 lists the compile options used in the sample programs in this application.

Table 2.5 Compile Options

Item	Command line format	Option	Description
CPU/operating mode	-cpu =	300	Creates H8/300 object code.
Object format	-code =	machinecode	Outputs machine code.
Details of optimization	-speed =	register	Uses PUSH and POP instructions, instead of a runtime routine, to expand the code that saves and recovers registers at the entry and exit of functions.

2. Batch file examples

The following shows the batch files used for the sample programs.

A. Batch file for the slave device

```
CH38 -cpu = 300 -code = machinecode -speed = register main.c (main.c: Normal program on the slave side.)
CH38 -cpu = 300 -code = machinecode -speed = register slvf.c (slvf.c: Program/erase control program on the
slave side.)

ASM38 init.src -cpu = 300 (nit.src: Stack-pointer setting program.)
LNK -sub = link.sub
CNVS slvf.abs
```

link.sub contains:

```
output slvf.abs
print slvf.map
input ..\init.obj main.obj slvf.obj lcdt.obj
lib C:\Hew\Tools\Hitachi\H8\3_0a_0\lib\c38reg.lib
start CV1(00000),P(00100),DLCDDT1(00400),DLCDDT2(00800),DLCDDT3(00FFA)
start FZTAT,PFZTAT,DFZTAT,FZEND(01000),RAM,PRAM,DRAM,B(0F780)
exit
```

B. Batch file for the master device

```
CH38 -cpu = 300 -code = machinecode mst.c (main.c: mst.c: Program on the master side.)
ASM38 init.src -cpu = 300 (init.src: Stack-pointer setting program)
LNK -sub = link.sub
CNVS mst.abs
```

link.sub contains:

```
output mst.abs
print mst.map
input ..\init.obj mst.obj lcdt.obj
lib C:\Hew\Tools\Hitachi\H8\3_0a_0\lib\c38reg.lib
start CV1 (00000), CV2 (00008), P (01000)
start D (00100), DLCDDT1 (00400), DLCDDT2 (00800), DLCDDT3 (00FFA)
start B (0FB80)
exit
```

3. Compiler versions used

Table 2.6 lists the versions of the development environment used in this application.

Table 2.6 Development Environment Versions

Software name	Version used
C compiler	H8S, H8/300 SERIES C/C++ Compiler Ver3.0A
Cross assembler	H8S, H8/300 SERIES CROSS ASSEMBLER Ver3.0B
Linkage editor	H SERIES LINKAGE EDITOR Ver.6.0D
Object converter	H SERIES SYSROF STYPE OBJECT CONVERTER Ver.2.0A

3. Operation Overview

3.1 Normal Operation

1. The program/erase control program must be programmed in flash memory of the slave device in advance.
2. The normal application normally accesses the data table in flash memory. The data table is received from the master and programmed.
3. The program for receiving the program start command and RAM transfer program must be programmed in flash memory of the slave device in advance.
4. Eight-bit parallel communications are used for data communication between the master and slave.

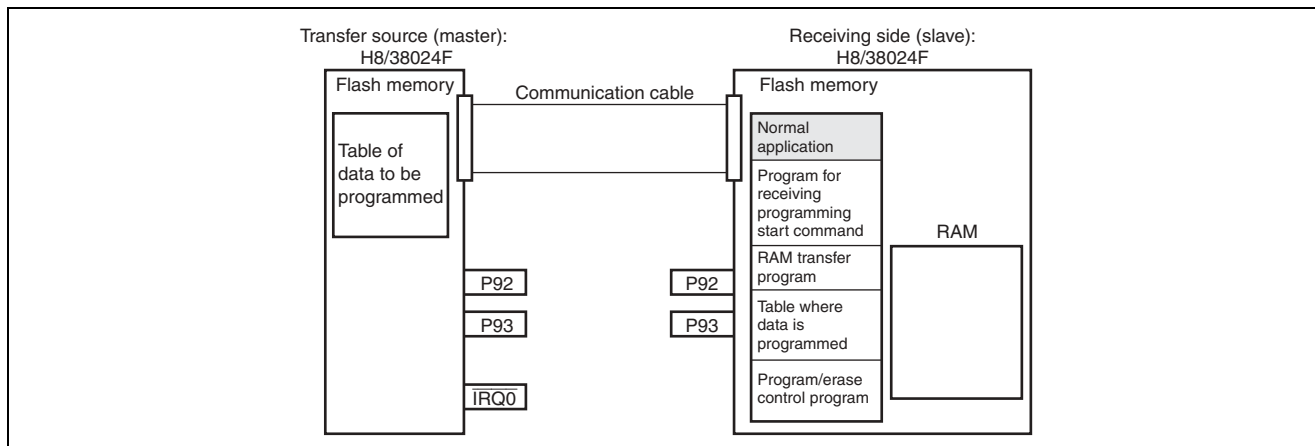


Figure 3.1 Normal Operation

3.2 Preparations for On-Board Programming

1. When a low trigger is input to the $\overline{\text{IRQ0}}$ pin of the master device, the master transmits the 0x55 program start command.
2. During this process, P92 outputs a high level and P93 outputs a low level on the master side.

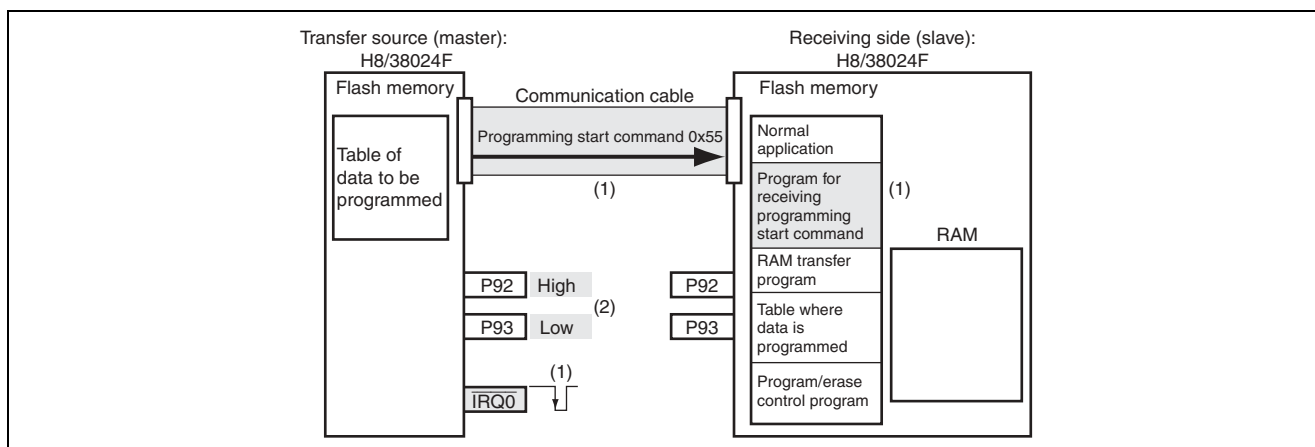


Figure 3.2 Preparations for On-Board Programming

3.3 Starting On-Board Programming

1. Upon receiving 0x55, the slave initiates the RAM transfer program to transfer the program/erase control program to the on-chip RAM.
2. During this process, P92 outputs a high level and P93 outputs a low level on the slave side.

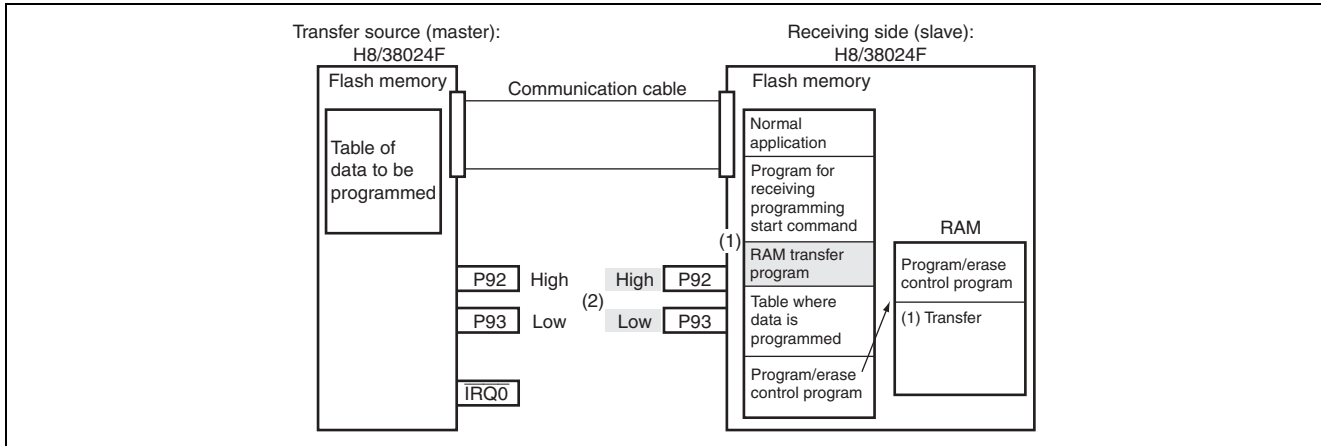


Figure 3.3 Starting On-Board Programming

3.4 Initiating Program/Erase Control Program

1. After transfer, the RAM transfer program branches to the program/erase control program on RAM.

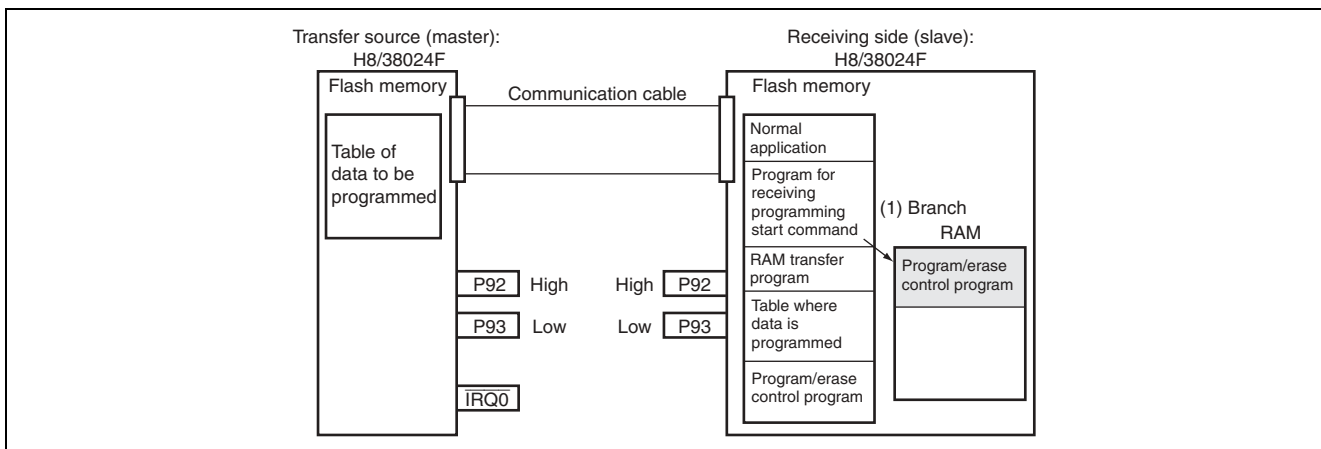


Figure 3.4 Initiating Program/Erase Control Program

3.5 Erasing Blocks in Flash Memory

1. The slave receives the 0x77 erase command from the master.
2. The program/erase control program erases the specified blocks in flash memory.

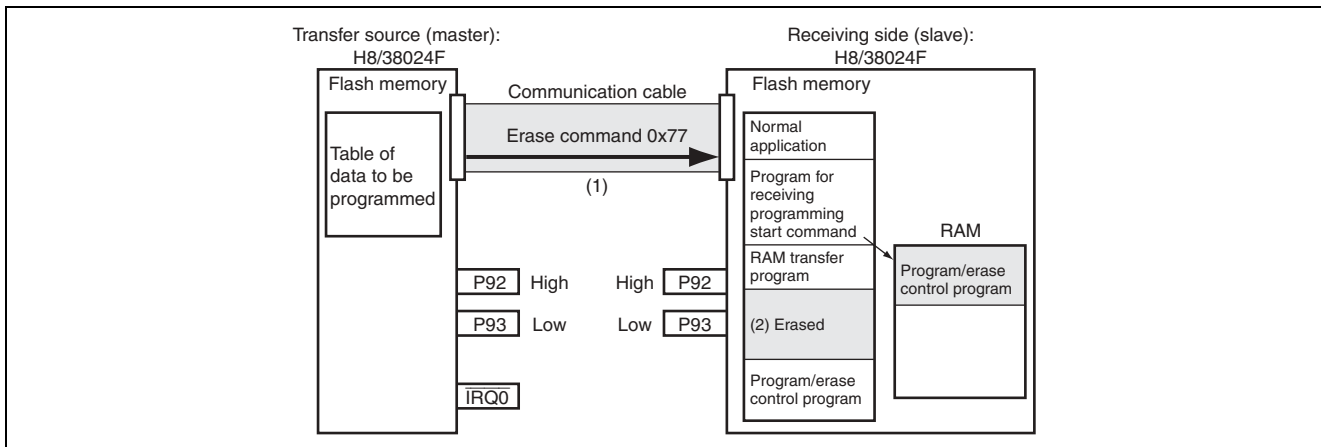


Figure 3.5 Erasing Blocks in Flash Memory

3.6 Programming into Flash Memory

1. The slave receives the 0x88 program command from the transfer source (master).
2. The program/erase control program receives a new data table from the transfer source and programs it into flash memory.
3. After programming is finished, P92 outputs a low level and P93 outputs a high level on both the master and slave sides.

