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.

H8/3687

Access to the EEPROM (I^2C EEPROM) (H8/3687)

Introduction

The H8/3687 group are single-chip microcomputers based on the high-speed H8/300H CPU, and integrate all the peripheral functions necessary for system configuration. The H8/300H CPU employs an instruction set which is compatible with the H8/300 CPU.

The H8/3687 group incorporates, as peripheral functions necessary for system configuration, four types of timers, I^2C bus interface, serial communication interface, and 10-bit A/D converter. These devices can be utilized as embedded microcomputers in sophisticated control systems.

These H8/300 H Series -H8/3687- Application Notes consist of a "Basic Edition" which describes operation examples when using the on-chip peripheral functions of the H8/3687 group in isolation; they should prove useful for software and hardware design by the customer.

The operation of the programs and circuits described in these Application Notes has been verified, but in actual applications, the customer should always confirm correct operation prior to actual use.

Target Device

H8/3687

Contents

1. Overview

The I²C EEPROM is read or written to via the H8/3687 I²C interface.

2. Configuration

Figure 2.1 shows a diagram of connections between the H8/3687 and I²C EEPROM.

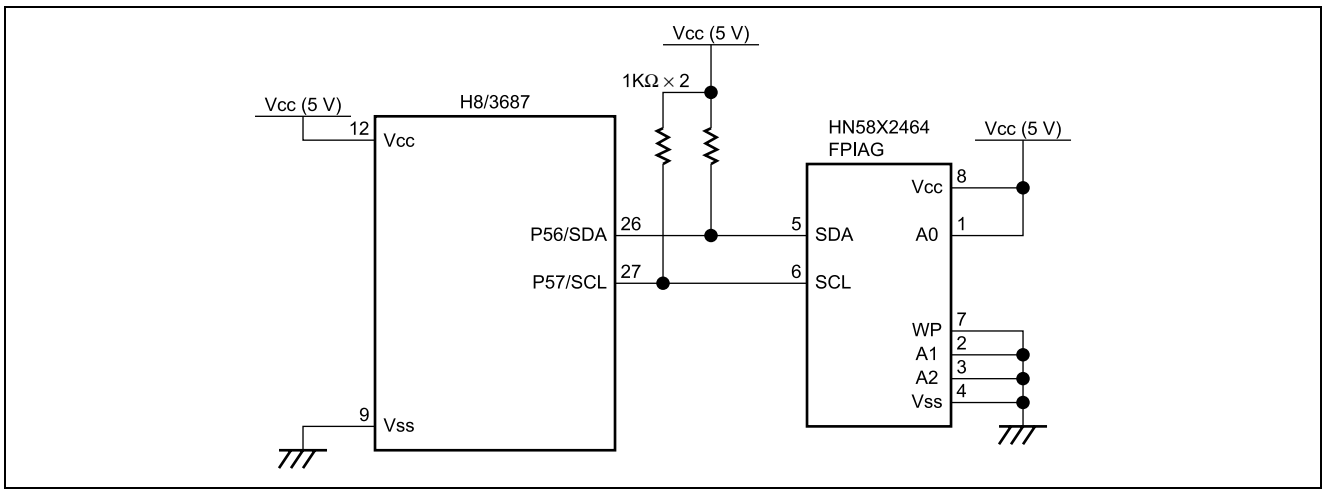


Figure 2.1 Connection to I²C EEPROM

Specifications:

- H8/3687 operating frequency: 16 MHz
- Table 2.1 shows the I²C EEPROM (HN58X2464FPIAG) pin
- I²C EEPROM specifications: 64 kbits (8192 × 8 bits)

Table 2.1 I²C EEPROM Pin Specifications

Pin Name	Function
A0 to A2	Device address (A0 is fixed to high, and A1 and A2 are fixed to low.)
SCL	Serial clock input
SDA	Serial data input/output
WP	Write protect
Vcc	Power supply
Vss	Ground

3. Sample Programs

3.1 Functions

1. One byte of data is written to the I²C EEPROM (Byte Write).
2. One byte of data is read from the I²C EEPROM (Random Read).
3. Data is written to the I²C EEPROM continuously (Page Write).
4. Data is read from the I²C EEPROM continuously (Sequential Read).

3.2 Embedding the Sample Programs

1. Sample program 1-A
   Incorporate #define directives.
2. Sample program 1-B
   Incorporate prototype declarations.
3. Sample program 1-C
   Incorporate source program.

3.3 Modifications to Sample Programs

Without modifications to the sample program, the system may not run. Modifications must be made according to the customer's program and system environment.