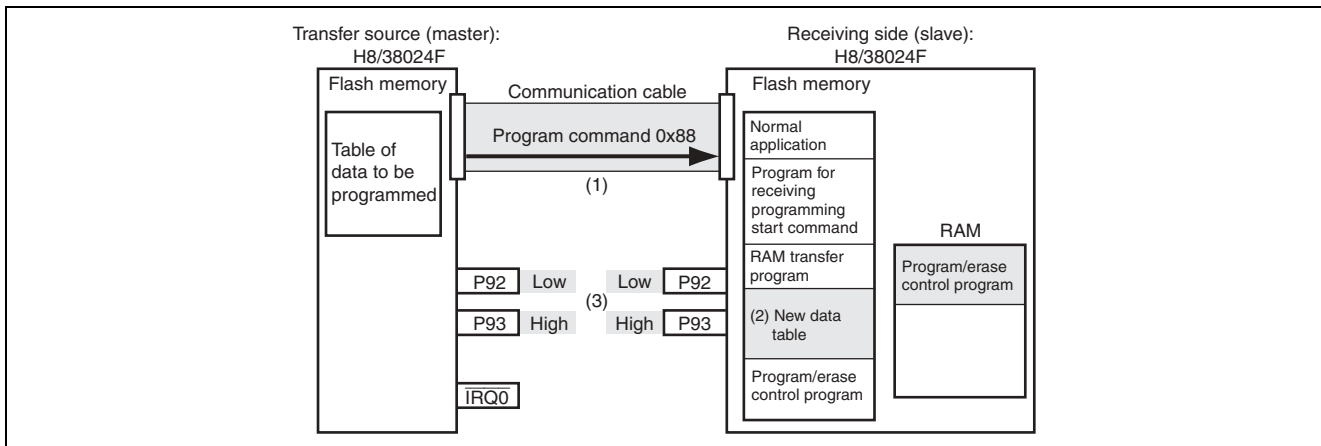


Figure 3.6 Programming into Flash Memory

3.7 Initiating the Normal Application Program

1. After a reset, the normal application that accesses the new data table is initiated.

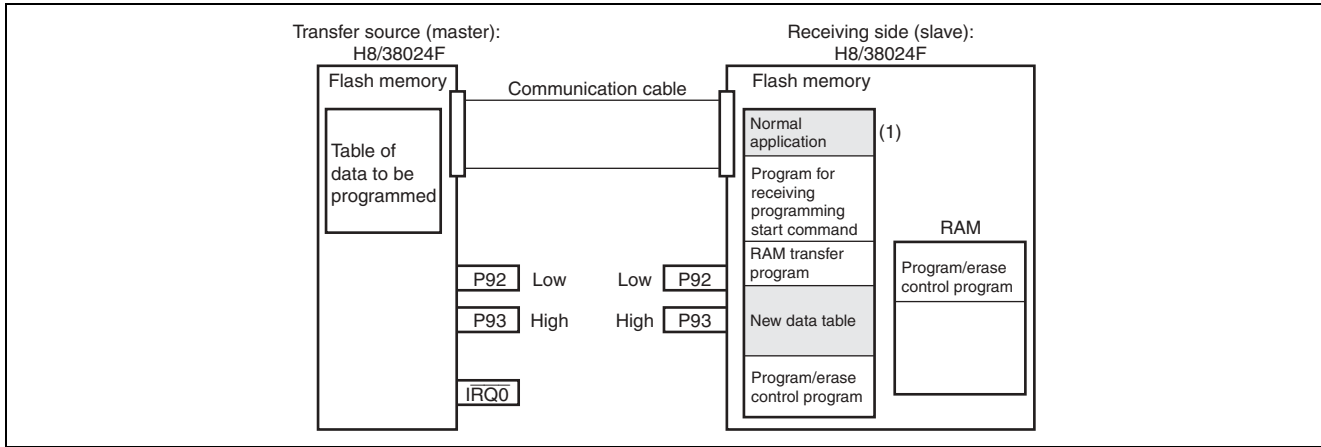


Figure 3.7 Initiating the Normal Application Program

4. Sequence Diagrams

1. Normal operation

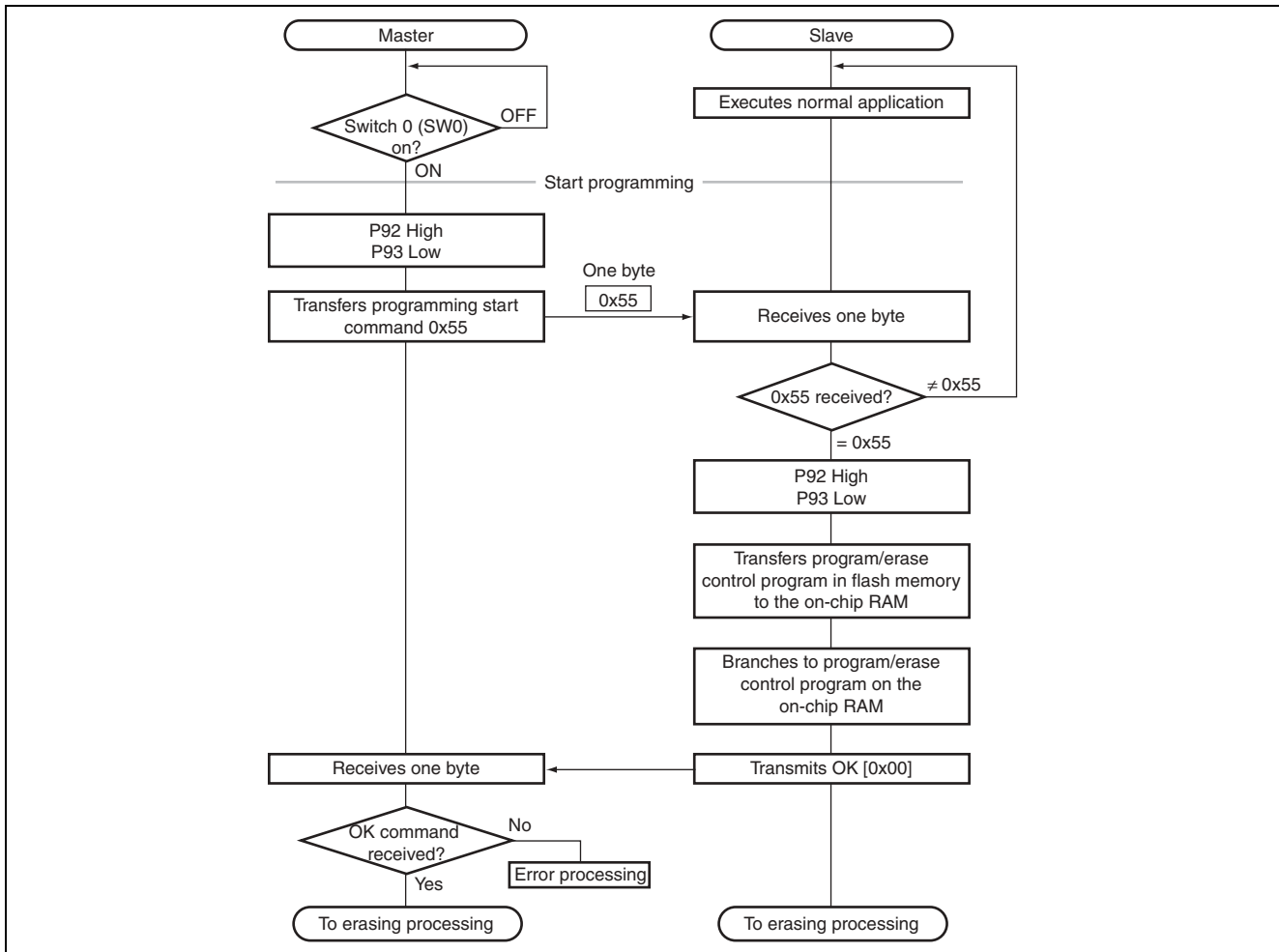


Figure 4.1 Normal Operation

2. Erase processing

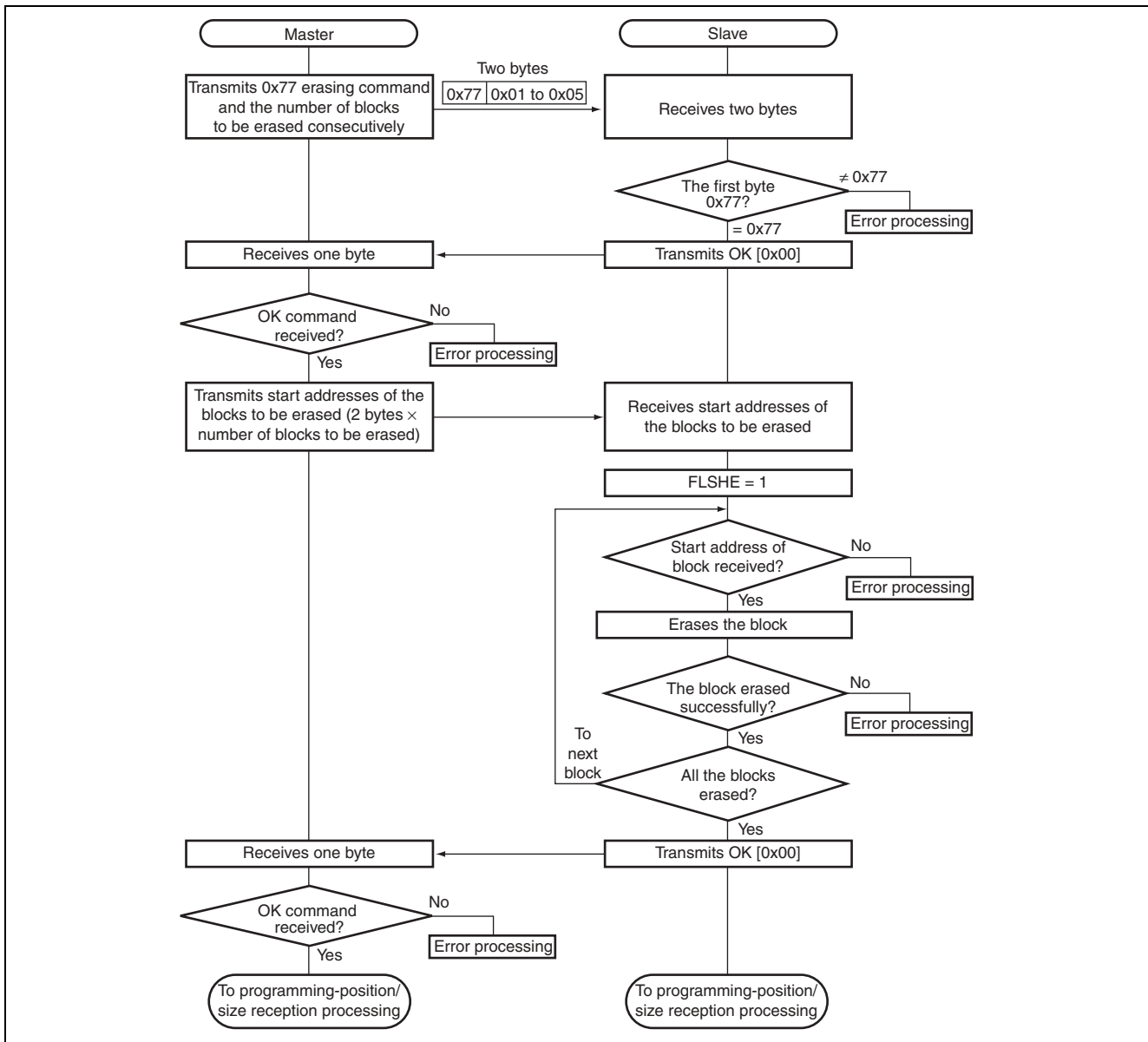


Figure 4.2 Erase Processing

3. Program-position/size receive processing

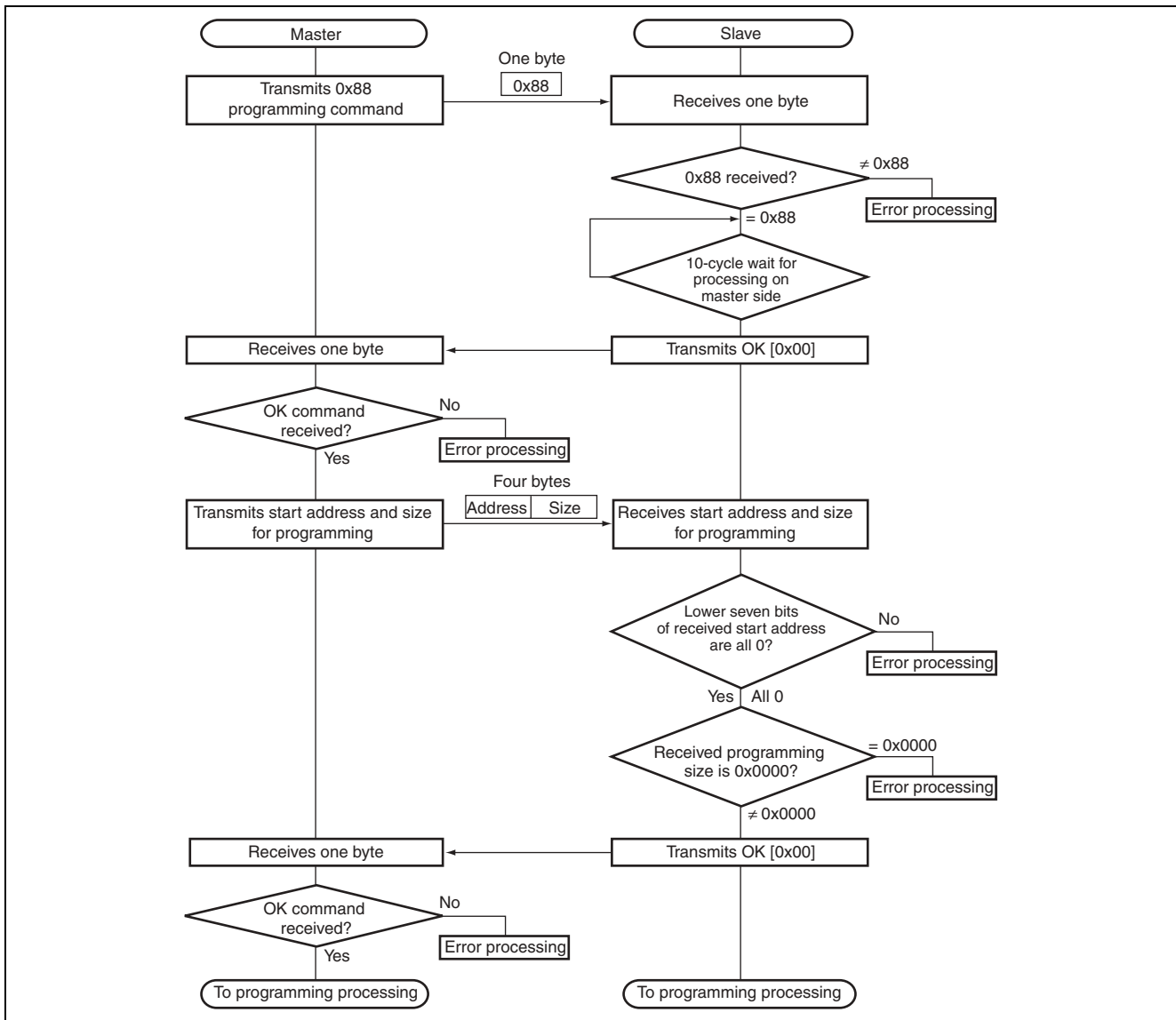


Figure 4.3 Program-Position/Size Receive Processing

4. Program processing

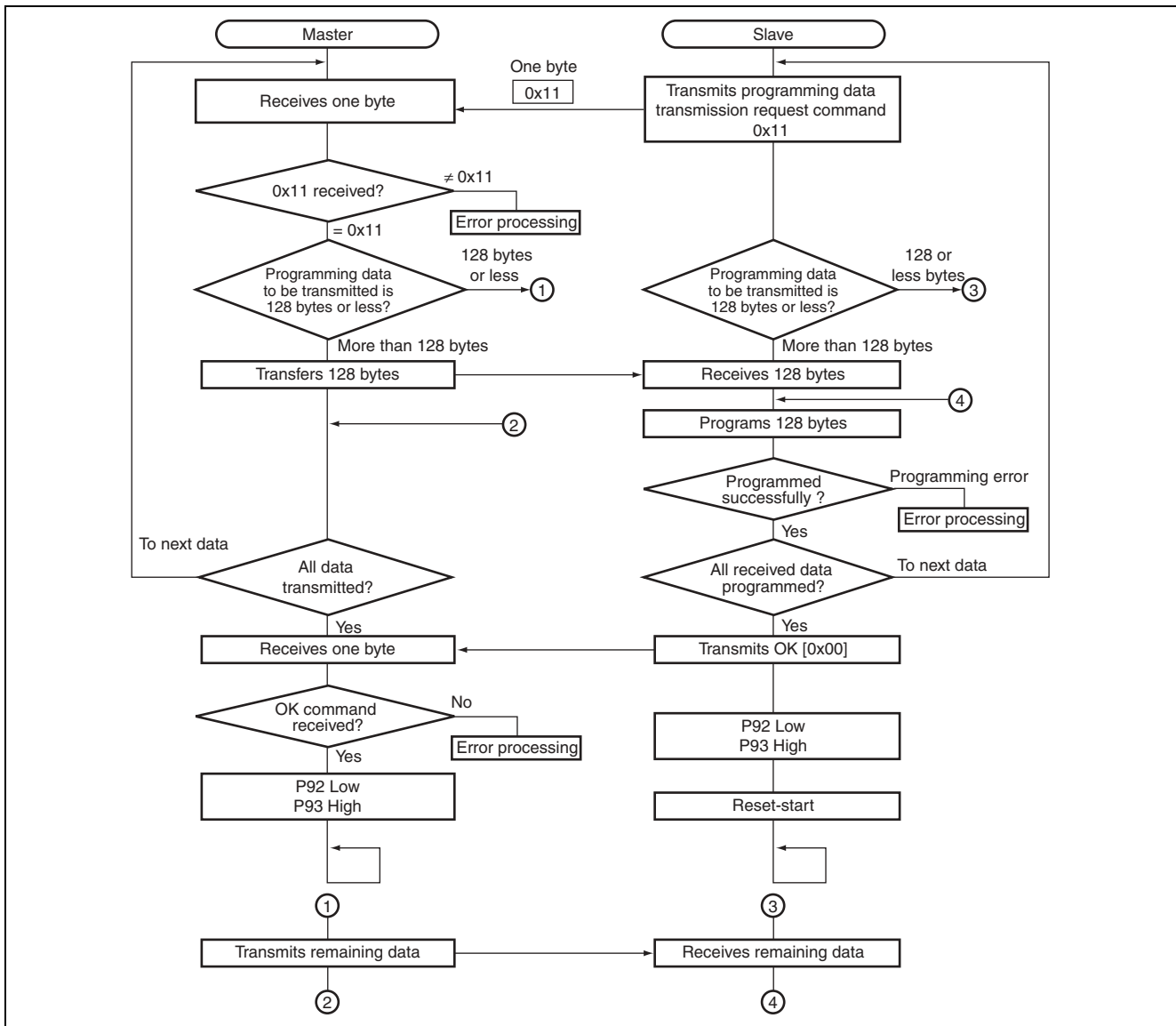


Figure 4.4 Program Processing

5. Error processing

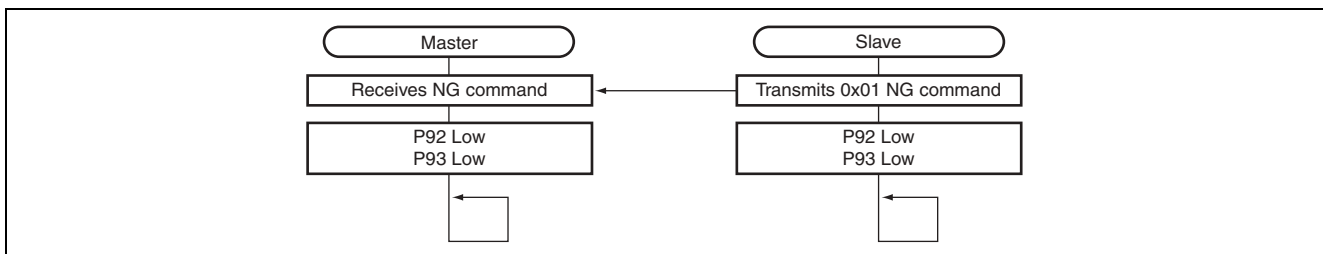


Figure 4.5 Error Processing

6. One-byte communication from master to slave

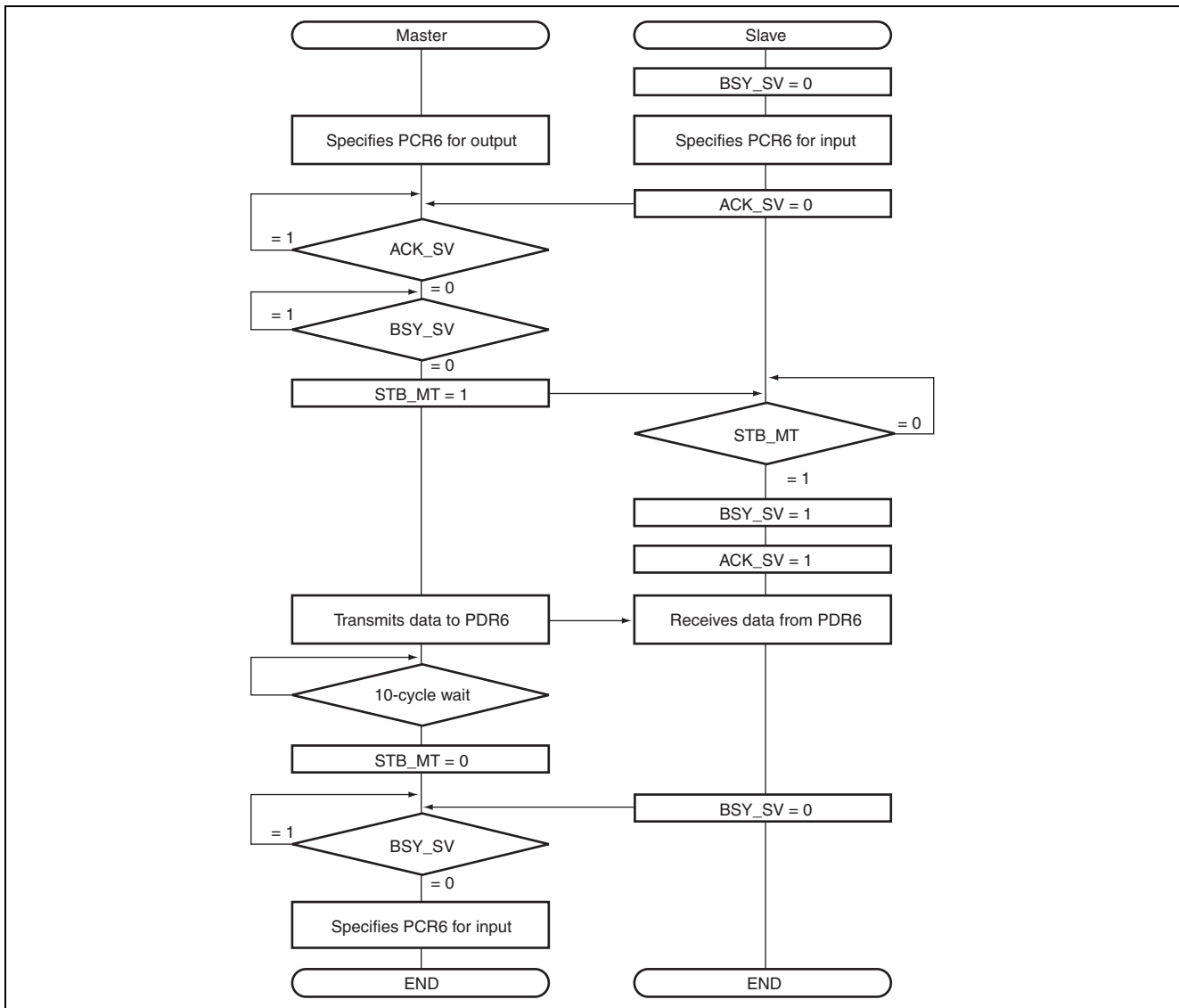


Figure 4.6 One-Byte Communication from Master to Slave

7. One-byte communication from slave to master

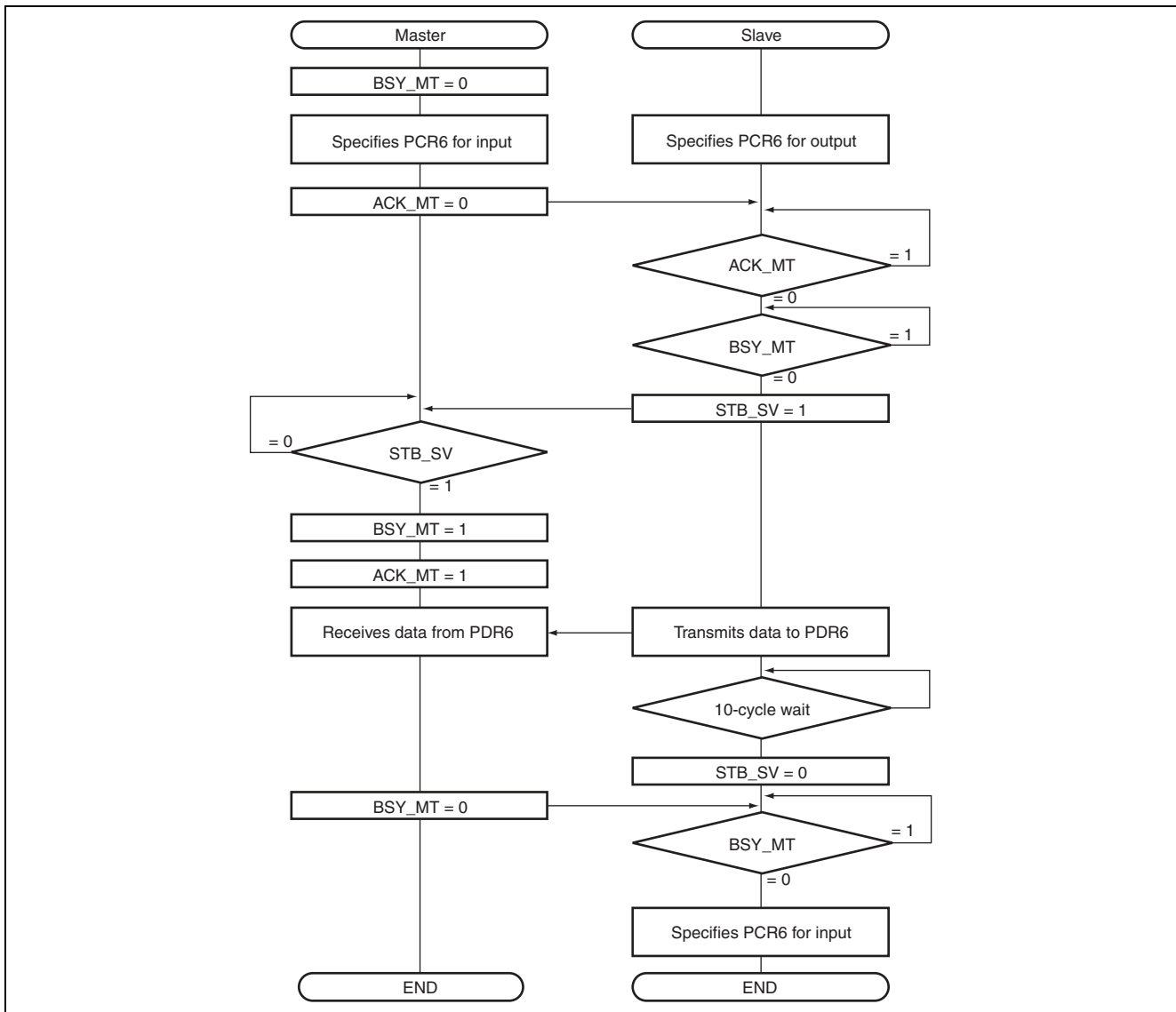


Figure 4.7 One-Byte Communication from Slave to Master

5. Normal Program on Slave Side

5.1 Hierarchical Structure

The normal program running on flash memory of the slave device executes the user application program (normal application), receives the program start command, and transfers the program/erase control program in flash memory to the on-chip RAM. Figure 5.1 shows the hierarchical structure of the routines used in the normal program on the slave side.

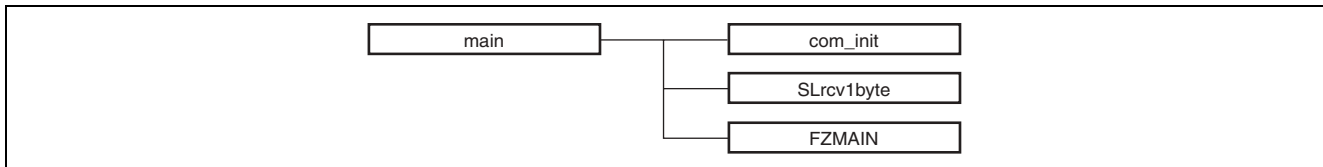


Figure 5.1 Normal Program on Slave Side

5.2 Functions

Table 5.1 Functions of Normal Program on Slave Side

Function	Overview
main	Executes the normal application, receives the program start command, and transfers the program/erase control program in flash memory to the on-chip RAM.
com_init	Initializes the communication settings.
SLrcv1byte	Receives one byte of data.
FZMAIN	Program/erase control program for flash memory

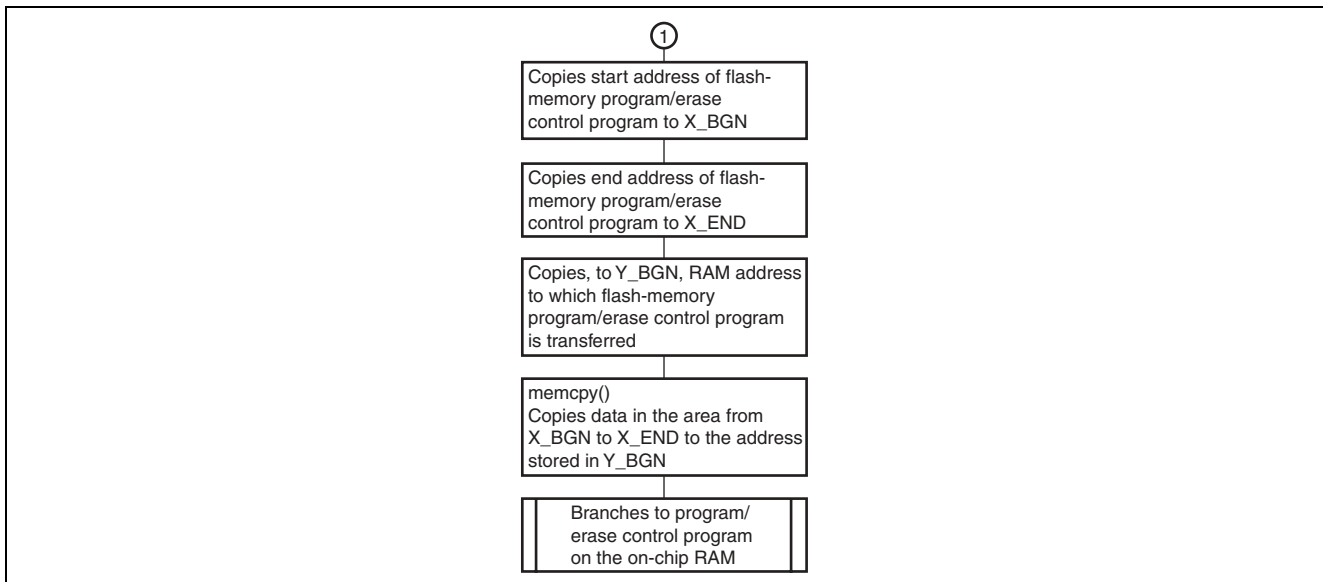
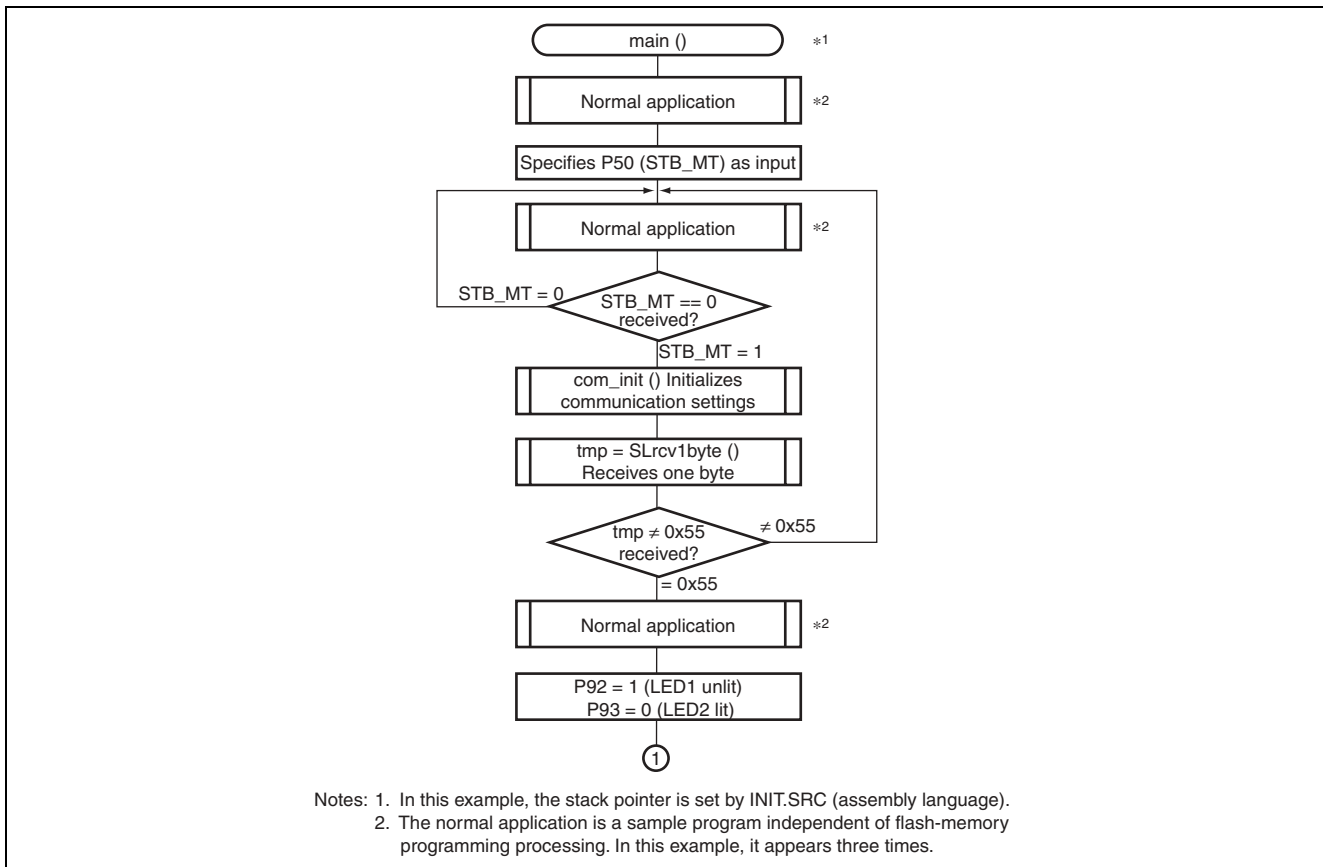
5.3 Description of Functions

1. main() function
 - A. Specifications
 - void main(void)
 - B. Operation
 - Executes the user application program (normal application).
 - Receives the program start command.
 - Transfers the program/erase control program to RAM.
 - Branches to the program/erase control program.
 - C. Arguments
 - Input: None
 - Output: None
 - D. Global variables
 - None
 - E. Subroutines used
 - com_init(): Initializes the communication settings.
 - SLrcv1byte(): Receives one byte of data.
 - FZMAIN(): Branches to the program/erase control program.
 - F. Internal registers used

Table 5.2 Registers Used by main() Function

Register	Function	Address	Setting	
PDR9	P93	Port data register 9 (port data register 93) When P93 = 0, the output level of the P93 pin is low. When P93 = 1, the output level of the P93 pin is high.	0xFFDC Bit 3	0
	P92	Port data register 9 (port data register 92) When P92 = 0, the output level of the P92 pin is low. When P92 = 1, the output level of the P92 pin is high.	0xFFDC Bit 2	1
LPCR	LCD port control register Used by the sample normal application.	0xFFC0	—	
LCR	LCD control register Used by the sample normal application.	0xFFC1	—	
LCR2	LCD control register 2 Used by the sample normal application.	0xFFC2	—	
LCDRAM	LCD RAM Used by the sample normal application.	0xF740 to 0xF74F	—	
PDR3	P37	Port data register 3 (port data register 37) Used by the sample normal application.	0xFFDC Bit 7	—
PDR5	P50	Port data register 5 (port data register 50) A port used as STB_MT: When STB_MT (P50) = 0, the master is not transmitting data. When STB_MT (P50) = 1, the master is transmitting data.	0xFFD8 Bit 0	—

G. Flowchart



2. com_init() function

A. Specifications

void com_init(void)

B. Operation

— Initializes the communication settings for 8-bit parallel communications.

C. Arguments

— Input: None

— Output: None

D. Global variables

None

E. Subroutines used

None

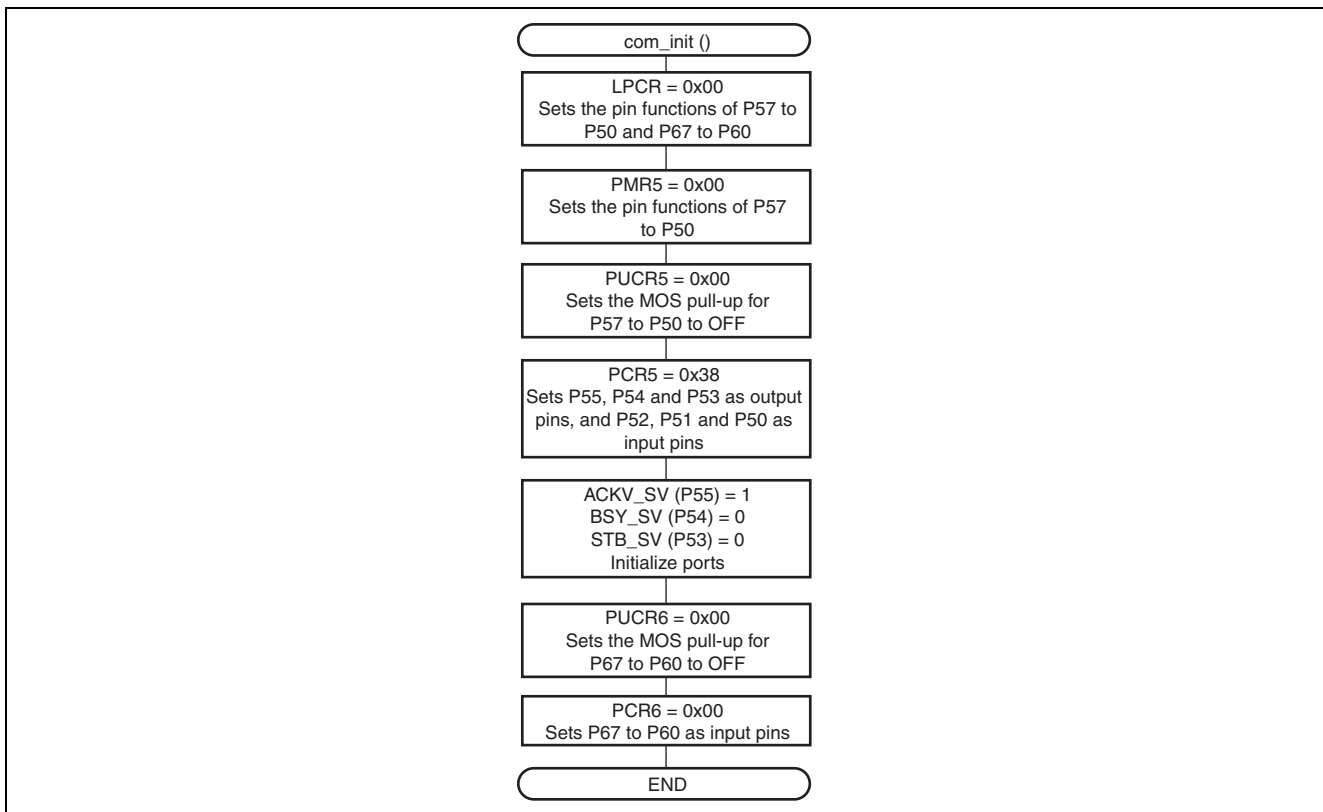
F. Internal registers used

Table 5.3 Registers Used by com_init() Function

Register	Function	Address	Setting
LPCR	SGS3 LCD port control register (segment driver selection)	0xFFC0	SGS3 = 0
	SGS2 When SGS3 = 0, SGS2 = 0, SGS1 = 0, and SGS0 = 0, the	Bit 3	SGS2 = 0
	SGS1 SEG32 to SEG1 pins function as ports.	Bit 2	SGS1 = 0
	SGS0	Bit 1	SGS0 = 0
		Bit 0	
PMR5	Port mode register 5 When PMR5 = 0x00, the $\overline{WKP7}$ to $\overline{WKP0}$ input pins function as P57 to P50 pins.	0xFFCC	0x00
PDR5	P55 Port data register 5 (port data register 55) A port used as ACK_SV: When ACK_SV (P55) = 0, the slave can receive data. When ACK_SV (P55) = 1, the slave cannot receive data.	0xFFD8 Bit 5	1
	P54 Port data register 5 (port data register 54) A port used as BSY_SV: When BSY_SV (P54) = 0, internal processing on the slave side ended. When BSY_SV (P54) = 1, internal processing on the slave side is in progress.	0xFFD8 Bit 4	0
	P53 Port data register 5 (port data register 53) A port used as STB_SV: When STB_SV (P53) = 0, the slave is not transmitting data. When STB_SV (P53) = 1, the slave is transmitting data.	0xFFD8 Bit 3	0
PUCR5	Port pull-up control register 5 When PUCR5 = 0x00, the MOS pull-up is turned off for P57 to P50.	0xFFE2	0x00
PUCR6	Port pull-up control register 6 When PUCR6 = 0x00, the MOS pull-up is turned off for P67 to P60.	0xFFE3	0x00

Register	Function	Address	Setting
PCR5	Port control register 5 When PCR5 = 0x38, P55, P54, and P53 are set as output pins and P52, P51, and P50 are set as input pins.	0xFFE8	0x38
PCR6	Port control register 6 When PCR6 = 0x00, P67 to P60 are set as input pins.	0xFFE9	0x00

G. Flowchart

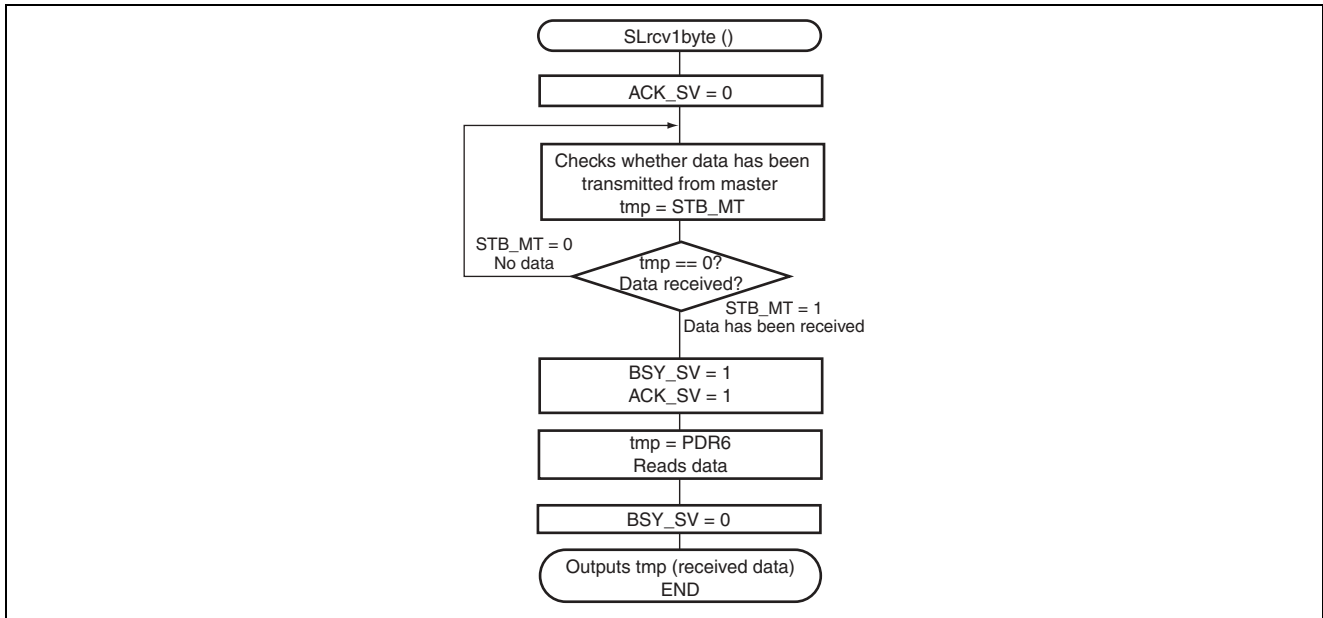


3. SLrcv1byte() function
 - A. Specifications
 - unsigned char SLrcv1byte(void)
 - B. Operation
 - Receives one byte of 8-bit parallel data.
 - C. Arguments
 - Input: None
 - Output: One byte of receive data
 - D. Global variables
 - None
 - E. Subroutines used
 - None
 - F. Internal registers used

Table 5.4 Registers Used by SLrcv1byte() Function

Register	Function	Address	Setting
PDR5	P55 Port data register 5 (port data register 55) A port used as ACK_SV: When ACK_SV (P55) = 0, the slave can receive data. When ACK_SV (P55) = 1, the slave cannot receive data.	0xFFD8 Bit 5	0
	P54 Port data register 5 (port data register 54) A port used as BSY_SV: When BSY_SV (P54) = 0, internal processing on the slave side ended. When BSY_SV (P54) = 1, internal processing on the slave side is in progress.	0xFFD8 Bit 4	1
	P50 Port data register 5 (port data register 50) A port used as STB_MT: When STB_MT (P50) = 0, the master is not transmitting data. When STB_MT (P50) = 1, the master is transmitting data.	0xFFD8 Bit 0	—
PDR6	Port data register 6 An 8-bit register that stores receive data.	0xFFD9	—

G. Flowchart



4. FZMAIN() function

Calls the main routine of the program/erase control program.

6. Program/Erase Control Program on Slave Side

6.1 Hierarchical Structure

The program/erase control program erases specified blocks, receives program data, and programs it into flash memory. Figure 6.1 shows the hierarchical structure of the routines used in the program/erase control program. The subroutines, excluding the FZMAIN() function, are classified into communication processing and flash-memory program/erase processing.

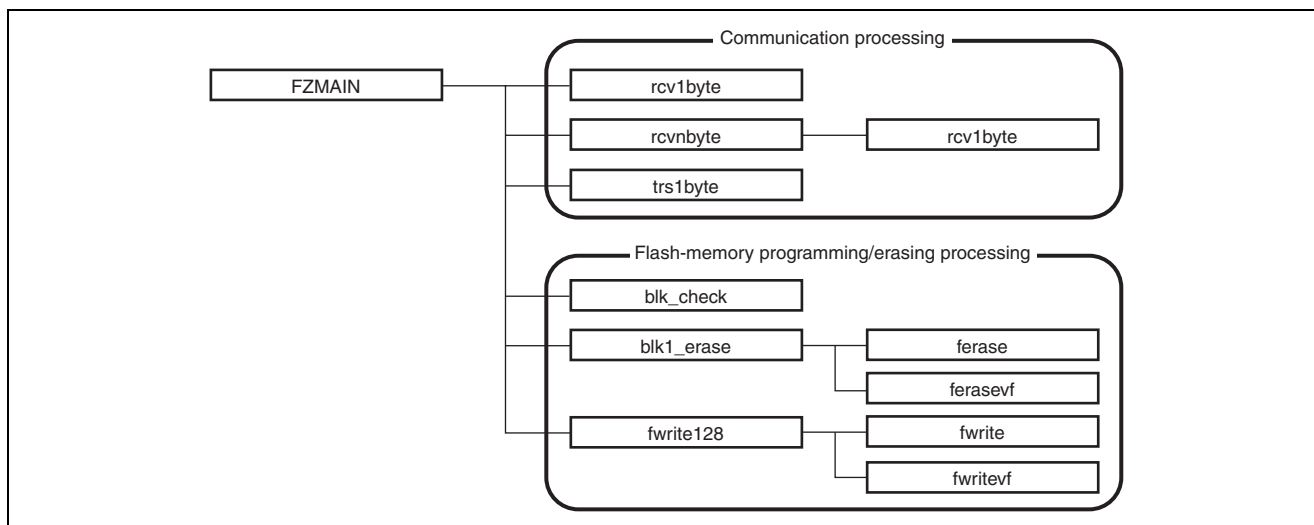


Figure 6.1 Program/Erase Control Program

6.2 Functions

Table 6.1 Functions of Program/Erase Control Program

Function	Overview
FZMAIN	Main routine of the program/erase control program
rcv1byte	Receives one byte of data.
rcvnbyte	Receives n bytes of data.
trs1byte	Transmits one byte of data.
blk_check	Determines the erase block number from the erase start address.
blk1_erase	Erases the specified block in flash memory.
ferase	Erases the specified block.
ferasevf	Verifies that the specified block has been erased.
fwrite128	Programs and verifies 128 bytes.
fwrite	Programs data to the specified address.
fwritevf	Verifies data for the specified address and creates reprogram data.

6.3 Constants

Table 6.2 Constants

Constant	Value	Description
OK	0x00	Return value for normal operation
NG	0x01	Return value for abnormal operation
WNG	0x02	Program error
MAXBLK1	0x0A	Total number of blocks in flash memory (5) × 2
WDT_ERASE	0x00	WDT count for flash-memory erasure
WDT_WRITE	0xFB	WDT count for flash-memory programming
OW_COUNT	0x06	Number of reprogrammings
WLOOP1	1*MHZ/KEISU+1 = 1 (0x01)	Number of WAIT statement executions (1 μs WAIT)
WLOOP2	2*MHZ/KEISU+1 = 2 (0x02)	Number of WAIT statement executions (2 μs WAIT)
WLOOP4	4*MHZ/KEISU+1 = 3 (0x03)	Number of WAIT statement executions (4 μs WAIT)
WLOOP5	5*MHZ/KEISU+1 = 4 (0x04)	Number of WAIT statement executions (5 μs WAIT)
WLOOP10	10*MHZ/KEISU+1 = 7 (0x07)	Number of WAIT statement executions (10 μs WAIT)
WLOOP20	20*MHZ/KEISU+1 = 13 (0x0D)	Number of WAIT statement executions (20 μs WAIT)
WLOOP50	50*MHZ/KEISU+1 = 32 (0x20)	Number of WAIT statement executions (50 μs WAIT)
WLOOP100	100*MHZ/KEISU+1 = 63 (0x3F)	Number of WAIT statement executions (100 μs WAIT)
TIME10	10*MHZ/KEISU = 6 (0x06)	Number of WAIT statement executions (10 μs WAIT)
TIME30	30*MHZ/KEISU = 18 (0x12)	Number of WAIT statement executions (30 μs WAIT)
TIME200	200*MHZ/KEISU = 125 (0x7D)	Number of WAIT statement executions (200 μs WAIT)
TIME10000	10000*MHZ/KEISU = 6250 (0x186A)	Number of WAIT statement executions (10 ms WAIT)

Notes: MHZ:5 Indicates that the operating frequency is 5 MHz.

KEISU:8 Indicates that the number of steps in a loop which is repeated by 'for' statement is 8.

6.4 Description of Communication Processing Functions

1. FZMAIN() function

A. Specifications

void FZMAIN(void)

B. Operation

- Erases blocks in flash memory.
- Receives data to be programmed into flash memory.
- Programs data into flash memory.
- Resets and starts the system after programming.

C. Arguments

- Input: None
- Output: None

D. Global variables

None

E. Subroutines used

- rcv1byte(): Receives one byte of data.
- rcvnbyte(): Receives n bytes of data.
- trs1byte(): Transmits one byte of data.

fwrite128(): Programs and verifies 128 bytes.
 blk_check(): Determines the erase block number from the erase start address.
 blk1_erase(): Erases the specified blocks in flash memory.

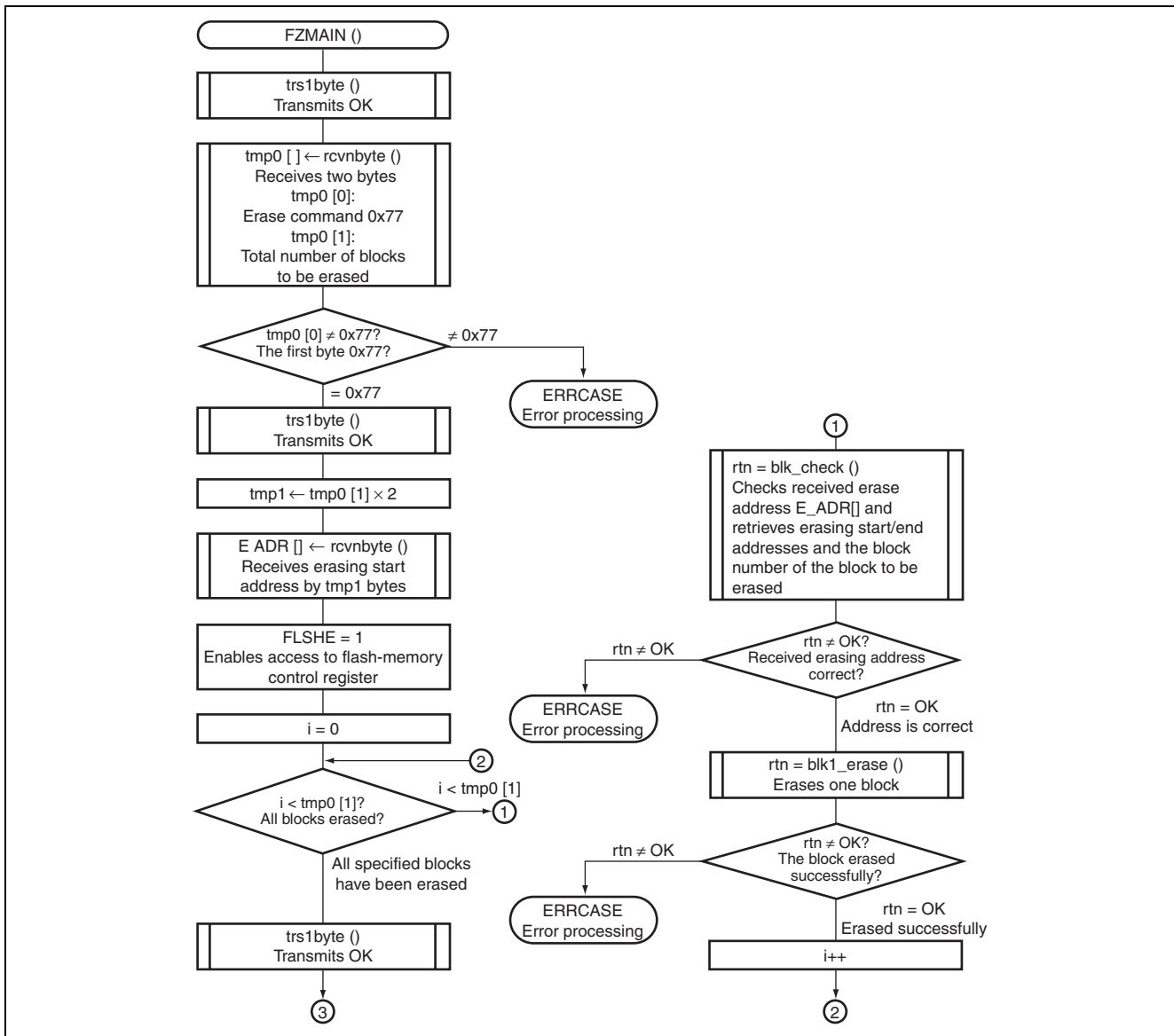
F. Internal registers used

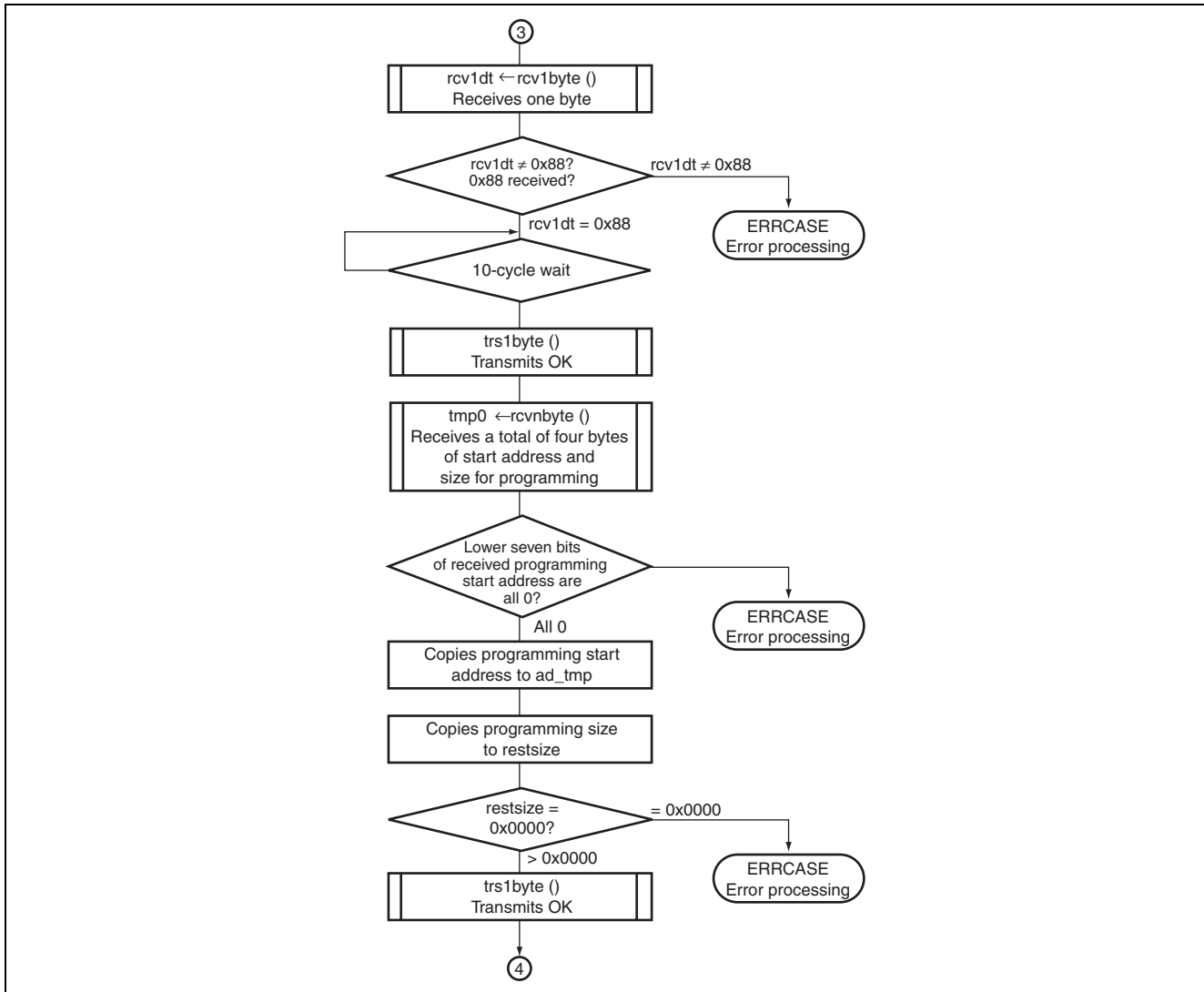
Table 6.3 Registers Used by FZMAIN() Function

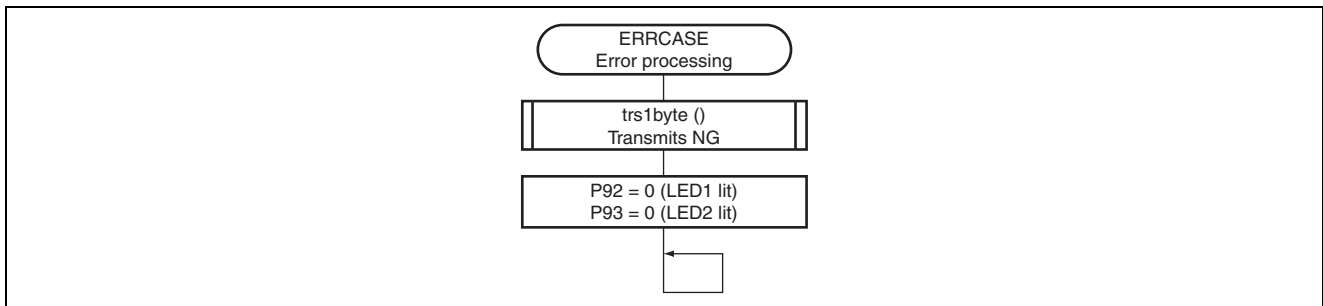
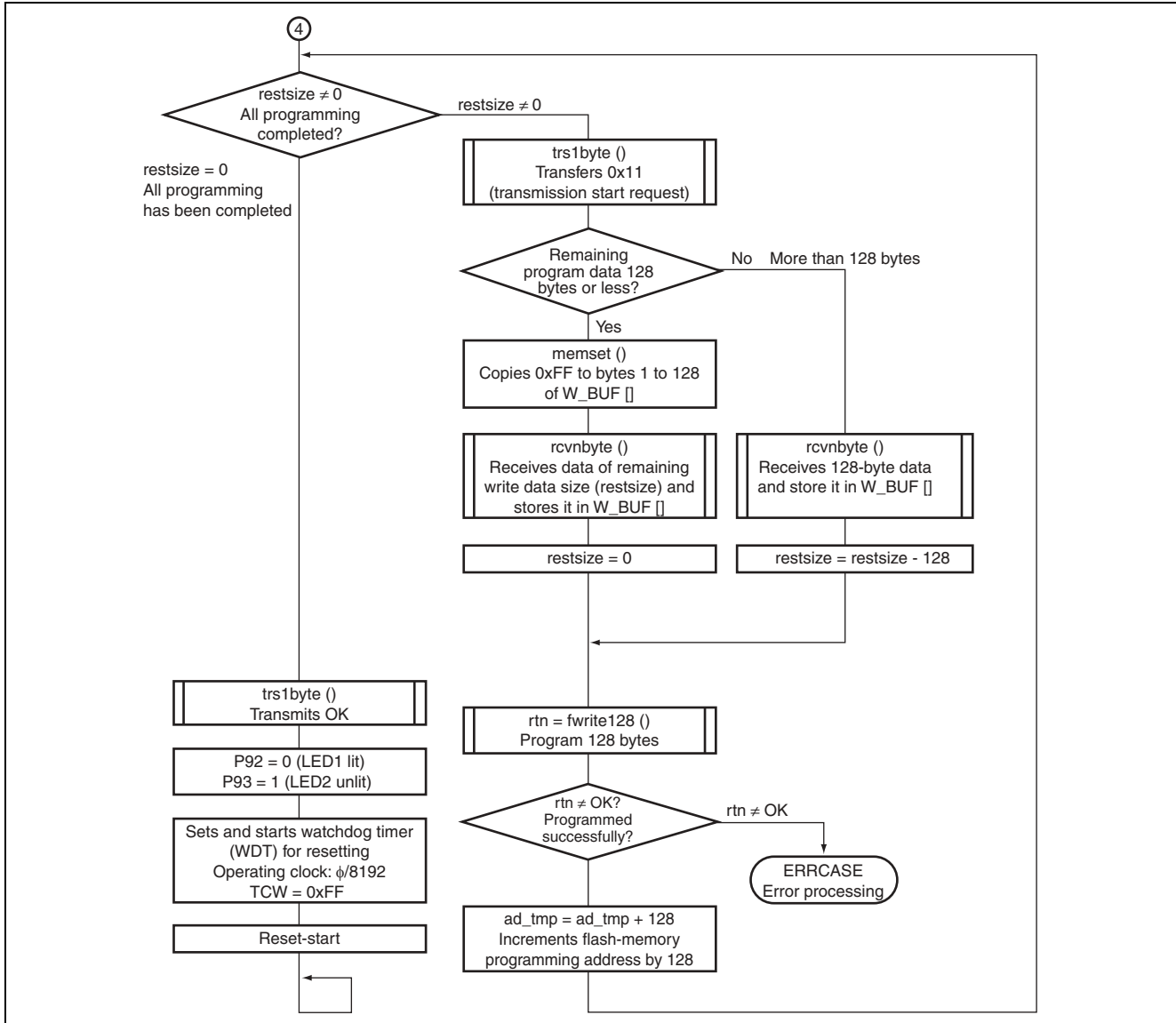
Register	Function	Address	Setting
FENR FLSHE	Flash memory enable register (flash memory control register enable) When FLSHE = 0, the flash memory control register can be accessed. When FLSHE = 1, the flash memory control register cannot be accessed.	0xF02B Bit 7	1
TCSRW B6WI	Timer control/status register W (Bit 6 write disable) When B6WI = 0, writing to bit 6 in TCSRW is enabled. When B6WI = 1, writing to bit 6 in TCSRW is disabled.	0xFFC0 Bit 7	0
TCWE	Timer control/status register W (timer counter W write enable) When TCWE = 1, writing 8-bit data to TCW is enabled.	0xFFC0 Bit 6	1
B4WI	Timer control/status register W (Bit 4 write disable) When B4WI = 0, writing to bit 4 in TCSRW is enabled. When B4WI = 1, writing to bit 4 in TCSRW is disabled.	0xFFC0 Bit 5	0
TCSRWE	Timer control/status register W (timer control/status register W write enable) When TCSRWE = 1, writing to bits 2 and 0 in TCSRW is enabled.	0xFFC0 Bit 4	1

Register		Function	Address	Setting
TCSRW	B2WI	Timer control/status register W (Bit 2 write disable) When B2WI = 0, writing to bit 2 in TCSRW is enabled. When B2WI = 1, writing to bit 2 in TCSRW is disabled.	0xFFC0 Bit 3	0
	WDON	Timer control/status register W (watchdog timer on) When WDON = 0, the watchdog timer is disabled. When WDON = 1, the watchdog timer is enabled.	0xFFC0 Bit 2	0
	B0WI	Timer control/status register W (Bit 0 write disable) When B0WI = 0, writing to bit 0 in TCSRW is enabled. When B0WI = 1, writing to bit 0 in TCSRW is disabled.	0xFFC0 Bit 1	0
	WRST	Timer control/status register W (watchdog timer reset) When WRST = 0, indicates that a TCW overflow has not occurred and an internal reset signal is not generated. When WRST = 1, indicates that a TCW overflow occurred and an internal reset signal has been generated.	0xFFC0 Bit 0	0
TCW	Timer counter W 8-bit counter that uses system clock divided by 8192 as input	0xFFC1	0xFF	
PDR9	P93	Port data register 9 (port data register 93) When P93 = 0, the output level of the P93 pin is low. When P93 = 1, the output level of the P93 pin is high.	0xFFDC Bit 3	1
	P92	Port data register 9 (port data register 92) When P92 = 0, the output level of the P92 pin is low. When P92 = 1, the output level of the P92 pin is high.	0xFFDC Bit 2	0
PMR2	WDCKS	Port mode register 2 (watchdog timer source clock) When WDCKS = 0, ϕ (system clock)/8192 is selected as the source clock of the watchdog timer. When WDCKS = 1, ϕ_w (subclock)/32 is selected as the source clock of the watchdog timer.	0xFFE0 Bit 2	0

G. Flowchart





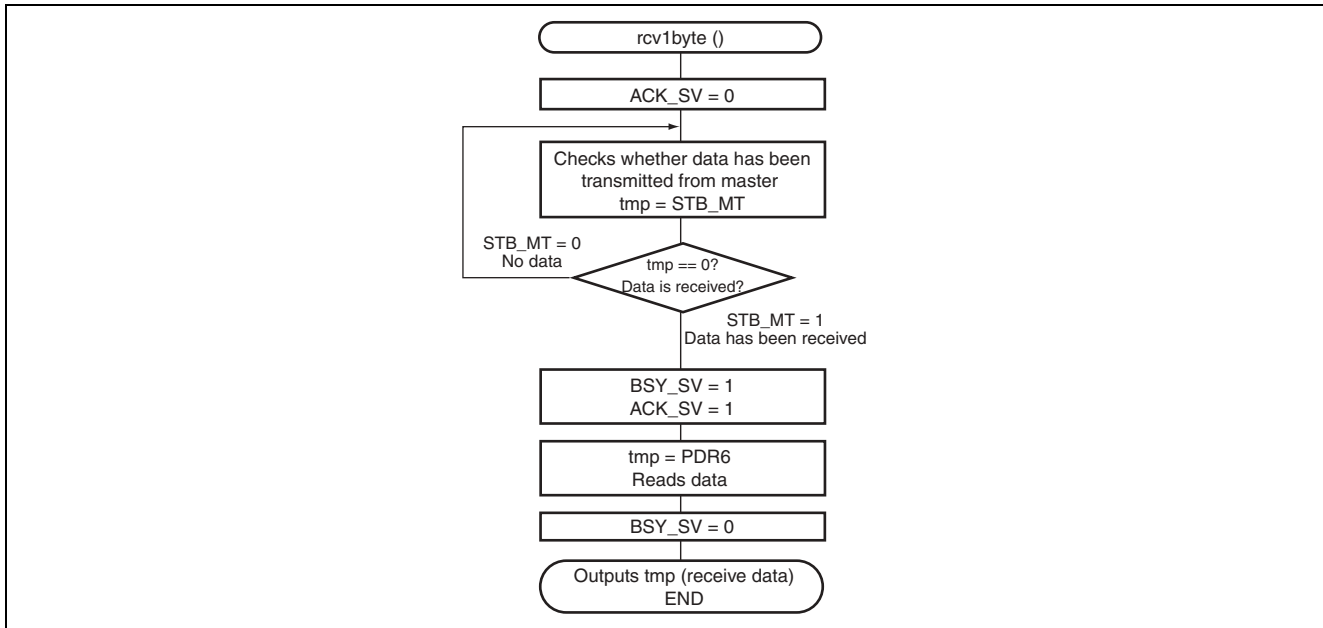


- 2. rcv1byte() function
 - A. Specifications
 - unsigned char rcv1byte(void)
 - B. Operation
 - Receives one byte of 8-bit parallel data.
 - C. Arguments
 - Input: None
 - Output: One byte of received data
 - D. Global variables
 - None
 - E. Subroutines used
 - None
 - F. Internal registers used

Table 6.4 Registers Used by rcv1byte() Function

Register	Function	Address	Setting
PDR5 P55	Port data register 5 (port data register 55) A port used as ACK_SV: When ACK_SV (P55) = 0, the slave can receive data. When ACK_SV (P55) = 1, the slave cannot receive data.	0xFFD8 Bit 5	0
P54	Port data register 5 (port data register 54) A port used as BSY_SV: When BSY_SV (P54) = 0, internal processing on the slave side ended. When BSY_SV (P54) = 1, internal processing on the slave side is in progress.	0xFFD8 Bit 4	1
P50	Port data register 5 (port data register 50) A port used as STB_MT: When STB_MT (P50) = 0, the master is not transmitting data. When STB_MT (P50) = 1, the master is transmitting data.	0xFFD8 Bit 0	—
PDR6	Port data register 6 An 8-bit register that stores receive data.	0xFFD9	—

G. Flowchart



3. rcvnbyte() function

A. Specifications

```
void rcvnbyte(
    unsigned char dtno,
    unsigned char *ram
)
```

B. Operation

— Receives n bytes of 8-bit parallel data.

C. Arguments

— Input:

dtno: Number of receive bytes

*ram: RAM start address to store receive data

— Output: One byte of receive data

*ram: Receive data

D. Global variables

None

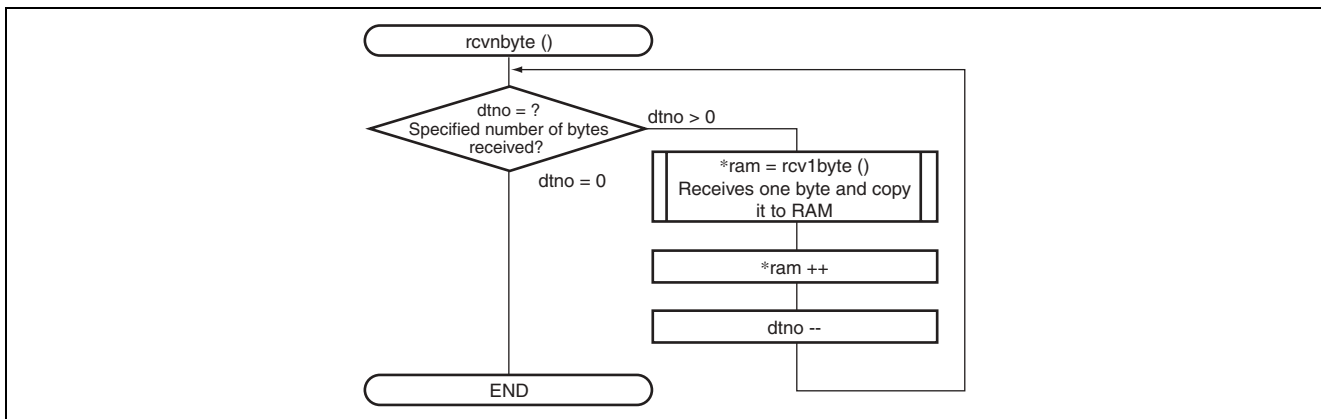
E. Subroutine used

rcv1byte: Receives one byte of data.

F. Internal registers used

None

G. Flowchart

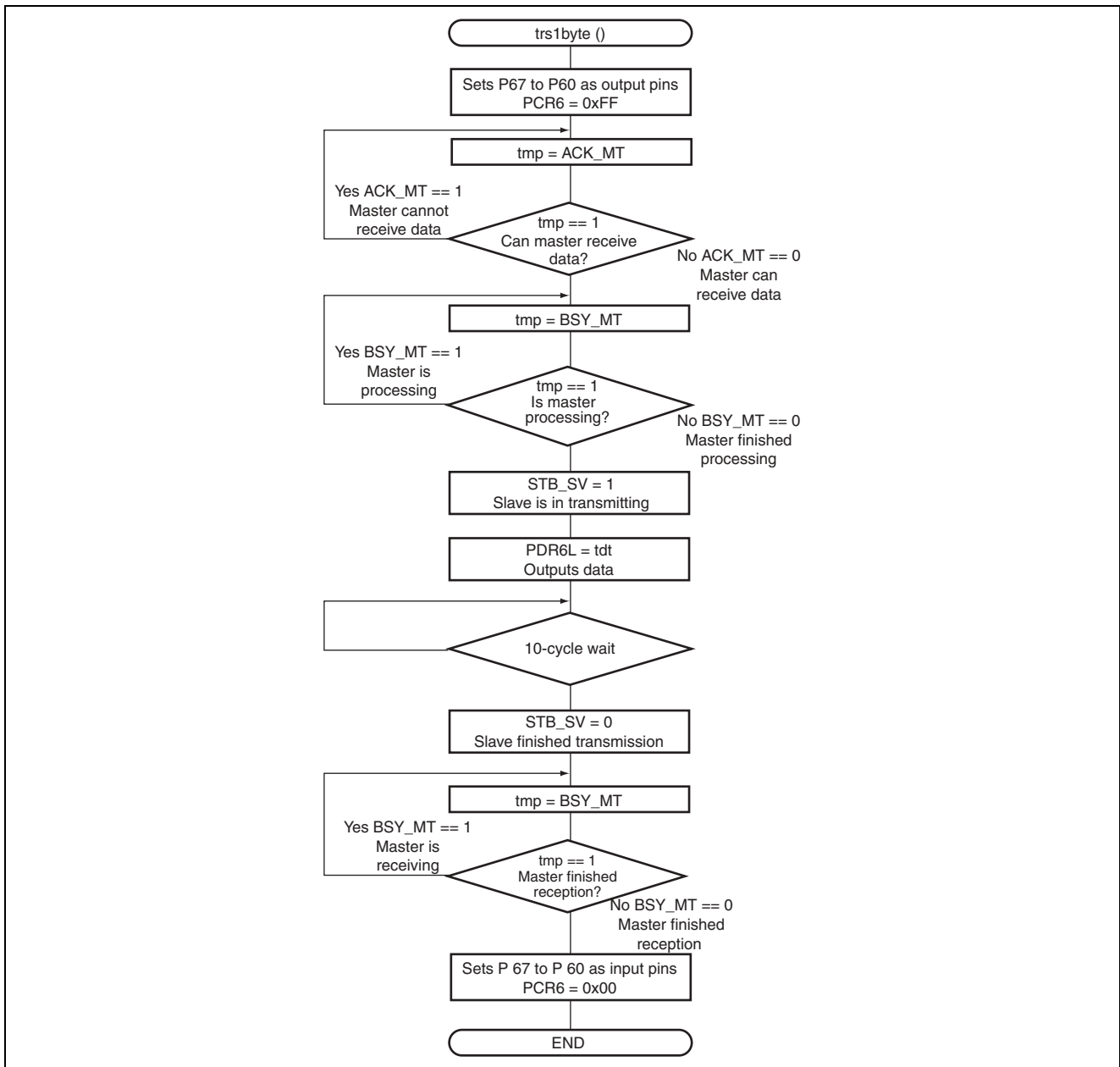


4. trs1byte() function
 - A. Specifications
void trs1byte(unsigned char tdt)
 - B. Operation
— Transmits one byte of 8-bit parallel data.
 - C. Arguments
— Input:
tdt: One byte of transmit data
— Output: None
 - D. Global variables
None
 - E. Subroutines used
None
 - F. Internal registers used

Table 6.5 Registers Used by trs1byte() Function

Register	Function	Address	Setting
PDR5 P53	Port data register 5 (port data register 53) A port used as STB_SV: When STB_SV (P53) = 0, the slave is not transmitting data. When STB_SV (P53) = 1, the slave is transmitting data.	0xFFD8 Bit 3	1
P52	Port data register 5 (port data register 52) A port used as ACK_MT: When ACK_MT (P52) = 0, the master can receive data. When ACK_MT (P52) = 1, the master cannot receive data.	0xFFD8 Bit 2	—
P51	Port data register 5 (port data register 51) A port used as BSY_MT: When BSY_MT (P51) = 0, internal processing on the master side ended. When BSY_MT (P51) = 1, internal processing on the master side is in progress.	0xFFD8 Bit 1	—
PDR6	Port data register 6 An 8-bit register that stores transmit data.	0xFFD9	—
PCR6	Port control register 6 When PCR6 = 0xFF, P67 to P60 are set as output pins.	0xFFE9	0xFF

G. Flowchart



6.5 Description of Functions for Flash-Memory Program/Erase Processing

1. blk_check() function

A. Specifications

```
char blk_check(
    unsigned short ersad,
    unsigned short *evf_st,
    unsigned short *evf_ed,
    unsigned char *blk_no
)
```

B. Operation

- Determines the erase block number from the erase start address.
- Compares the received erase start address with BLOCKADR1[] to determine whether the address is correct and returns the result flag, erase start address, erase end address, and erase block number.

C. Arguments

— Input:

ersad: Erase start address
 *evf_st: Erase start address after verification
 *evf_ed: Erase end address after verification
 *blk_no: Block number of the block to be erased

— Output:

Return value: Result flag (OK = 0x00, NG = 0x01)
 *evf_st: Erase start address after verification
 *evf_ed: Erase end address after verification
 *blk_no: Block number of the block to be erased

D. Global variable

BLOCKADR1[]: Stores the start and end addresses of each block of flash memory.

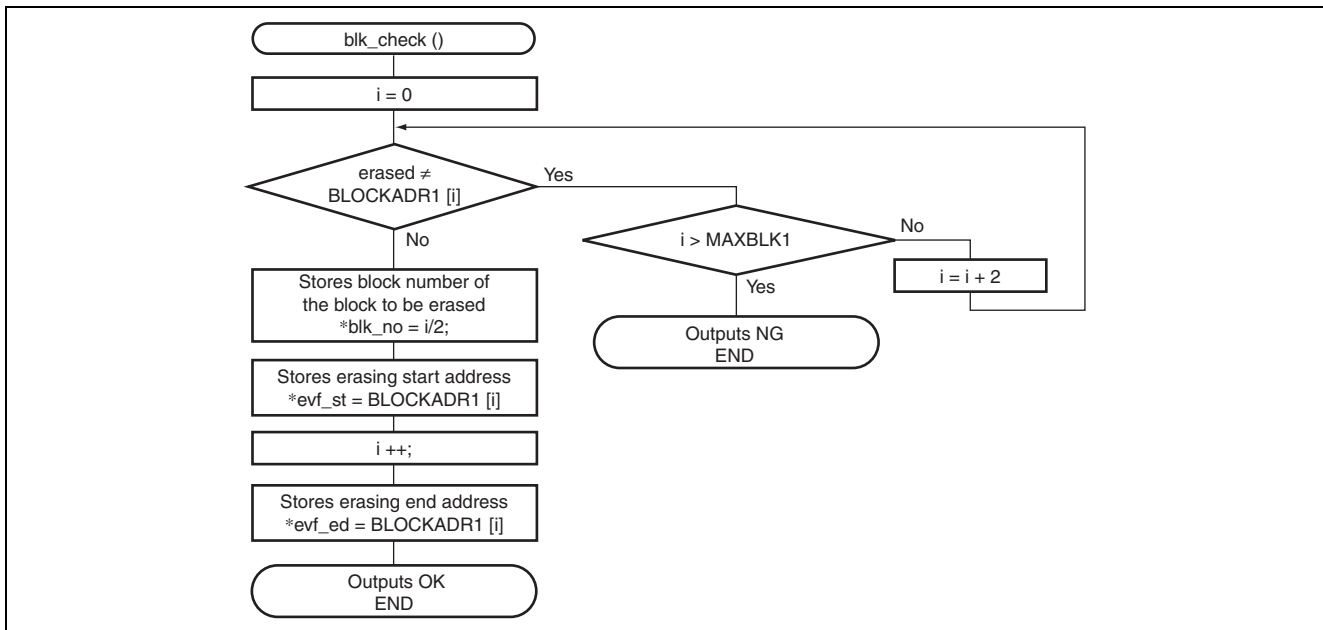
E. Subroutines used

None

F. Internal registers used

None

G. Flowchart



2. blk1_erase() function

A. Specifications

```

char blk1_erase(
    unsigned short evf_st,
    unsigned short evf_ed,
    unsigned char blk_no,
    unsigned char ET_COUNT
)
  
```

B. Operation

— Erases the specified block in flash memory.

C. Arguments

— Input:

evf_st: Erase start address
evf_ed: Erase end address
blk_no: Bit number for the erase target block
ET_COUNT: Maximum number of erases

— Output:

Return value: Result flag (OK = 0x00, NG = 0x01)

D. External RAM

None

E. Subroutines used

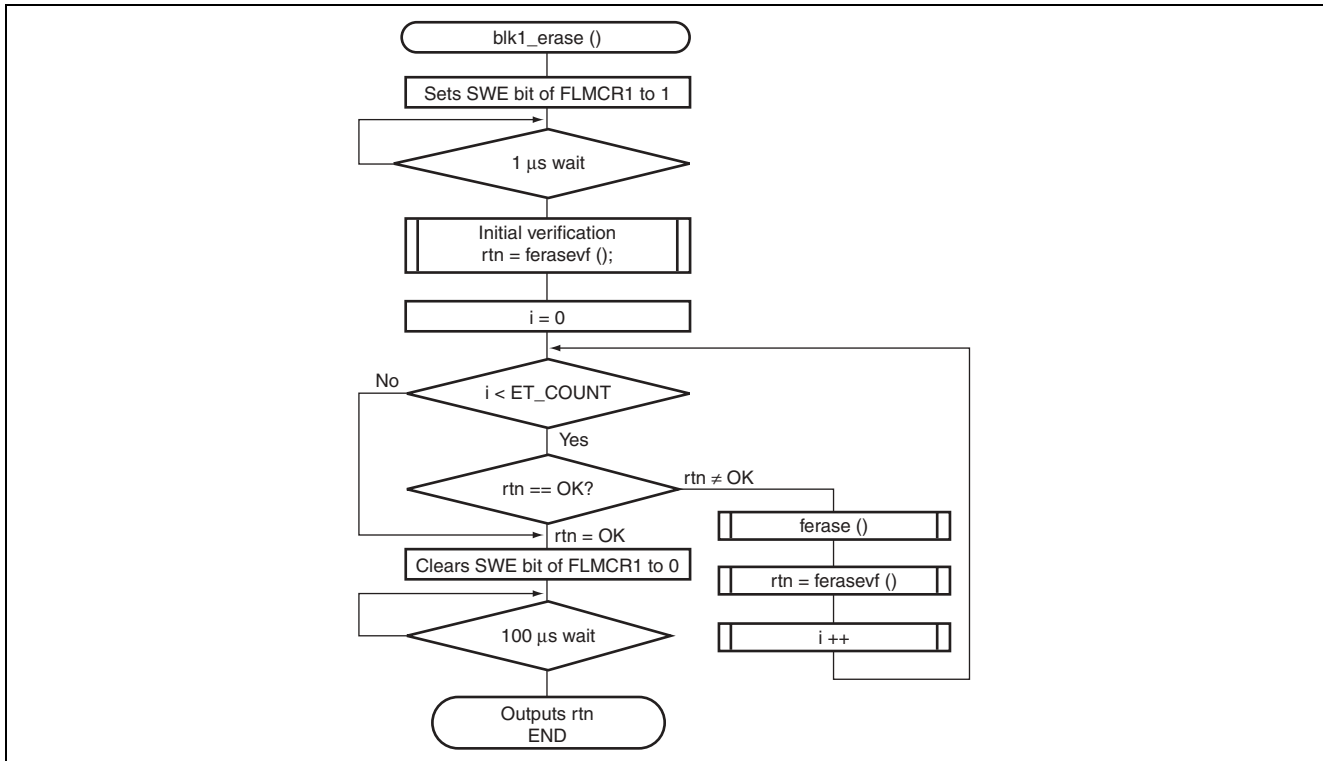
ferase(): Erases the specified block.
ferasevf(): Verifies that the specified block has been erased.

F. Internal register used

Table 6.6 Register Used by blk1_erase() Function

Register	Function	Address	Setting
FLMCR1	SWE	0xF020	1
		Bit 6	
	When SWE = 0, flash-memory programming/erasing is disabled.		
	When SWE = 1, flash-memory programming/erasing is enabled.		

G. Flowchart



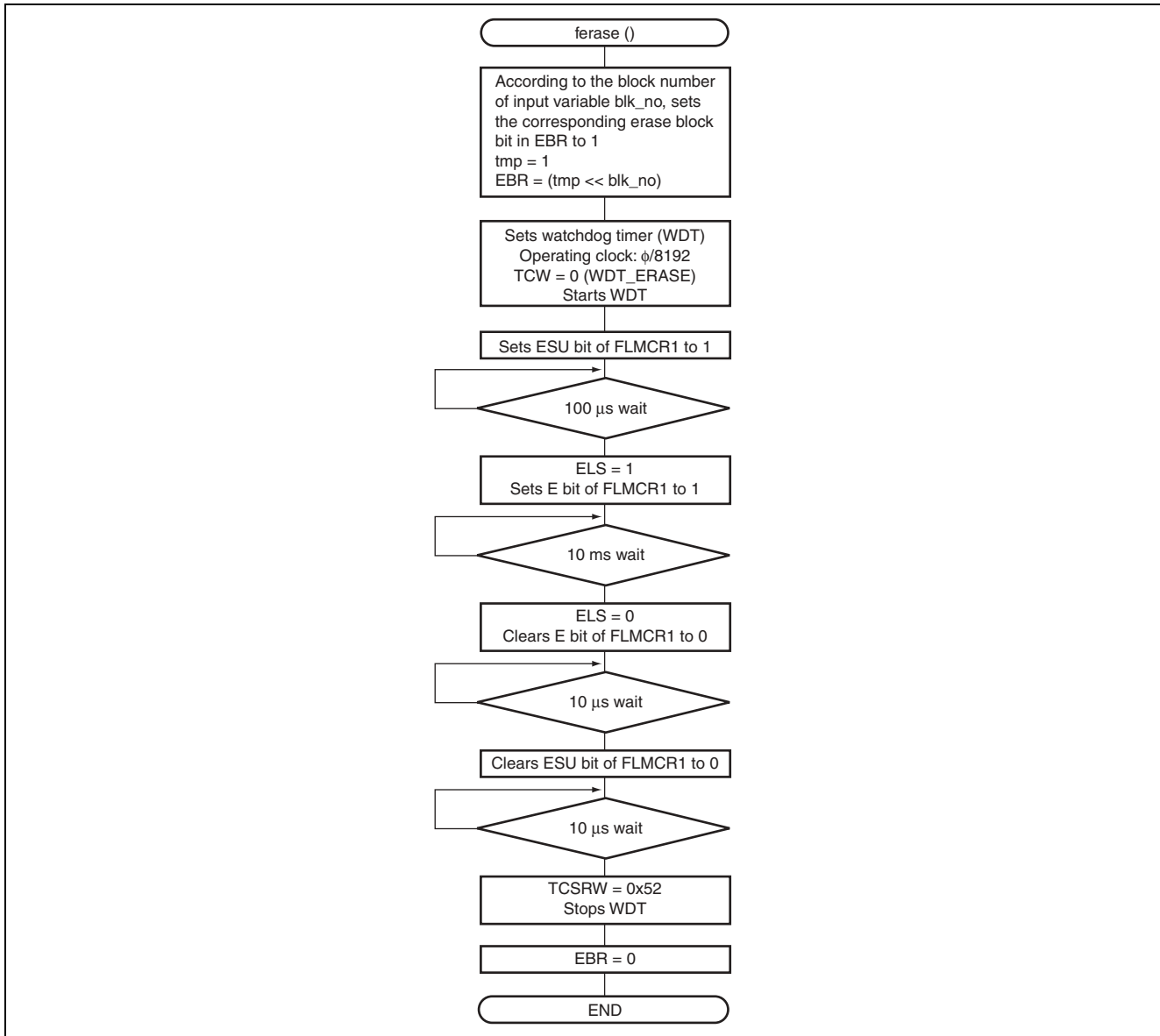
3. `ferase()` function
 - A. Specifications
 - `void ferase(unsigned char blk_no)`
 - B. Operation
 - Erases the specified block in flash memory.
 - C. Arguments
 - Input:
 - `blk_no`: Block number of the block to be erased
 - Output:
 - None
 - D. Global variables
 - None
 - E. Subroutines used
 - None
 - F. Internal registers used

Table 6.7 Registers Used by `ferase()` Function

Register		Function	Address	Setting
FLMCR1	ESU	Flash memory control register 1 (erase setup) When ESU = 0, the erase setup state is canceled. When ESU = 1, the erase setup state is entered.	0xF020 Bit 5	1
	E	Flash memory control register 1 (erase) When E = 0, erase mode is canceled. When SWE = 1, ESU = 1, and E = 1, erase mode is entered.	0xF020 Bit 1	1
EBR	EB4	Erase block register When any one of EB4 to EB0 bits is set to 1, the corresponding flash memory block can be erased.	0xF023	—
	EB3			
	EB2			
	EB1			
	EB0			
TCSRW	B6WI	Timer control/status register W (Bit 6 write disable) When B6WI = 0, writing to bit 6 in TCSRW is enabled. When B6WI = 1, writing to bit 6 in TCSRW is disabled.	0xFFC0 Bit 7	0
	TCWE	Timer control/status register W (timer counter W write enable) When TCWE = 1, writing 8-bit data to TCW is enabled.	0xFFC0 Bit 6	1
	B4WI	Timer control/status register W (Bit 4 write disable) When B4WI = 0, writing to bit 4 in TCSRW is enabled. When B4WI = 1, writing to bit 4 in TCSRW is disabled.	0xFFC0 Bit 5	0
	TCSRWE	Timer control/status register W (timer control/status register W write enable) When TCSRWE = 1, writing to bits 2 and 0 in TCSRW is enabled.	0xFFC0 Bit 4	1

Register		Function	Address	Setting
TCSRW	B2WI	Timer control/status register W (Bit 2 write disable) When B2WI = 0, writing to bit 2 in TCSRW is enabled. When B2WI = 1, writing to bit 2 in TCSRW is disabled.	0xFFC0 Bit 3	0
	WDON	Timer control/status register W (watchdog timer on) When WDON = 0, the watchdog timer is disabled. When WDON = 1, the watchdog timer is enabled.	0xFFC0 Bit 2	0
	B0WI	Timer control/status register W (Bit 0 write disable) When B0WI = 0, writing to bit 0 in TCSRW is enabled. When B0WI = 1, writing to bit 0 in TCSRW is disabled.	0xFFC0 Bit 1	0
	WRST	Timer control/status register W (watchdog timer reset) When WRST = 0, indicates that a TCW overflow has not occurred and an internal reset signal is not generated. When WRST = 1, indicates that a TCW overflow occurred and an internal reset signal has been generated.	0xFFC0 Bit 0	0
TCW		Timer counter W 8-bit counter that uses system clock divided by 8192 as input	0xFFC1	0x00
PMR2	WDCKS	Port mode register 2 (watchdog timer source clock) When WDCKS = 0, ϕ (system clock)/8192 is selected as the source clock of the watchdog timer. When WDCKS = 1, ϕ_w (subclock)/32 is selected as the source clock of the watchdog timer.	0xFFE0 Bit 2	0

G. Flowchart



4. `ferasevf()` function

A. Specifications

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

B. Operation

— Verifies that the specified block of flash memory has been erased.

C. Arguments

— Input:

`evf_st`: Erase start address

`evf_ed`: Erase end address

— Output:

Return value: Result flag (OK = 0x00, NG = 0x01)

D. Global variables

None

E. Subroutines used

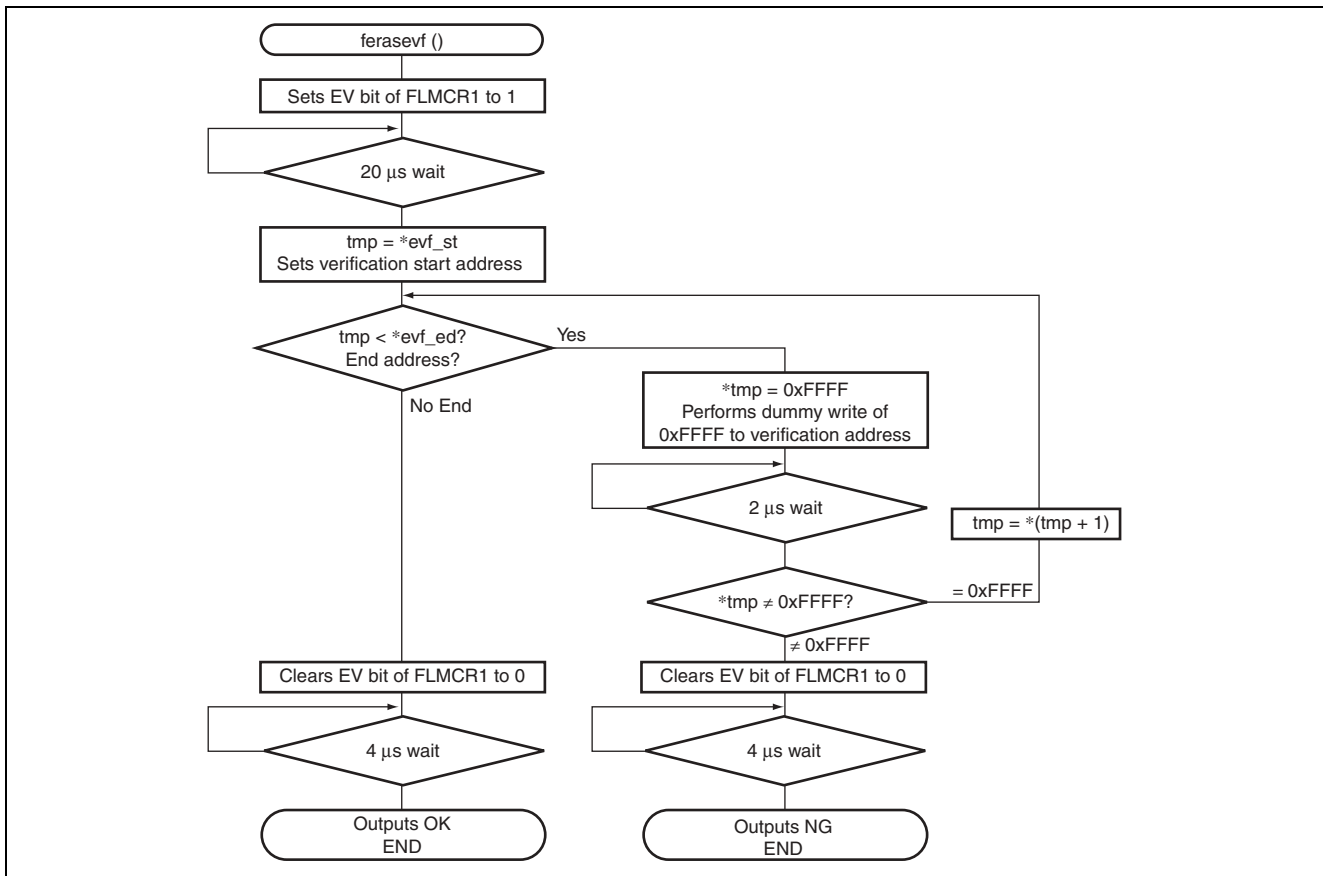
None

F. Internal register used

Table 6.8 Register Used by `ferasevf()` Function

Register	Function	Address	Setting
FLMCR1 EV	Flash memory control register 1 (erase verification) When EV = 0, erase-verify mode is canceled. When EV = 1, erase-verify mode is entered.	0xF020 Bit 3	1

G. Flowchart



5. fwrite128() function

A. Specifications

```
char fwrite128(
    unsigned char *BUFF,
    unsigned char *OWBUFF,
    unsigned char *w_adr,
    unsigned char *w_buf,
    unsigned short WT_COUNT
)
```

B. Operation

— Programs and verifies 128 bytes.

C. Arguments

— Input:

*BUFF: Program data buffer
 *OWBUFF: Additional program data buffer
 *w_adr: Program address
 *w_buf: 128 bytes of program data
 WT_COUNT: Maximum number of programmings

— Output:

Return value: Result flag (OK = 0x00, NG = 0x01, WNG = 0x02)
 *BUFF: Program data buffer
 *OWBUFF: Additional program data buffer
 *w_adr: Program address
 *w_buf: 128 bytes of program data

D. Global variables

None

E. Subroutines used

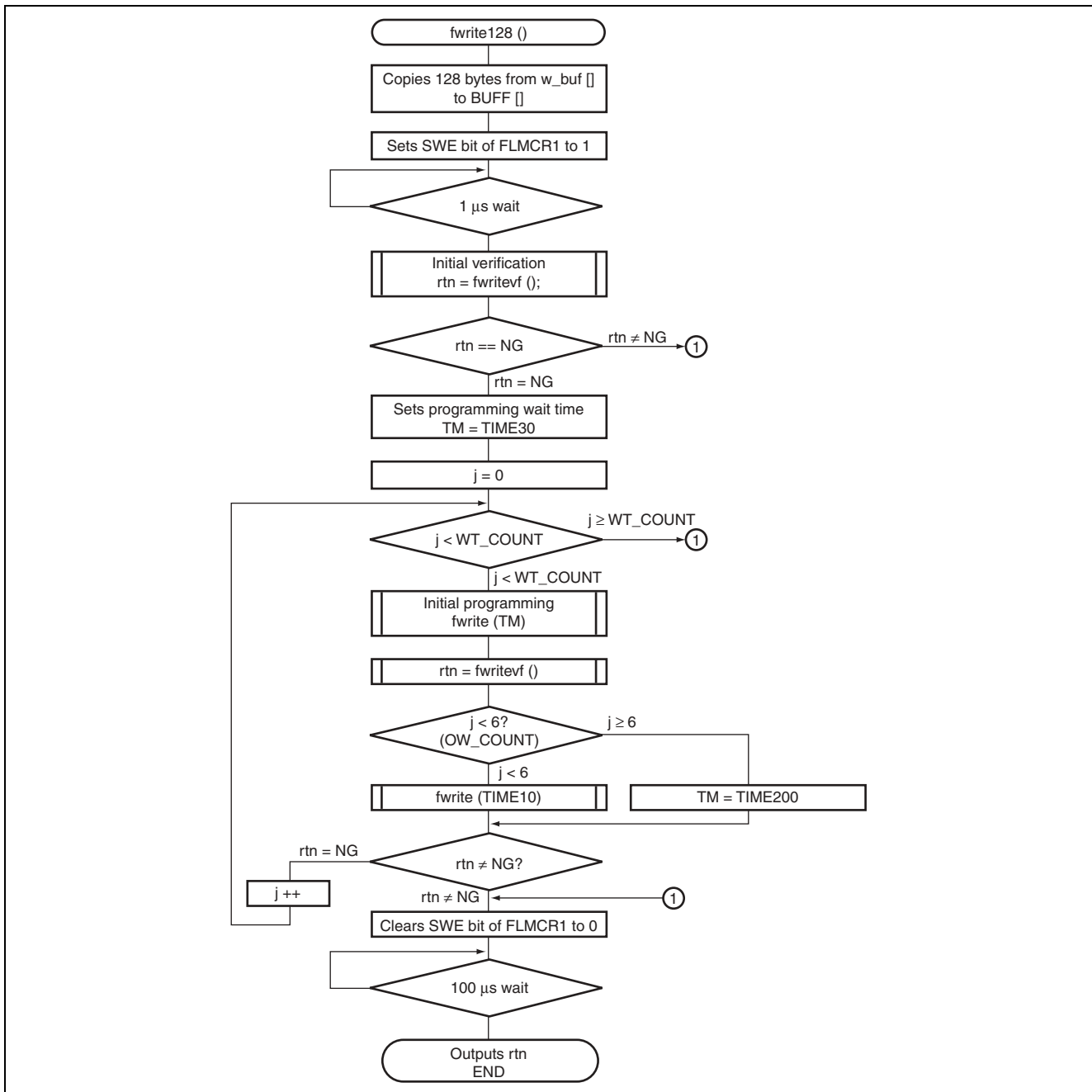
fwrite: Programs data to the specified address.
 fwritevf: Verifies data for the specified address and creates reprogram data.

F. Internal register used

Table 6.9 Register Used by fwrite128() Function

Register	Function	Address	Setting
FLMCR1	SWE	Flash memory control register 1 (software write enable) When SWE = 0, flash-memory programming/erasing is disabled. When SWE = 1, flash-memory programming/erasing is enabled.	0xF020 Bit 6 1

G. Flowchart



6. fwrite() function

A. Specifications

```
void fwrite(
    unsigned char *buf,
    unsigned char *w_adr,
    unsigned char ptime
)
```

B. Operation

— Programs data to the specified address.

C. Arguments

— Input:

*buf: Start address of the program data (reprogram data or additional program data)

*w_adr: Program address

ptime: P-Bit setting time (10 μ s, 30 μ s, or 2000 μ s)

— Output:

None

D. Global variables

None

E. Subroutines used

None

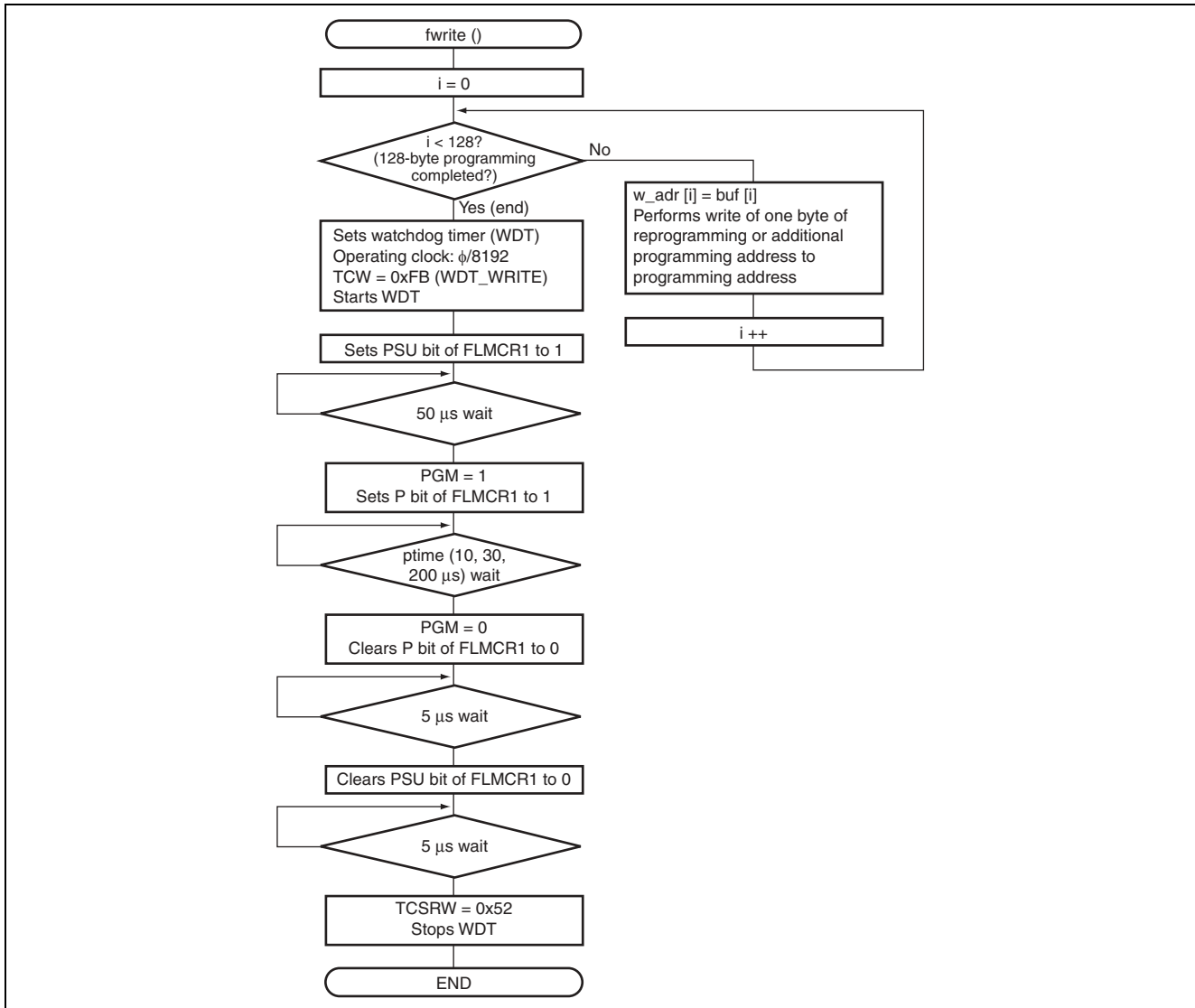
F. Internal registers used

Table 6.10 Registers Used by fwrite() Function

Register	Function	Address	Setting	
FLMCR1	PSU	Flash memory control register 1 (program setup) When PSU = 0, the program setup state is canceled. When PSU = 1, the program setup state is entered.	0xF020 Bit 4	1
	P	Flash memory control register 1 (program) When P = 0, program mode is canceled. When SWE = 1, PSU = 1, and P = 1, program mode is entered.	0xF020 Bit 0	1
TCSRW	B6WI	Timer control/status register W (Bit 6 write disable) When B6WI = 0, writing to bit 6 in TCSRW is enabled. When B6WI = 1, writing to bit 6 in TCSRW is disabled.	0xFFC0 Bit 7	0
	TCWE	Timer control/status register W (timer counter W write enable) When TCWE = 1, writing 8-bit data to TCW is enabled.	0xFFC0 Bit 6	1
	B4WI	Timer control/status register W (Bit 4 write disable) When B4WI = 0, writing to bit 4 in TCSRW is enabled. When B4WI = 1, writing to bit 4 in TCSRW is disabled.	0xFFC0 Bit 5	0
	TCSRWE	Timer control/status register W (timer control/status register W write enable) When TCSRWE = 1, writing to bits 2 and 0 in TCSRW is enabled.	0xFFC0 Bit 4	1

Register	Function	Address	Setting
TCSRW	B2WI Timer control/status register W (Bit 2 write disable) When B2WI = 0, writing to bit 2 in TCSRW is enabled. When B2WI = 1, writing to bit 2 in TCSRW is disabled.	0xFFC0 Bit 3	0
	WDON Timer control/status register W (watchdog timer on) When WDON = 0, the watchdog timer is disabled. When WDON = 1, the watchdog timer is enabled.	0xFFC0 Bit 2	0
	B0WI Timer control/status register W (Bit 0 write disable) When B0WI = 0, writing to bit 0 in TCSRW is enabled. When B0WI = 1, writing to bit 0 in TCSRW is disabled.	0xFFC0 Bit 1	0
	WRST Timer control/status register W (watchdog timer reset) When WRST = 0, indicates that a TCW overflow has not occurred and an internal reset signal is not generated. When WRST = 1, indicates that a TCW overflow occurred and an internal reset signal has been generated.	0xFFC0 Bit 0	0
TCW	Timer counter W 8-bit counter that uses system clock divided by 8192 as input	0xFFC1	0xFB
PMR2	WDCKS Port mode register 2 (watchdog timer source clock) When WDCKS = 0, ϕ (system clock)/8192 is selected as the source clock of the watchdog timer. When WDCKS = 1, ϕ_w (subclock)/32 is selected as the source clock of the watchdog timer.	0xFFE0 Bit 2	0

G. Flowchart



7. fwritevf() function

A. Specifications

```
char fwritevf(
    unsigned char *owbuff,
    unsigned char *buff,
    unsigned char *w_adr,
    unsigned char *w_buf
)
```

B. Operation

— Verifies data for the specified address and creates reprogram data.

C. Arguments

— Input:

*owbuff: 128 bytes of additional program data

*buff: 128 bytes of reprogram data

*w_adr: Program address

*w_buf: 128 bytes of program data

— Output:

Return value: Result flag (OK = 0x00, NG = 0x01, WNG = 0x02)

*owbuff: 128 bytes of additional program data

*buff: 128 bytes of reprogram data

*w_adr: Program address

*w_buf: 128 bytes of program data

D. Global variables

None

E. Subroutines used

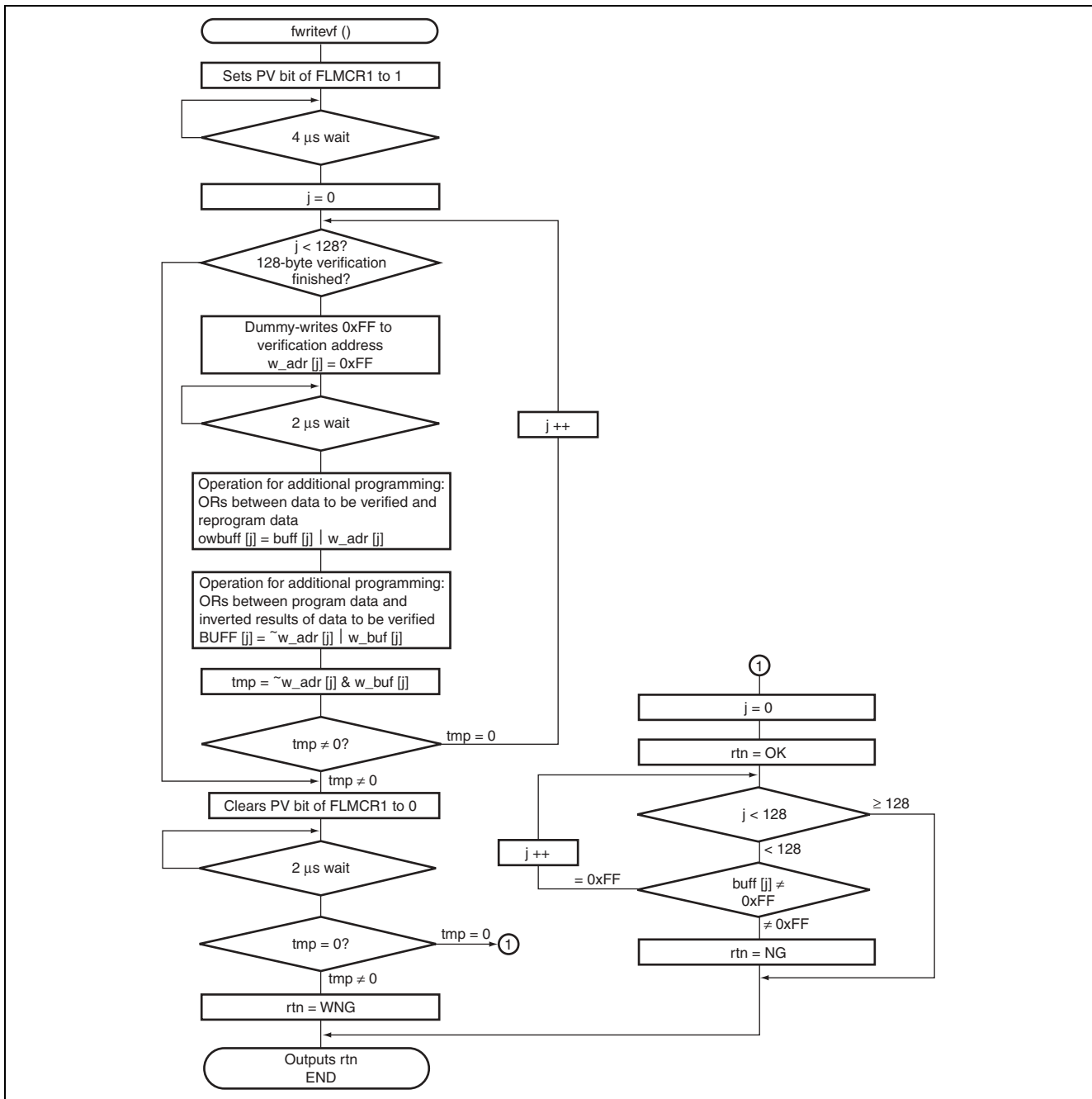
None

F. Internal register used

Table 6.11 Register Used by fwritevf() Function

Register	Function	Address	Setting
FLMCR1 PV	Flash memory control register 1 (program verification) When PV = 0, program-verify mode is canceled. When PV = 1, program-verify mode is entered.	0xF020 Bit 2	1

G. Flowchart



7. Program on Master Side

7.1 Hierarchical Structure

Through communication with the slave device, the program on the master device sends commands for erasing/programming of flash memory of the slave device and transmits program data. Figure 7.1 shows the hierarchical structure of the routines used in the program for the master device.

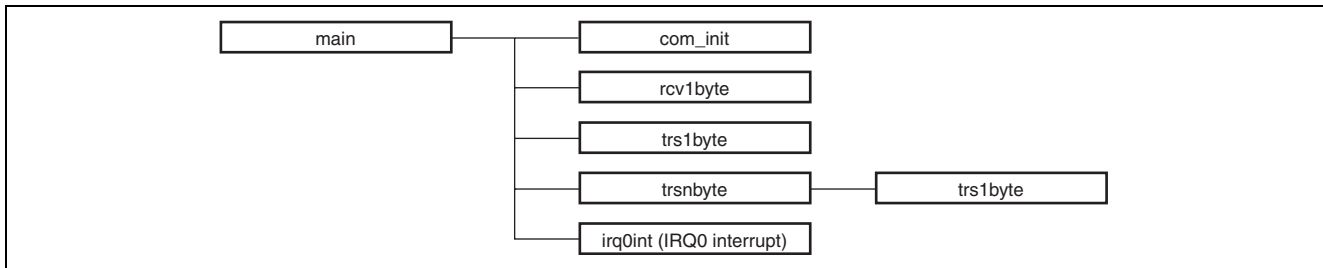


Figure 7.1 Program/Erase Control Program on Master Side

7.2 Functions

Table 7.1 Functions of Program/Erase Control Program

Function	Overview
main	Main routine of the program/erase control program
com_init	Initializes the communication settings.
rcv1byte	Receives one byte of data.
trs1byte	Transmits one byte of data.
trsnbyte	Transmits n bytes of data.
irq0int	IRQ0 interrupt processing routine that sets a flag for transition to the processing for transmission of the program start command.

7.3 Constants

Table 7.2 Constants

Constant	Value	Description
OK	0x00	Return value for normal operation
NG	0x01	Return value for abnormal operation

7.4 Description of Functions

1. main() function

A. Specifications

void main(void)

B. Operation

— Starts issuing commands and transmits/receives data to/from the slave device.

C. Arguments

— Input: None

— Output: None

D. Global variables

ramf: Determines whether to branch to the processing for transmitting the program start command.

ramf = 0: Branches to the processing for transmitting the program start command.

ramf = 1: Executes the sample normal application.

E. Subroutines used

com_init(): Initializes the communication settings.

rcv1byte(): Receives one byte of data.

trs1byte(): Transmits one byte of data.

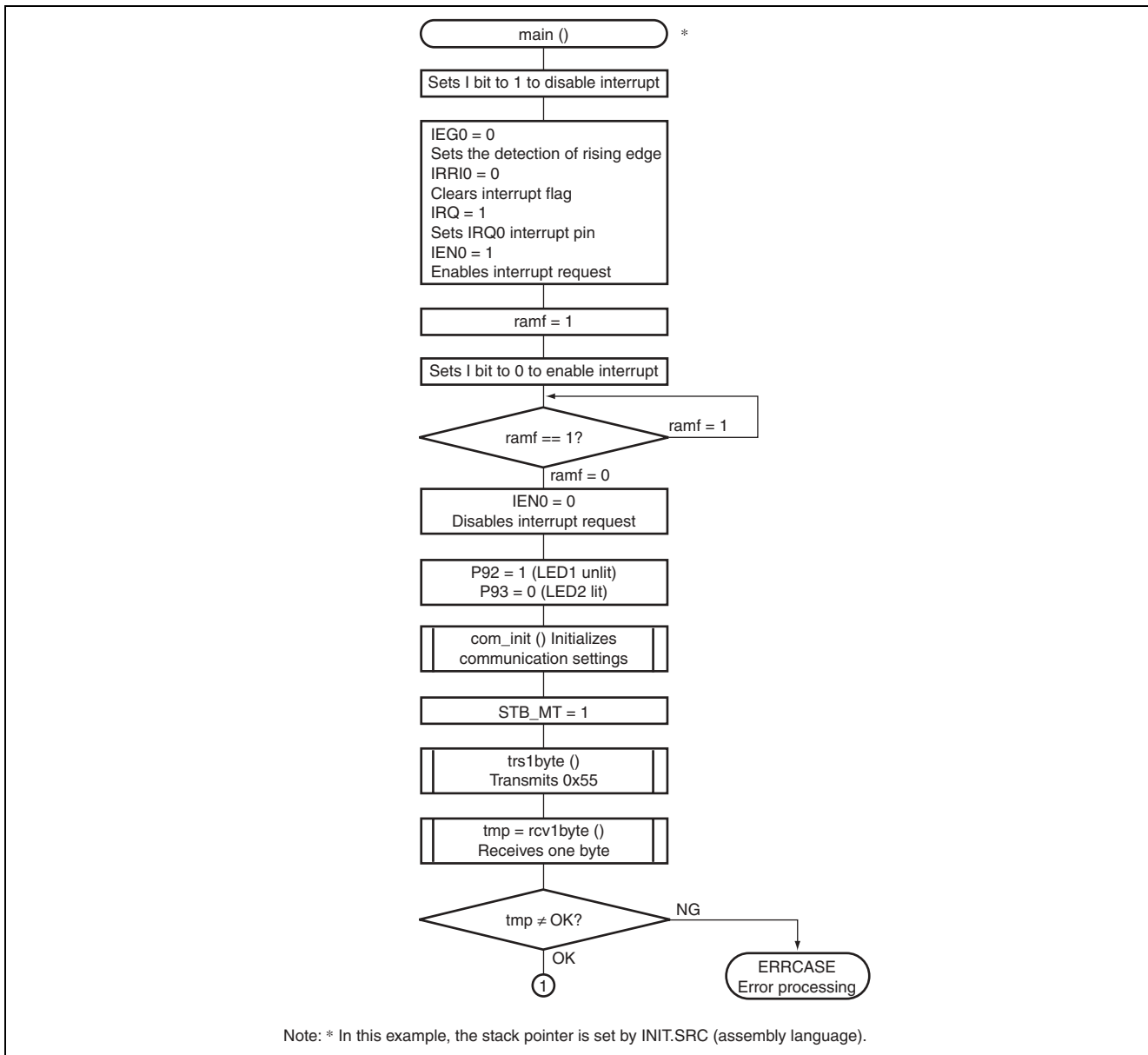
trsnbyte(): Transmits n bytes of data.

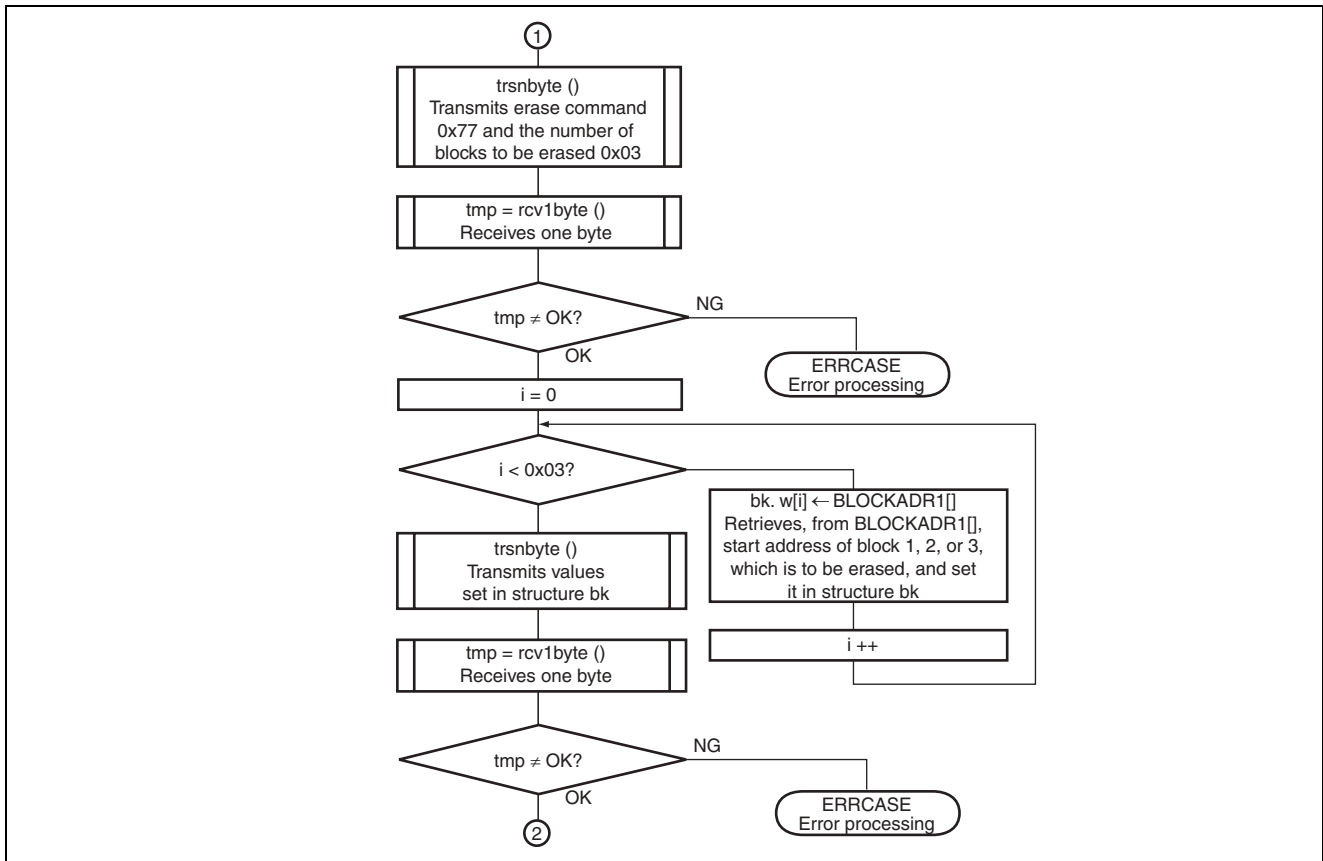
F. Internal registers used

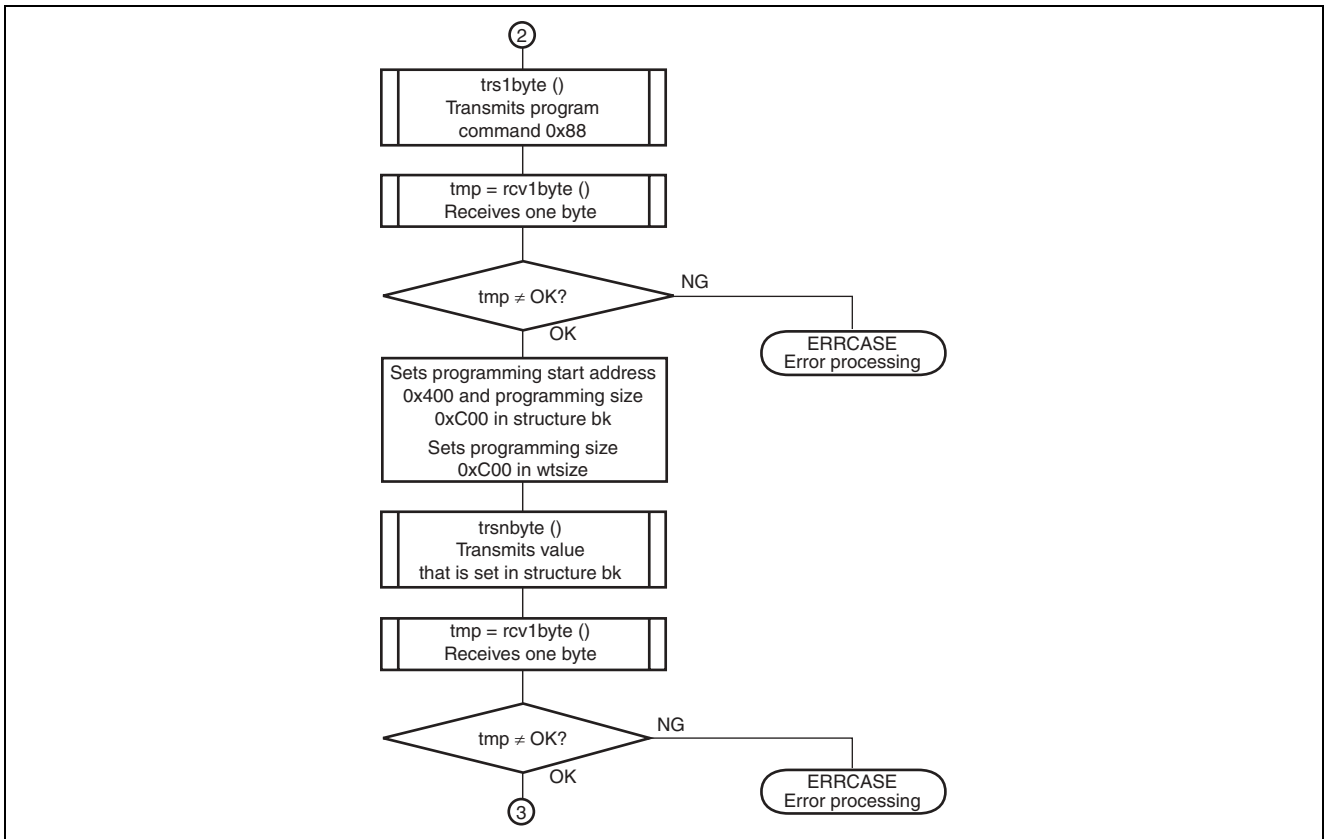
Table 7.3 Registers Used by main() Function

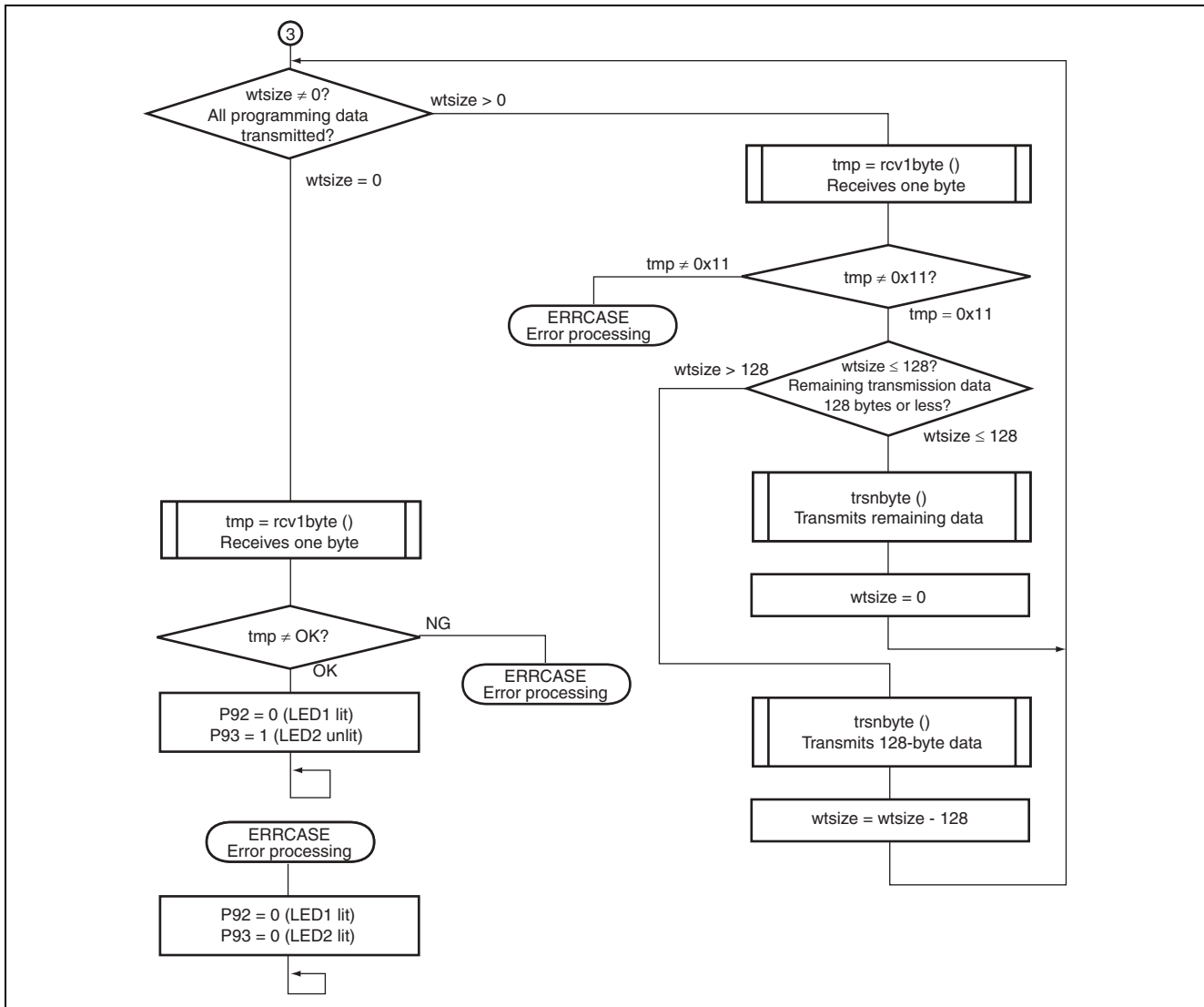
Register	Function	Address	Setting	
PDR9	P93	Port data register 9 (port data register 93) When P93 = 0, the output level of the P93 pin is low. When P93 = 1, the output level of the P93 pin is high.	0xFFDC Bit 3	0
	P92	Port data register 9 (port data register 92) When P92 = 0, the output level of the P92 pin is low. When P92 = 1, the output level of the P92 pin is high.	0xFFDC Bit 2	1
PMR2	IRQ0	Port mode register 2 (P43/ $\overline{\text{IRQ0}}$ pin function switch) When IRQ0 = 0, the pin functions as a P43 input/output pin. When IRQ0 = 1, the pin functions as an $\overline{\text{IRQ0}}$ input pin.	0xFFE0 Bit 0	1
IEGR	IEG0	IRQ edge select register (IRQ0 edge selection) When IEG0 = 0, detection of the falling edge of the $\overline{\text{IRQ0}}$ pin input is selected. When IEG0 = 1, detection of the rising edge of the $\overline{\text{IRQ0}}$ pin input is selected.	0xFFFF2 Bit 0	0
IENR1	IEN0	Interrupt enable register 1 (IRQ0 interrupt enable) When IEN0 = 0, a $\overline{\text{IRQ0}}$ pin interrupt request is disabled. When IEN0 = 1, a $\overline{\text{IRQ0}}$ pin interrupt request is enabled.	0xFFFF3 Bit 0	1
IRR1	IRRI0	Interrupt request register 1 (IRQ0 interrupt request flag) When IRRI0 = 0, an $\overline{\text{IRQ0}}$ interrupt has not been requested. When IRRI0 = 1, an $\overline{\text{IRQ0}}$ interrupt has been requested.	0xFFFF6 Bit 0	0

G. Flowchart









2. `com_init()` function

A. Specifications

`void com_init(void)`

B. Operation

— Initializes the communication settings for 8-bit parallel communications.

C. Arguments

— Input: None

— Output: None

D. Global variables

None

E. Subroutines used

None

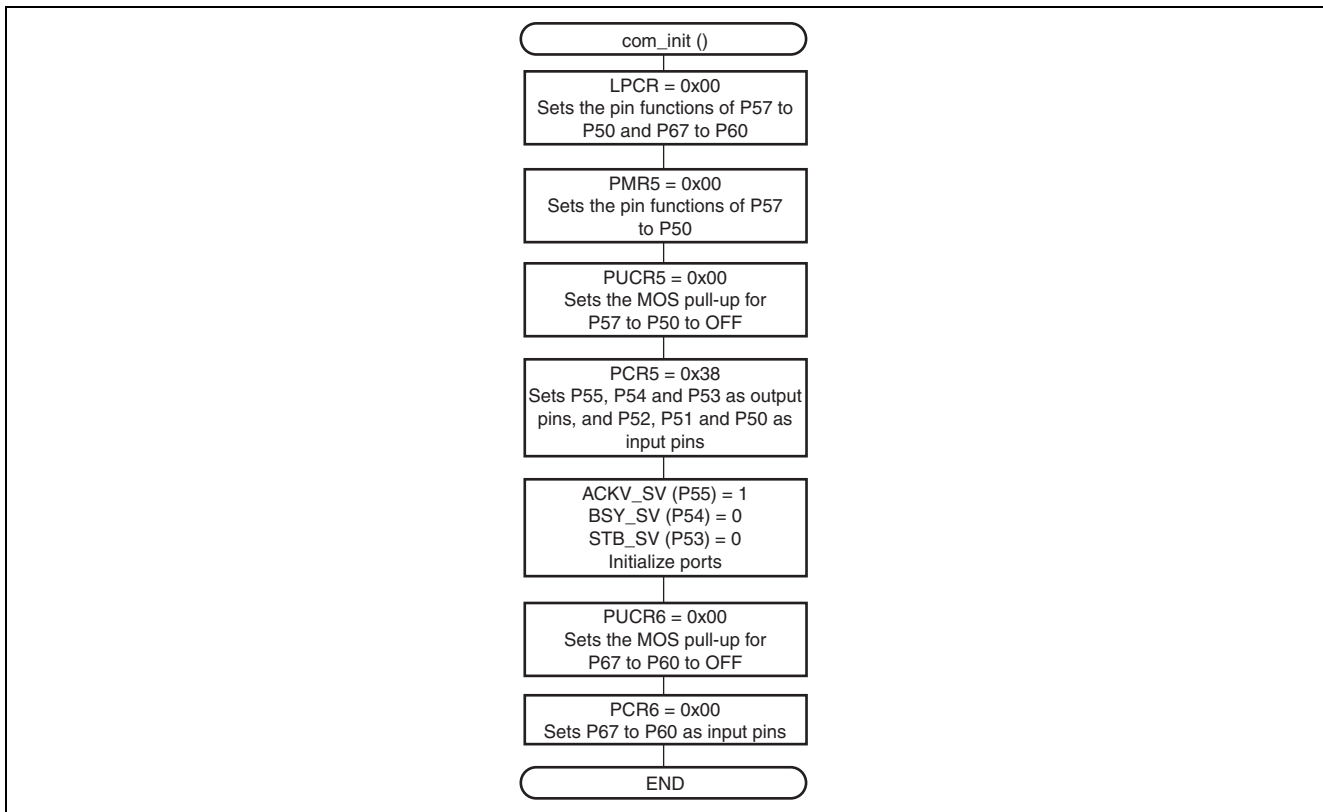
F. Internal registers used

Table7.4 Registers Used by `com_init()` Function

Register	Function	Address	Setting
LPCR	SGS3 LCD port control register (segment driver selection)	0xFFC0	SGS3 = 0
	SGS2 When SGS3 = 0, SGS2 = 0, SGS1 = 0, and SGS0 = 0, the	Bit 3	SGS2 = 0
	SGS1 SEG32 to SEG1 pins function as ports.	Bit 2	SGS1 = 0
	SGS0	Bit 1	SGS0 = 0
		Bit 0	
PMR5	Port mode register 5 When PMR5 = 0x00, the $\overline{WKP7}$ to $\overline{WKP0}$ input pins are set as P57 to P50 pins.	0xFFCC	0x00
PDR5	P52 Port data register 5 (port data register 52) A port used as ACK_MT: When ACK_MT (P52) = 0, the master can receive data. When ACK_MT (P52) = 1, the master cannot receive data.	0xFFD8 Bit 2	1
	P51 Port data register 5 (port data register 51) A port used as BSY_MT: When BSY_MT (P51) = 0, internal processing on the master side ended. When BSY_MT (P51) = 1, internal processing on the master side is in progress.	0xFFD8 Bit 1	0
	P50 Port data register 5 (port data register 50) A port used as STB_MT: When STB_MT (P50) = 0, the master is not transmitting data. When STB_MT (P50) = 1, the master is transmitting data.	0xFFD8 Bit 0	0
PUCR5	Port pull-up control register 5 When PUCR5 = 0x00, the MOS pull-up is turned off for P57 to P50.	0xFFE2	0x00
PUCR6	Port pull-up control register 6 When PUCR6 = 0x00, the MOS pull-up is turned off for P67 to P60.	0xFFE3	0x00

Register	Function	Address	Setting
PCR5	Port control register 5 When PCR5 = 0x07, P55, P54, and P53 are set as input pins and P52, P51, and P50 are set as output pins.	0xFFE8	0x07
PCR6	Port control register 6 When PCR6 = 0x00, P67 to P60 are set as input pins.	0xFFE9	0x00

G. Flowchart

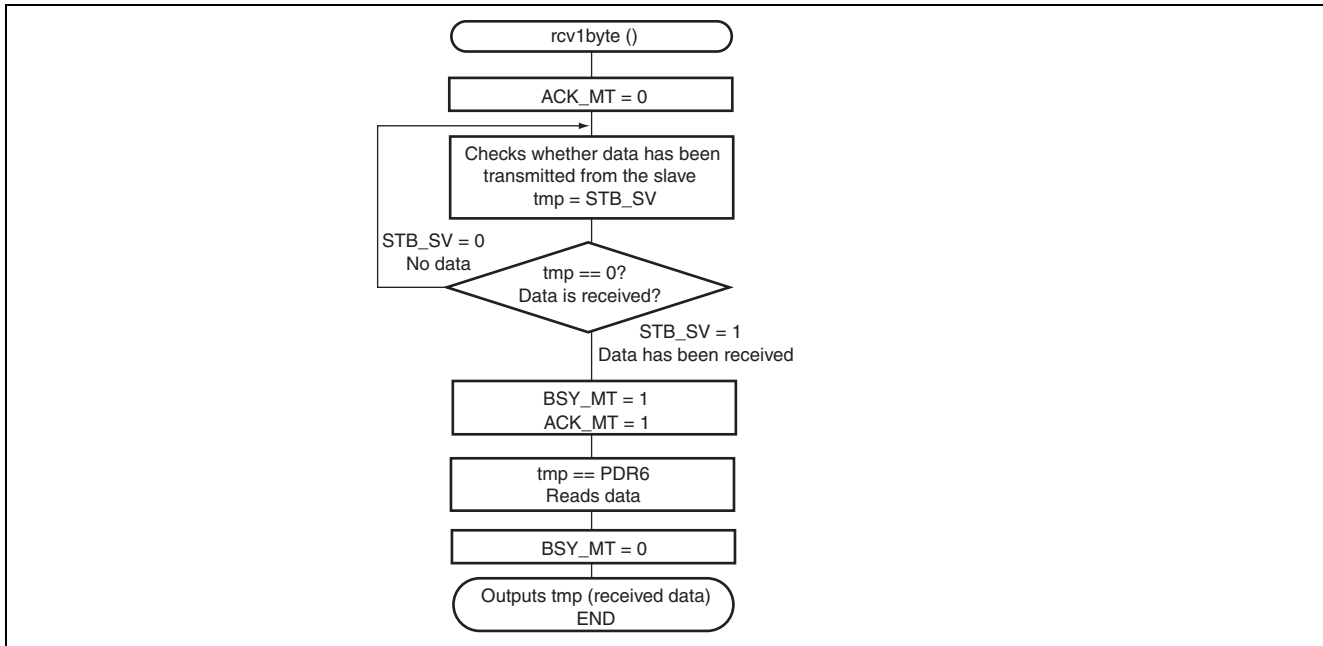


3. rcv1byte() function
 - A. Specifications
 - unsigned char rcv1byte(void)
 - B. Operation
 - Receives one byte of 8-bit parallel data.
 - C. Arguments
 - Input: None
 - Output: One byte of received data
 - D. Global variables
 - None
 - E. Subroutines used
 - None
 - F. Internal registers used

Table7.5 Registers Used by rcv1byte() Function

Register	Function	Address	Setting
PDR5 P53	Port data register 5 (port data register 53) A port used as STB_SV: When STB_SV (P53) = 0, the slave is not transmitting data. When STB_SV (P53) = 1, the slave is transmitting data.	0xFFD8 Bit 3	—
P52	Port data register 5 (port data register 52) A port used as ACK_MT: When ACK_MT (P52) = 0, the master can receive data. When ACK_MT (P52) = 1, the master cannot receive data.	0xFFD8 Bit 2	0
P51	Port data register 5 (port data register 51) A port used as BSY_MT: When BSY_MT (P51) = 0, internal processing on the master side ended. When BSY_MT (P51) = 1, internal processing on the master side is in progress.	0xFFD8 Bit 1	1
PDR6	Port data register 6 An 8-bit register that stores received data.	0xFFD9	—

G. Flowchart

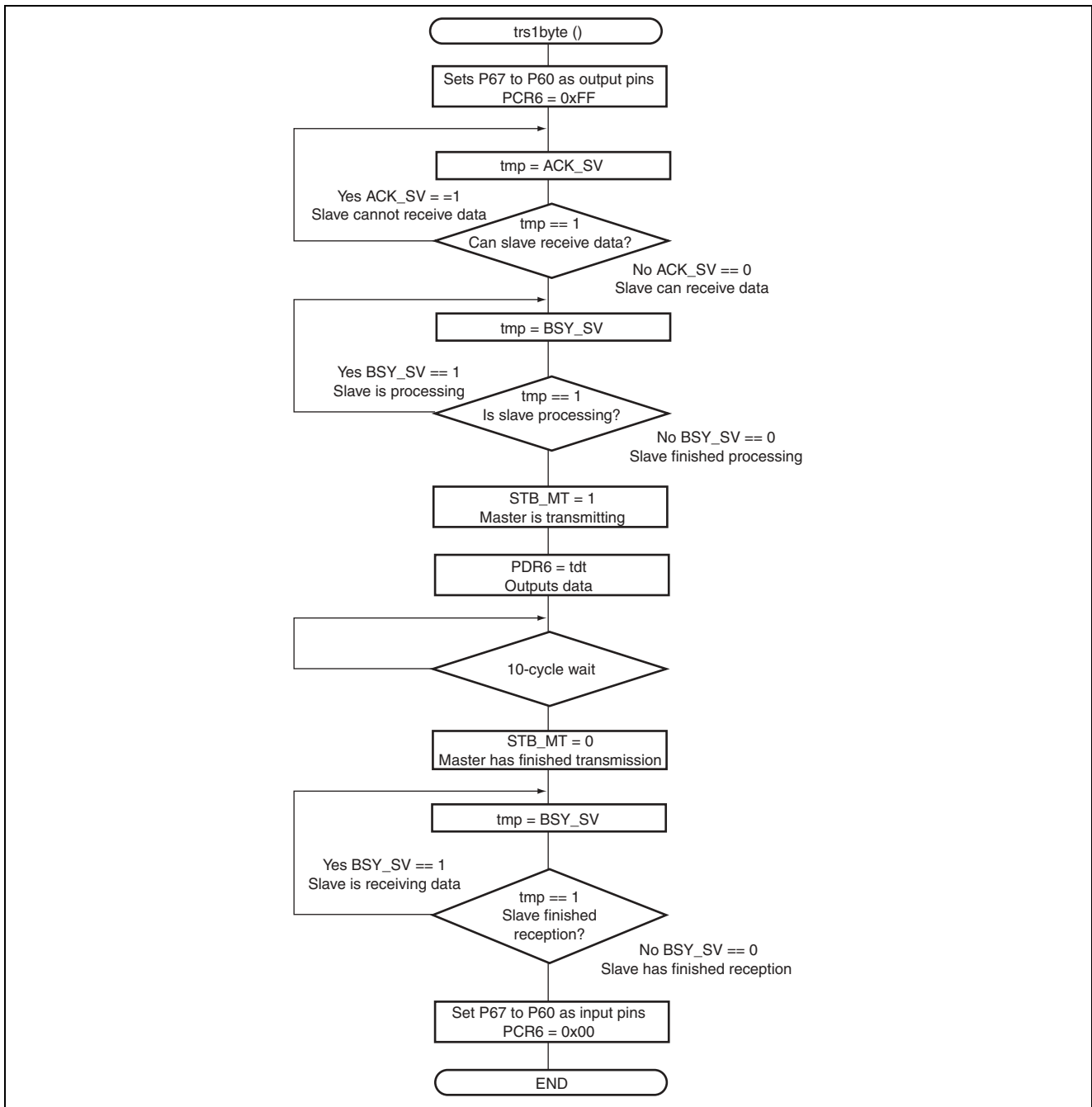


4. trs1byte() function
 - A. Specifications
 - void trs1byte(unsigned char tdt)
 - B. Operation
 - Transmits one byte of 8-bit parallel data.
 - C. Arguments
 - Input:
 - tdt: One byte of transmit data
 - Output: None
 - D. Global variables
 - None
 - E. Subroutines used
 - None
 - F. Internal registers used

Table7.6 Registers Used by trs1byte() Function

Register	Function	Address	Setting
PDR5	P55 Port data register 5 (port data register 55) A port used as ACK_SV: When ACK_SV (P55) = 0, the slave can receive data. When ACK_SV (P55) = 1, the slave cannot receive data.	0xFFD8 Bit 5	—
	P54 Port data register 5 (port data register 54) A port used as BSY_SV: When BSY_SV (P54) = 0, internal processing on the slave side ended. When BSY_SV (P54) = 1, internal processing on the slave side is in progress.	0xFFD8 Bit 4	—
	P50 Port data register 5 (port data register 50) A port used as STB_MT: When STB_MT (P50) = 0, the master is not transmitting data. When STB_MT (P50) = 1, the master is transmitting data.	0xFFD8 Bit 0	1
PDR6	Port data register 6 An 8-bit register that stores transmit data.	0xFFD9	—
PCR6	Port control register 6 When PCR6 = 0xFF, P67 to P60 are set as output pins.	0xFFE9	0xFF

G. Flowchart



5. trsnbyte() function

A. Specifications

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

B. Operation

— Transmits n bytes of 8-bit parallel data.

C. Arguments

— Input:

dtno: Number of transmit bytes

*tdt: RAM start address to store transmit data

— Output: None

D. Global variables

None

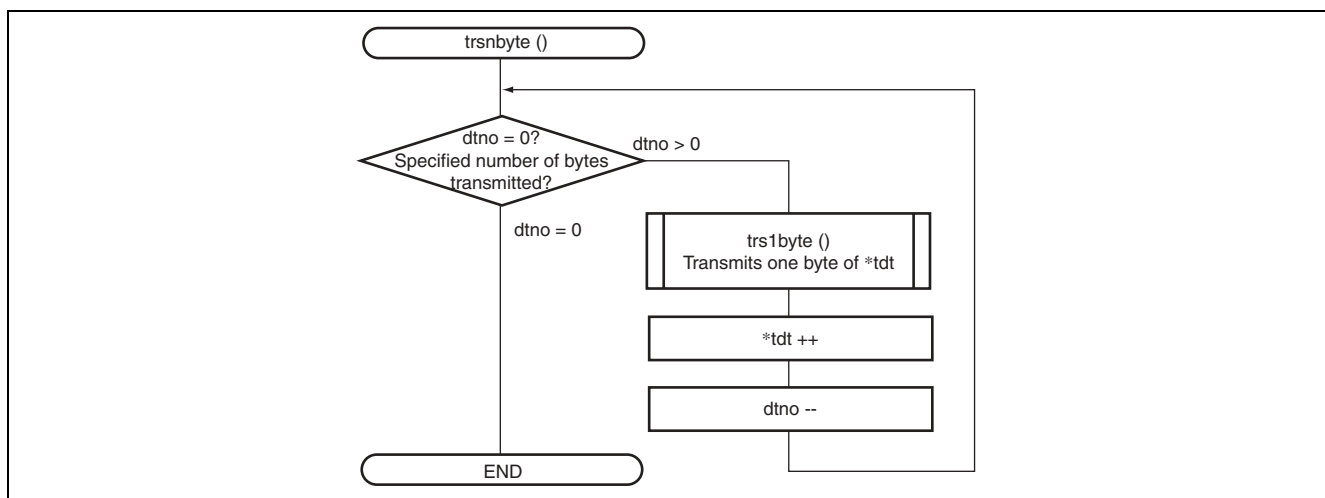
E. Subroutines used

trs1byte(): Transmits one byte of data.

F. Internal registers used

None

G. Flowchart



6. irq0int() function

A. Specifications

void irq0int(void)

B. Operation

— IRQ0 interrupt processing routine that sets a flag for branching to the processing for transmitting the program start command.

C. Arguments

— Input: None

— Output: None

D. Global variables

ramf:

ramf is cleared to 0 so that a branching to the program start command transmitting processing will take place after returning from this interrupt processing routine.

E. Subroutines used

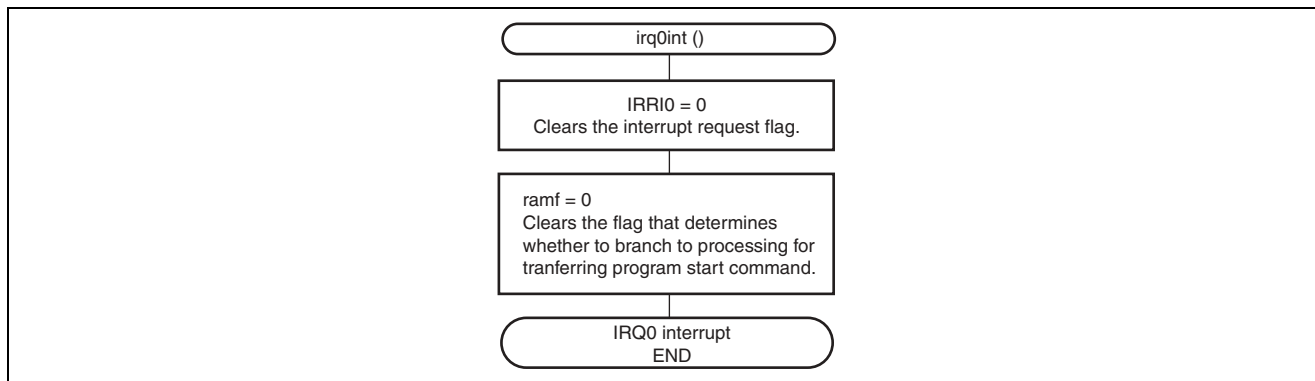
None

F. Internal register used

Table7.7 Register Used by irq0int() Function

Register	Function	Address	Setting
IRR1	IRR10	0xFFFF6	0
	Interrupt request register 1 (IRQ0 interrupt request flag)	Bit 0	
	When IRR10 = 0, an IRQ0 interrupt has not been requested.		
	When IRR10 = 1, an IRQ0 interrupt has been requested.		

G. Flowchart



8. Program Listing

8.1 Stack-Pointer Setting Program

INIT.SRC (same for the master and slave sides)

```

.EXPORT  _INIT
.IMPORT  _main
;
.SECTION P, CODE
_INIT:
MOV.W    #H'FF80,R7
LDC.B    #B'10000000,CCR
JMP      @_main
;
.END

```

8.2 Normal Program on Slave Side

```

/*****
*/
/* H8/300L Super Low Power Series
*/
/* -H8/38024 Series-
*/
/* Flash Memory Write/Erase Application Note
*/
/*
*/
/* Communication Interface
*/
/* : 8bit Parallel Interface
*/
/* Function
*/
/* : Slave Main Program
*/
/*
*/
/* External Clock : 10MHz
*/
/* Internal Clock : 5MHz
*/
/* 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 LPCR *(volatile unsigned char *)0xFFC0 /* LCD Port Control Register */
#define LCR *(volatile unsigned char *)0xFFC1 /* LCD Control Register */
#define LCR2 *(volatile unsigned char *)0xFFC2 /* LCD Control Register 2 */
#define LCDRAM (volatile unsigned char *)0xF740 /* LCD RAM */
#define PDR3_BIT (*(struct BIT *)0xFFD6) /* Port Data Register 3 */
#define P37 PDR3_BIT.b7 /* Port 37 */
#define PMR5 *(volatile unsigned char *)0xFFC5 /* Port Mode Register 5 */

```

```

#define PDR5      *(volatile unsigned char *)0xFFD8      /* Port Data Register 5      */
#define PDR5_BIT  (*(struct BIT *)0xFFD8)              /* Port Data Register 5      */
#define P55      PDR5_BIT.b5                          /* Port 55                    */
#define P54      PDR5_BIT.b4                          /* Port 54                    */
#define P53      PDR5_BIT.b3                          /* Port 53                    */
#define P52      PDR5_BIT.b2                          /* Port 52                    */
#define P51      PDR5_BIT.b1                          /* Port 51                    */
#define P50      PDR5_BIT.b0                          /* Port 50                    */
#define PDR6      *(volatile unsigned char *)0xFFD9      /* Port Data Register 6      */
#define PDR9_BIT  (*(struct BIT *)0xFFDC)              /* Port Data Register 9      */
#define P93      PDR9_BIT.b3                          /* Port 93                    */
#define P92      PDR9_BIT.b2                          /* Port 92                    */
#define PUCR5     *(volatile unsigned char *)0xFFE2      /* Port Pull-Up Control Register 5 */
#define PUCR6     *(volatile unsigned char *)0xFFE3      /* Port Pull-Up Control Register 6 */
#define PCR5      *(volatile unsigned char *)0xFFE8      /* Port Control Register 5   */
#define PCR6      *(volatile unsigned char *)0xFFE9      /* Port Control Register 6   */

#define ACK_SV    P55
#define BSY_SV    P54
#define STB_SV    P53
#define ACK_MT    P52
#define BSY_MT    P51
#define STB_MT    P50

/*****
/* Function define
/*****
extern void INIT(void);          /* SP Set
extern void FZMAIN(void);
void main(void);
unsigned char SLrcvlbyte(void);
void com_init(void);
extern unsigned char LCDDT1[6]; /* 0x0400 to 0x0405 Sample Data
extern unsigned char LCDDT2[6]; /* 0x0800 to 0x0805 Sample Data
extern unsigned char LCDDT3[6]; /* 0x0FAA to 0x0FFF Sample Data

/*****
/* Vector Address
/*****
#pragma section V1              /* Vector Section Set
void (*const VEC_TBL1[])(void) = {
    INIT                        /* 0x0000 Reset Vector
};

#pragma section                /* P

```

```

/*****
/* Main Program
*****/
void main(void)
{
    unsigned char i,tmp,tmp2;
    unsigned char *lcdram,sw16cnt;
    char *X_BGN;
    char *X_END;
    char *Y_BGN;

    LPCR = 0xC9; /* 1/4 Duty / SEG32 to SEG5 ON */
    LCR = 0xFE; /* LCD ON */
    LCR2 = 0x60;

    lcdram = LCDRAM;
    for(i = 0; i < 0x0F; i++){
        lcdram[i] = 0;
    }

    sw16cnt = 1; /* User Application Program Sample */
    do{
        PMR5 = 0x00; /* Port5 set */
        PUCR5 = 0x00; /* Port Pull-Up Control Register5 OFF */
        PCR5 = 0x38; /* P55-P53 Output/ P52 to P50 Input */
        LPCR = 0xC9; /* 1/4 Duty / SEG32 to SEG5 ON */

        do{
            if(sw16cnt == 1){
                for(i = 2; i < 6; i++){
                    lcdram[i] = LCDDT1[i];
                }
            }
            else if(sw16cnt == 2){
                for(i = 2; i < 6; i++){
                    lcdram[i] = LCDDT2[i];
                }
            }
            else if(sw16cnt == 3){
                for(i = 2; i < 6; i++){
                    lcdram[i] = LCDDT3[i];
                }
            }
            else{
                for(i = 2; i < 6; i++){
                    lcdram[i] = 0;
                }
            }

            sw16cnt++;
            if(sw16cnt > 3){
                sw16cnt = 1;
            }
        }
    }
}

```

```

do{
    tmp = P37;
    tmp2 = STB_MT;
    tmp = tmp&(~tmp2);
}while(tmp);
}while(tmp2 == 0);                                /* Data Receive? */

com_init();                                       /* Communication Initialize */
tmp = SLrcvlbyte();
}while(tmp != 0x55);                              /* Flash Memory Erase/Write Start? */

for(i = 0; i < 6; i++){
    lcdram[i] = 0;
}

/*----- Flash Memory Write Mode -----*/
P92 = 1;                                         /* LED1 OFF */
P93 = 0;                                         /* LED2 ON */

X_BGN = (char *)__sectop("FZTAT");              /* Flash , Ram Address Copy */
X_END = (char *)__secend("FZEND");
Y_BGN = (char *)__sectop("RAM");

memcpy(Y_BGN,X_BGN,X_END-X_BGN);               /* Flash -> RAM Copy */

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

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

    ACK_SV = 0;                                  /* Slave Acknowledge OFF */
    do{
        tmp = STB_MT;
    }while(tmp == 0);                            /* Master Strobe ON? */

    BSY_SV = 1;                                  /* Slave Busy ON */
    ACK_SV = 1;                                  /* Slave Acknowledge ON */
    tmp = PDR6;                                  /* Receive Data (1 byte) */
    BSY_SV = 0;                                  /* Slave Busy OFF */

    return (tmp);
}

```

```

/*****
/* Communication Initialize
*****/
void com_init(void)
{
    LPCR = 0x00;          /* Port6,Port5 set          */

    PMR5 = 0x00;        /* Port5 set                */
    PUCR5 = 0x00;       /* Port Pull-Up Control Register5 OFF */
    PCR5 = 0x38;       /* P55 to P53 Output/ P52 to P50 Input */
    STB_SV = 0;        /* Slave Strobe OFF        */
    BSY_SV = 0;        /* Slave Busy OFF          */
    ACK_SV = 1;        /* Slave Acknowledge ON    */

    PUCR6 = 0x00;       /* Port Pull-Up Control Register6 OFF */
    PCR6 = 0x00;       /* P67 to P60 Input Set    */
}

```

Link address specifications

Section Name	Address
CV1	0x0000
P	0x0100
DLCDDT1	0x0400
DLCDDT2	0x0800
DLCDDT3	0x0FFA
FZTAT, PFZTAT, DFZTAT, FZEND	0x1000
RAM, PRAM, DRAM, B	0xF780

8.3 Program/Erase Control Program on Slave Side

```

/*****
/*
/* H8/300L Super Low Power Series
/* -H8/38024 Series-
/* Flash Memory Write/Erase Application Note
/*
/* Communication Interface
/* : 8bit Parallel Interface
/* Function
/* : Slave Flash Memory Write/Erase Control Program
/*
/* External Clock : 10MHz
/* Internal Clock : 5MHz
/* Sub Clock : 32.768kHz
/*
/*****
#pragma section FZTAT

#include <machine.h>
#include "string.h"

/***** WAIT TIME *****/
#define MHZ 5 /* 5MHZ (10MHz/2)
#define KEISU 8 /* 1Loop 8Step <-- DEC.B(2)+MOV.B(2)+BNE4.

#define WLOOP1 1*MHZ/KEISU+1 /* LOOP WAIT TIME
#define WLOOP2 2*MHZ/KEISU+1
#define WLOOP4 4*MHZ/KEISU+1
#define WLOOP5 5*MHZ/KEISU+1
#define WLOOP10 10*MHZ/KEISU+1
#define WLOOP20 20*MHZ/KEISU+1
#define WLOOP50 50*MHZ/KEISU+1
#define WLOOP100 100*MHZ/KEISU+1
#define TIME10 10*MHZ/KEISU /* WRITE WAIT TIME
#define TIME30 30*MHZ/KEISU /* WRITE WAIT TIME
#define TIME200 200*MHZ/KEISU /* WRITE WAIT TIME
#define TIME10000 10000*MHZ/KEISU /* WRITE WAIT TIME

#define MAXBLK1 10
#define OK 0
#define NG 1
#define WNG 2

/*****
/* 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 *)0xF020      /* Flash Memory Control Register 1      */
#define FLMCR1_BIT  (*(struct BIT *)0xF020)              /* Flash Memory Control Register 1      */
#define SWE        FLMCR1_BIT.b6                        /* Software Write Enable                */
#define ESU        FLMCR1_BIT.b5                        /* Erase Setup                          */
#define PSU        FLMCR1_BIT.b4                        /* Program Setup                        */
#define EV         FLMCR1_BIT.b3                        /* Erase Verify                         */
#define PV         FLMCR1_BIT.b2                        /* Program Verify                       */
#define ELS        FLMCR1_BIT.b1                        /* Erase                                */
#define PGM        FLMCR1_BIT.b0                        /* Program                              */
#define EBR        *(volatile unsigned char *)0xF023      /* Erase Block Register                 */
#define FENR       *(volatile unsigned char *)0xF02B      /* Flash Memory Enable Register         */
#define FENR_BIT   (*(struct BIT *)0xF02B)              /* Flash Memory Enable Register         */
#define FLSHE      FENR_BIT.b7                         /* Flash Memory Control Register Enable */
#define TCSRW      *(volatile unsigned char *)0xFFB2      /* Timer Control/Status Register W     */
#define TCSRW_BIT  (*(struct BIT *)0xFFB2)              /* Timer Control/Status Register W     */
#define B6WI       TCSRW_BIT.b7                         /* Bit-6 Write Disable                 */
#define TCWE       TCSRW_BIT.b6                        /* Timer Counter W Write Enable        */
#define B4WI       TCSRW_BIT.b5                        /* Bit-4 Write Disable                 */
#define TCSRWE     TCSRW_BIT.b4                        /* Timer Control/Status Register W     */
/*                                     Write Enable */
#define B2WI       TCSRW_BIT.b3                        /* Bit-2 Write Disable                 */
#define WDON       TCSRW_BIT.b2                        /* Watchdog Timer ON                   */
#define B0WI       TCSRW_BIT.b1                        /* Bit-0 Write Disable                 */
#define WRST       TCSRW_BIT.b0                        /* Watchdog Timer Reset                 */
#define TCW        *(volatile unsigned char *)0xFFB3      /* Timer Counter W                     */
#define PMR2       *(volatile unsigned char *)0xFFC9      /* Port Mode Register 2                */
#define PMR2_BIT   (*(struct BIT *)0xFFC9)              /* Port Mode Register 2                */
#define WDCKS      PMR2_BIT.b2                         /* Watchdog Timer Source Clock         */
#define PMR5       *(volatile unsigned char *)0xFFC8      /* Port Mode Register 5                */
#define PDR5       *(volatile unsigned char *)0xFFD8      /* Port Data Register 5                */
#define PDR5_BIT   (*(struct BIT *)0xFFD8)              /* Port Data Register 5                */
#define P55        PDR5_BIT.b5                        /* Port 55                             */
#define P54        PDR5_BIT.b4                        /* Port 54                             */
#define P53        PDR5_BIT.b3                        /* Port 53                             */
#define P52        PDR5_BIT.b2                        /* Port 52                             */
#define P51        PDR5_BIT.b1                        /* Port 51                             */
#define P50        PDR5_BIT.b0                        /* Port 50                             */
#define PDR6       *(volatile unsigned char *)0xFFD9      /* Port Data Register 6                */
#define PDR9_BIT   (*(struct BIT *)0xFFDC)              /* Port Data Register 9                */
#define P93        PDR9_BIT.b3                        /* Port 93                             */
#define P92        PDR9_BIT.b2                        /* Port 92                             */
#define PUCR5      *(volatile unsigned char *)0xFFE2      /* Port Pull-Up Control Register 5     */
#define PUCR6      *(volatile unsigned char *)0xFFE3      /* Port Pull-Up Control Register 6     */
#define PCR5       *(volatile unsigned char *)0xFFE8      /* Port Control Register 5             */
#define PCR6       *(volatile unsigned char *)0xFFE9      /* Port Control Register 6             */

#define ACK_SV     P55
#define BSY_SV     P54
#define STB_SV     P53
#define ACK_MT     P52
#define BSY_MT     P51
#define STB_MT     P50

```

```

/*****
/* Function define
/*****
void FZMAIN(void);
unsigned char rcv1byte(void);
void rcvnbyte(unsigned char dtno,unsigned char *ram);
void trslbyte(unsigned char tdt);
void fwrite(unsigned char *buf,unsigned char *w_adr,unsigned char ptime);
char fwritevf(unsigned char *owbuff,unsigned char *buff,unsigned char *w_adr,unsigned char *w_buf);
char fwritel28(unsigned char *BUFF,unsigned char *OWBUFF,unsigned char *w_adr, unsigned char *w_buf,
unsigned short WT_COUNT);
char blk_check(unsigned short ersad,unsigned short *evf_st, unsigned short *evf_ed, unsigned char *blk_no);
void ferase(unsigned char blk_no);
char ferasevf(unsigned short evf_st, unsigned short evf_ed);
char blk1_erase(unsigned short evf_st, unsigned short evf_ed, unsigned char blk_no, unsigned char ET_COUNT);

unsigned short BLOCKADR1[10] = {
0x0000,0x03ff,
0x0400,0x07ff,
0x0800,0x0bff,
0x0c00,0x0fff,
0x1000,0x7fff,
};

#define WDT_ERASE 0x00
#define WDT_WRITE 0xFB
#define OW_COUNT 6

/*****
/* Flash Memory Write Main Program
/*****
void FZMAIN(void)
{
char tmp1,rtn;
unsigned char rcvldt;
unsigned short ad_tmp;
unsigned short E_ADR[10];
unsigned short restsize;
unsigned char tmp0[10];
unsigned char blk_no,i;
unsigned short evf_st,evf_ed;
unsigned char BUFF[128];
unsigned char W_BUF[128];
unsigned char OWBUFF[128];

trslbyte(OK);
/*----- Erase -----
rcvnbyte(2,tmp0);
if(tmp0[0] != 0x77)
goto ERRCASE;

trslbyte(OK);

tmp1 = tmp0[1] << 1;
rcvnbyte(tmp1, (unsigned char*)E_ADR);

FLSHE = 1;
for(i = 0; i < tmp0[1]; i++){
rtn = blk_check(E_ADR[i],&evf_st,&evf_ed,&blk_no);

```

```

    if(rtn != OK)
        goto ERRCASE;

    rtn = blk1_erase(evf_st, evf_ed, blk_no, 3);          /* 1 block Erase          */
    if(rtn != OK)
        goto ERRCASE;
}

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

/*----- Write Address / Size Receive -----*/

    rcvldt = rcvbyte();                                 /* Receive 1 byte Data -> RAM Area */
    if(rcvldt != 0x88)
        goto ERRCASE;

    for(i = 0; i < 10; i++){
        trslbyte(OK);                                   /* SEND OF OK Code          */

        rcvnbyte(4, tmp0);                              /* Receive Write Top Address & Size */

        if(tmp0[1] & 0x7F)
            goto ERRCASE;

        ad_tmp = tmp0[0];                                /* Address Copy            */
        ad_tmp <<= 8;
        ad_tmp = ad_tmp | tmp0[1];

        restsize = tmp0[2];                              /* Size Copy                */
        restsize <<= 8;
        restsize = restsize | tmp0[3];
        if(restsize == 0x0000){
            goto ERRCASE;
        }

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

/*----- 128 byte Flash Memory Write -----*/

        while(restsize != 0){
            trslbyte(0x11);                              /* SEND OF Request          */

            if(restsize <= 128){                          /* Receive Write Data from HOST */
                memset(W_BUF, 0xFF, 128);                /* INITIALIZE RECEIVE BUFFER WITH 0xFF */
                rcvnbyte((unsigned char)restsize, W_BUF); /* "rsize" byte Receive      */
                restsize = 0;
            }
            else{
                rcvnbyte(128, W_BUF);                    /* 128 bytes Receive        */
                restsize -= 128;
            }

            rtn = fwrite128(BUFF, OWBUFF, (unsigned char*)ad_tmp, W_BUF, 1000);
            if(rtn != OK)
                goto ERRCASE;

            ad_tmp = ad_tmp + 128;
        }
}

```

```

    trslbyte(OK);                /* SEND OF OK Code                */
    P92 = 0;                      /* LED1 ON                          */
    P93 = 1;                      /* LED2 OFF                          */

    PMR2 &= 0xFB;                /* PMR2 WDCKS = 0 phi/8192          */
    TCSRW = 0x50;                /* WDT STOP, TCW ENABLE              */
    TCW = 0xFF;                  /* INITIALIZED WDT COUNT             */
    TCSRW = 0x56;                /* WDT START                          */

    while(1);                    /* OK End                            */

/*----- Error Case -----*/
ERRCASE:                          /* Error Case                        */
    trslbyte(NG);
    P92 = 0;                      /* LED1 ON                          */
    P93 = 0;                      /* LED2 ON                          */
    while(1);
}

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

    ACK_SV = 0;                  /* Slave Acknowledge OFF            */

    do{
        tmp = STB_MT;
    }while(tmp == 0);           /* Master Strobe ON?                */

    BSY_SV = 1;                  /* Slave Busy ON                    */
    ACK_SV = 1;                  /* Slave Acknowledge ON              */
    tmp = PDR6;                  /* Receive Data (1 byte)            */
    BSY_SV = 0;                  /* Slave Busy OFF                    */

    return (tmp);
}

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

```

```

/*****
/* Transmit 1 byte
/*****
void trslbyte (unsigned char tdt)

{
    unsigned char tmp,i;

    PCR6 = 0xFF; /* P67 to P60 Output Set */
    do{
        tmp = ACK_MT;
    }while(tmp == 1); /* Master Acknowledge ON ? */

    do{
        tmp = BSY_MT;
    }while(tmp == 1); /* Master Busy ON ? */

    STB_SV = 1; /* Slave Strobe ON */
    PDR6L = b4dt; /* Transmit Data (1bit) */
    for(i = 0; i < 10; i++);
    STB_SV = 0; /* Slave Strobe OFF */

    do{
        tmp = BSY_MT;
    }while(tmp == 1); /* Master Busy ON ? */

    PCR6 = 0x00; /* P67 to P60 Input Set */
}

/*****
/* Erase Block Check Routine
/*****
char blk_check(unsigned short ersad,unsigned short *evf_st, unsigned short *evf_ed, unsigned char *blk_no)
{
    unsigned char i;

    for(i = 0; ersad != BLOCKADR1[i]; i += 2){ /* COMPARE BLOCK_START_ADDRESS */
        if(MAXBLK1 < i) /* BLOCK NUMBER MAX? */
            return (NG); /* ERASE BLOCK ADDRESS ERROR */
    }

    *blk_no = i>>1; /* ERASE BLOCK NUMBER */
    *evf_st = BLOCKADR1[i]; /* ERASE START ADDRESS */
    i++;
    *evf_ed = BLOCKADR1[i]; /* ERASE END ADDRESS */

    return (OK);
}

```

```

/*****
/* Flash Memory 1 block Erase
/*****
char blk1_erase(unsigned short evf_st, unsigned short evf_ed, unsigned char blk_no, unsigned char ET_COUNT)
{
    unsigned char i;
    char rtn;

    SWE = 1; /* Set the SWE bit */
    for(i = 0; i < WLOOP1; i++); /* Need to wait lusec */

    rtn = ferasevf(evf_st, evf_ed); /* Erase Verify */

    for(i = 0; i < ET_COUNT; i++){ /* Count Check (Max Erase count) */
        if(!rtn)
            break;
        ferase(blk_no); /* Erase */
        rtn = ferasevf(evf_st, evf_ed); /* Erase Verify */
    }

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

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

    tmp = 1;
    tmp <<= blk_no;
    EBR = tmp; /* Set the EBR Erase Block bit */

    PMR2 &= 0xFB; /* PMR2 WDCKS = 0 phi/8192 */
    TCSRW = 0x50; /* WDT STOP, TCW ENABLE */
    TCW = WDT_ERASE; /* INITIALIZED WDT COUNT */
    TCSRW = 0x56; /* WDT START */

    ESU = 1; /* Set the ESU bit */
    for(i = 0; i < WLOOP100; i++); /* Need to wait 100 usec */

    ELS = 1; /* Set the E bit (ERASE) */
    for(j = 0; j < TIME10000; j++); /* Need to wait 10 msec */

    ELS = 0; /* Clear the E bit */
    for(i = 0; i < WLOOP10; i++); /* Need to wait 10 usec */

    ESU = 0; /* Clear the ESU bit */
    for(i = 0; i < WLOOP10; i++); /* Need to wait 10 usec */

    TCSRW = 0x52; /* WDT STOP */

    EBR = 0;
}

```

```

/*****
/* Erase Verify
*****/
char   ferasevf(unsigned short evf_st, unsigned short evf_ed)
{
    unsigned short *tmp;
    unsigned char  i;

    EV = 1;                               /* Set the EV bit                */
    for(i = 0; i < WLOOP20; i++);         /* Need to wait 20 usec         */

    for(tmp = (unsigned short*)evf_st; tmp < (unsigned short*)evf_ed; tmp = (unsigned short*)(tmp+1)){
        *tmp = 0xFFFF;                   /* Perform dummy write          */
        for(i = 0; i < WLOOP2; i++);     /* Need to wait 2 usec          */

        if(*tmp != 0xFFFF){              /* Verify                        */
            EV = 0;                       /* Clear the EV bit             */
            for(i = 0; i < WLOOP4; i++);   /* Need to wait 4 usec          */
            return(NG);                   /* NG flag set                  */
        }
    }

    EV = 0;                               /* Clear the EV bit                */
    for(i = 0; i < WLOOP4; i++);         /* Need to wait 4 usec          */

    return(OK);                           /* OK flag set                   */
}

/*****
/* Flash Memory 128 byte Write
*****/
char   fwritel28(unsigned char *BUFF, unsigned char *OWBUFF, unsigned char *w_adr, unsigned char *w_buf,
                unsigned short WT_COUNT)
{
    char  rtn;
    unsigned char  TM, i;
    unsigned short j;

    memcpy(BUFF, w_buf, 128);             /* W_BUF -> BUFF  BLOCK COPY    */

    SWE = 1;                             /* Set the SWE bit               */
    for(i = 0; i < WLOOP1; i++);         /* Need to wait 1 usec          */

    rtn = fwritevf(OWBUFF, BUFF, w_adr, w_buf); /* 1st Program Verify          */
    if(rtn == NG){                       /* 1st Verify END               */
        TM = TIME30;
        for(j = 0; j < WT_COUNT; j++){
            fwrite(BUFF, w_adr, TM);       /* Input P Pulse (30 usec)      */
            rtn = fwritevf(OWBUFF, BUFF, w_adr, w_buf);

            if(j < OW_COUNT){             /* Count Check(additional Write Count) */
                fwrite(OWBUFF, w_adr, TIME10);
            }
            else{
                TM = TIME200;             /* Input P Pulse (200 usec)     */
            }
        }
    }
}

```



```

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

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

/*****
/* Flash Memory Write */
*****/
void fwrite(unsigned char *buf,unsigned char *w_adr,unsigned char ptime)
{
    unsigned char i;

    for(i = 0; i < 128; i++){ /* 128 byte repeat */
        w_adr[i] = buf[i]; /* Rewrite data dummy write */
    }

    PMR2 &= 0xFB; /* PMR2 WDCKS = 0 phi/8192 */
    TCSRW = 0x50; /* WDT STOP, TCW ENABLE */
    TCW = WDT_WRITE; /* INITIALIZED WDT COUNT */
    TCSRW = 0x56; /* WDT START */

    PSU = 1; /* Set the PSU bit */
    for(i = 0; i < WLOOP50; i++); /* Need to wait 50 usec */

    PGM = 1; /* Set the P bit */
    for(i = 0; i < ptime; i++); /* Writing Time 10/30/200 usec */

    PGM = 0; /* Clear the P bit */
    for(i = 0; i < WLOOP5; i++); /* Need to wait 5 usec */

    PSU = 0; /* Clear the PSU bit */
    for(i = 0; i < WLOOP5; i++); /* Need to wait 5 usec */

    TCSRW = 0x52; /* WDT STOP */
}

/*****
/* Flash Memory Verify */
*****/
char fwritevf(unsigned char *owbuff,unsigned char *buff,unsigned char *w_adr,unsigned char *w_buf)
{
    unsigned char i,j;
    unsigned char tmp;
    char rtn;

    PV = 1; /* Set the PV bit */
    for(i = 0; i < WLOOP4; i++); /* Need to wait 4 usec */

    for(j = 0; j < 128; j++){
        w_adr[j] = 0xFF; /* Dummy Write */
        for(i = 0; i < WLOOP2; i++); /* Need to wait 2 usec */

        owbuff[j] = buff[j] | w_adr[j];
    }
}

```

```

tmp = ~w_adr[j];
buff[j] = tmp | w_buf[j];

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

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

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

return(rtn);
}
#pragma section FZEND

```

Link address specifications

Section Name	Address
CV1	0x0000
P	0x0100
DLCDDT1	0x0400
DLCDDT2	0x0800
DLCDDT3	0x0FFA
FZTAT, PFZTAT, DFZTAT, FZEND	0x1000
RAM, PRAM, DRAM, B	0xF780

8.4 Program on Master Side

```

/*****
/*
/* H8/300L Super Low Power Series
/* -H8/38024 Series-
/* Flash Memory Write/Erase Application Note
/*
/* Communication Interface
/* : 8bit Parallel Interface
/* Function
/* : Master Main Program
/*
/* External Clock : 10MHz
/* Internal Clock : 5MHz
/* Sub Clock : 32.768kHz
/*
/*****
#include <machine.h>
#include "string.h"

/*****
#define OK 0
#define NG 1

/*****
/* 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 LPCR *(volatile unsigned char *)0xFFC0 /* LCD Port Control Register */
#define PMR2 *(volatile unsigned char *)0xFFC9 /* Port Mode Register 2 */
#define PMR2_BIT (*(struct BIT *)0xFFC9) /* Port Mode Register 2 */
#define IRQ0 PMR2_BIT.b0 /* P43/IRQ0 Select */
#define PMR5 *(volatile unsigned char *)0xFFCC /* Port Mode Register 5 */
#define PDR5 *(volatile unsigned char *)0xFFD8 /* Port Data Register 5 */
#define PDR5_BIT (*(struct BIT *)0xFFD8) /* Port Data Register 5 */
#define P55 PDR5_BIT.b5 /* Port 55 */
#define P54 PDR5_BIT.b4 /* Port 54 */
#define P53 PDR5_BIT.b3 /* Port 53 */
#define P52 PDR5_BIT.b2 /* Port 52 */
#define P51 PDR5_BIT.b1 /* Port 51 */
#define P50 PDR5_BIT.b0 /* Port 50 */
#define PDR6 *(volatile unsigned char *)0xFFD9 /* Port Data Register 6 */
#define PDR9_BIT (*(struct BIT *)0xFFDC) /* Port Data Register 9 */
#define P93 PDR9_BIT.b3 /* Port 93 */
#define P92 PDR9_BIT.b2 /* Port 92 */
#define PUCR5 *(volatile unsigned char *)0xFFE2 /* Port Pull-Up Control Register 5 */
#define PUCR6 *(volatile unsigned char *)0xFFE3 /* Port Pull-Up Control Register 6 */
#define PCR5 *(volatile unsigned char *)0xFFE8 /* Port Control Register 5 */

```

```

#define PCR6      *(volatile unsigned char *)0xFFE9      /* Port Control Register 6      */
#define IEGR_BIT  (*(struct BIT *)0xFFF2)              /* Interrupt Edge Select Register 1 */
#define IEG0      IEGR_BIT.b0                          /* IEG0 Edge Select              */
#define IENR1_BIT (*(struct BIT *)0xFFF3)              /* Interrupt Enable Register 1    */
#define IEN0      IENR1_BIT.b0                         /* IEN0 Interrupt Enable         */
#define IRR1_BIT  (*(struct BIT *)0xFFF6)              /* Interrupt Request Register 1   */
#define IRRIO     IRR1_BIT.b0                          /* IRRIO Interrupt Request Register */

#define ACK_SV    P55
#define BSY_SV    P54
#define STB_SV    P53
#define ACK_MT    P52
#define BSY_MT    P51
#define STB_MT    P50

#pragma interrupt (irq0int)
/*****
/* Function define
*****/
extern void INIT(void); /* SP Set
void main(void);
void irq0int(void);
unsigned char rcv1byte(void);
void trslbyte(unsigned char tdt);
void trsnbyte(short dtno,unsigned char *tdt);
void com_init(void);

volatile char ramf;
extern unsigned char LCDDT1[0x0C00];
unsigned short BLOCKADR1[10] = {
    0x0000,0x03ff, /* EB0 1KBYTE
    0x0400,0x07ff, /* EB1 1KBYTE
    0x0800,0x0bff, /* EB2 1KBYTE
    0x0c00,0x0fff, /* EB3 1KBYTE
    0x1000,0x7fff, /* EB4 28KBYTES
};

/*****
/* Vector Address
*****/
#pragma section V1 /* Vector Section Set
void (*const VEC_TBL1[]) (void) = {
    INIT /* 0x0000 - 0x000F
    /* 0x0000 Reset Vector
};
#pragma section V2 /* Vector Section Set
void (*const VEC_TBL2[]) (void) = {
    irq0int /* 0x0008 IRQ0 Interrupt Vector
};

#pragma section /* P

```

```

/*****
/* Main Program
/*****
void main(void)
{
    unsigned char senddt[10],tmp;
    unsigned short wtsize;
    unsigned char i;
    unsigned short j;
    union trsbk{
        unsigned char b[10];
        unsigned short w[5];
    }bk;

    set_imask_ccr(1); /* Interrupt Disable */

    IEG0 = 0; /* Initialize IRQ0 Terminal Input Edge */
    IRRIO = 0; /* Initialize IRQ0 Interrupt Request Flag */
    IRQ0 = 1;
    IEN0 = 1; /* IRQ0 Interrupt Enable */

    ramf = 1;
    set_imask_ccr(0); /* Interrupt Enable */
    while(ramf); /* Flash Memory Write Mode Check */
    IEN0 = 0; /* IRQ0 Interrupt Enable */

/*-----*/
    P92 = 1; /* LED1 OFF */
    P93 = 0; /* LED2 ON */

    com_init();
    STB_MT = 1; /* Master Strobe ON */

/*----- Flash Erase/Write Program Start -----*/

    trslbyte(0x55); /* Start Command */

    tmp = rcvlbyte();
    if(tmp != OK)
        goto ERRCASE;

/*----- Erase -----*/

    senddt[0] = 0x77; /* Erase Command(0x77) */
    senddt[1] = 0x03; /* Erase Area Block Count */
    trsnbyte(2,senddt);

    tmp = rcvlbyte();
    if(tmp != OK) /* Error Check */
        goto ERRCASE;

    for(i = 0; i < senddt[1]; i++){ /* Erase Block No 1 & 2 & 3 */
        tmp = i+1;
        tmp <<= 1;
        bk.w[i] = BLOCKADR1[tmp];
    }
    trsnbyte(2*senddt[1],bk.b); /* Send of Erase Block */

    tmp = rcvlbyte();

```

```

if(tmp != OK)                                /* Error Check                */
    goto ERRCASE;

/*----- Send of Write Address & Size -----*/

    trslbyte(0x88);                            /* SEND OF Write Command(0x88) */

    tmp = rcvlbyte();
    if(tmp != OK)                              /* Error Check                */
        goto ERRCASE;

    bk.w[0] = 0x400;                            /* Write Address = 0x400      */
    wtsize = 0x0c00;                            /* Write Size = 0x0c00 byte   */
    bk.w[1] = wtsize;
    trsnbyte(4,bk.b);                          /* Send of Address & Size     */

    tmp = rcvlbyte();
    if(tmp != OK)                              /* Error Check                */
        goto ERRCASE;

/*----- Send of Write Data -----*/

    j = 0;
    while(wtsize != 0){
        tmp = rcvlbyte();                      /* Receive of Request        */
        if(tmp != 0x11)                        /* Error Check                */
            goto ERRCASE;

        if(wtsize <= 128){
            tmp = wtsize;
            trsnbyte(tmp,&(LCDDT1[j]));
            wtsize = 0;
        }
        else{
            trsnbyte(128,&(LCDDT1[j]));
            j += 128;
            wtsize -= 128;
        }
    }

    tmp = rcvlbyte();
    if(tmp != OK)                              /* Error Check                */
        goto ERRCASE;

    P92 = 0;                                    /* LED1 ON                    */
    P93 = 1;                                    /* LED2 OFF                   */
    while(1);                                  /* OK End                     */

/*----- Error Case -----*/
ERRCASE:                                       /* Error Case                  */
    P92 = 0;
    P93 = 0;
    while(1);
}

```

```

/*****
/* IRQ0 Interrupt */
/*****
void irq0int(void)
{
    IRRIO = 0; /* Initialize IRQ0 Interrupt Request Flag */

    ramf = 0;
}

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

    ACK_MT = 0; /*Master Acknowledge OFF */

    do{
        tmp = STB_SV;
    }while (tmp == 0); /* Slave Strobe ON? */

    BSY_MT = 1; /* Master Busy ON */
    ACK_MT = 1; /* Master Acknowledge ON */
    tmp = PDR6; /* Receive Data (1 byte) */
    BSY_MT = 0; /* Master Busy OFF */

    return (tmp);
}

dh <<= 4;
tmp = dh | dl;

return(tmp);
}

/*****
/* Transmit 1 byte */
/*****
void trslbyte(unsigned char tdt)
{
    unsigned char tmp, i;

    PCR6 = 0xFF; /* P67 to P60 Output Set */

    do{
        tmp = ACK_SV;
    }while(tmp == 1); /* Slave Acknowledge ON? */

    do{
        tmp = BSY_SV;
    }while(tmp == 1); /* Slave Busy ON? */

    STB_MT = 1; /* Master Strobe ON */
    PDR6 = tdt; /* Transmit Data (1 byte) */
    for(i = 0; i < 10; i++);
    STB_MT = 0; /* Master Strobe OFF */
}

```

```

do{
    tmp = BSY_SV;
}while(tmp == 1); /* Slave Busy ON? */

PCR6 = 0x00; /* P67 to P60 Input Set */
}

/*****
/* Transmit N byte */
/*****
void trsnbyte(short dtno,unsigned char *tdt)
{
    while(dtno--){ /* dtno = 0 ? */
        trslbyte((unsigned char)*tdt); /* tdt -> 1 byte Transmit Data */

        *tdt++;
    }
}

/*****
/* Communication Initialize */
/*****
void com_init(void)
{
    LPCR = 0x00; /* Port6,Port5 set */

    PMR5 = 0x00; /* Port5 set
    PUCR5 = 0x00; /* Port Pull-Up Control Register5 OFF
    PCR5 = 0x07; /* P55-P53 Input/ P52-P50 Output
    STB_MT = 0; /* Master Strobe OFF
    BSY_MT = 0; /* Master Busy OFF
    ACK_MT = 1; /* Master Acknowledge ON

    PUCR6 = 0x00; /* Port Pull-Up Control Register6 OFF
    PCR6 = 0x00; /* P67 to P60 Input Set
}

```

Link address specifications

Section Name	Address
CV1	0x0000
CV2	0x0008
D	0x0100
DLCDDT1	0x0400
DLCDDT2	0x0800
DLCDDT3	0x0FFA
P	0x1000
B	0xF780

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Dec.19.03	—	First edition issued

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.