1. A file with definitions of IO register structures can be obtained free of charge from the following Renesas Technology web site:
   http://www.renesas.com/eng/products/mpumcu/tool/crosstool/iodef/index.html
   The sample program can be used without further changes. When creating definitions independently, the customer should modify the IO register structures used in the sample program as appropriate.
2. In the sample program, Timer Z is designed to start every 10 ms and timeout after 5 seconds, in order to monitor the state of the I²C interface. The timer processing can be modified according to the needs of the customer, and of course can be used without modification. When using the timer processing in the sample program without modification, the following changes should be made.
   A. Sample program 1-D
      - The Timer Z reset vector should be added.
      - com_timer should be added as a common variable.
      - The Timer Z initial setting processing should be added.
      - (The GRA setting should be changed according to the operating frequency of the microcomputer being used, so that the timer Z interrupt occurs in 10 ms. For setting values, refer to the H8/3687 Hardware Manual; for the location of the setting to be changed, refer to the program notes in the sample program.)
      - The timer Z interrupt processing should be added.
3. The I²C interface transfer rate ICCR1 (CKS3 to CKS0) should be set according to the target device specifications and the microcomputer operating frequency. Refer to the H8/3687 Hardware Manual for setting values, and to the program notes in the sample program for the location to be changed. In this sample program, the transfer rate is set to 200 kbps.

3.4 Method of Use

1. One byte of data is written to the I²C EEPROM.

```
unsigned int com_i2c_eeprom_write
(unsigned char slave_addr, unsigned int rom_addr, unsigned char rom_data)
```

Argument	Description
slave_addr	Specifies the I²C EEPROM slave address. The slave address is determined by the device code (fixed to 1010) and I²C EEPROM address pins (A2 to A0). In this sample program, A2 and A1 are low and A0 is high.

slave_addr format							
Device code				Address pin			
1	0	1	0	A2	A1	A0	0

Therefore, the setting value for slave_addr is 0xA2.

rom_addr	Specifies the ROM address where data is written to.
rom_data	Specifies 1-byte data to be written.

Return value	Description
0	Normal termination
1	Abnormal termination (bus busy timeout)
2	Abnormal termination (transfer-preparation completion wait timeout)
3	Abnormal termination (acknowledge timeout)
4	Abnormal termination (transfer-completion wait timeout)
5	Abnormal termination (reception-completion wait timeout)
6	Abnormal termination (stop-condition detection timeout)

Example of use:

```
int ret;
unsigned char slave_addr, rom_data;
unsigned int rom_addr;

ret = com_i2c_eeprom_write (slave_addr, rom_addr, rom_data)
```

2. One byte of data is read from the I²C EEPROM.

```
unsigned int com_i2c_eeprom_read
        (unsigned char slave_addr, unsigned int rom_addr, unsigned char *rom_data)
```

Argument	Description
slave_addr	Specifies the I²C EEPROM slave address. The slave address is determined by the device code (fixed to 1010) and I²C EEPROM address pins (A2 to A0). In this sample program, A2 and A1 are low and A0 is high.

slave_addr format							
Device code				Address pin			
1	0	1	0	A2	A1	A0	0

Therefore, the setting value for slave_addr is 0xA2.

rom_addr	Specifies the ROM address where data is read from.
*rom_data	Specifies the address where read data is stored.

Return value	Description
0	Normal termination
1	Abnormal termination (bus busy timeout)
2	Abnormal termination (transfer-preparation completion wait timeout)
3	Abnormal termination (acknowledge timeout)
4	Abnormal termination (transfer-completion wait timeout)
5	Abnormal termination (reception-completion wait timeout)
6	Abnormal termination (stop-condition detection timeout)

Example of use:

```
int ret;
unsigned char slave_addr, *rom_data;
unsigned int rom_addr;

ret = com_i2c_eeprom_read (slave_addr, rom_addr, *rom_data)
```

3. Data is continuously written to the I²C EEPROM.

```
unsigned int com_i2c_eeprom_page_write
(unsigned char slave_addr, unsigned int rom_addr, unsigned int rom_length, unsigned
char *rom_data)
```

Argument	Description
slave_addr	Specifies the I²C EEPROM slave address. The slave address is determined by the device code (fixed to 1010) and I²C EEPROM address pins (A2 to A0). In this sample program, A2 and A1 are low and A0 is high.

slave_addr format							
Device code				Address pin			
1	0	1	0	A2	A1	A0	0

Therefore, the setting value for slave_addr is 0xA2.

rom_addr	Specifies the start address of the ROM area where data is written to.
rom_length	Specifies the number of bytes to be written. On this device, up to 32 bytes of data can be written.
*rom_data	Specifies the start address of the area where write data is stored.

Return value	Description
0	Normal termination
1	Abnormal termination (bus busy timeout)
2	Abnormal termination (transfer-preparation completion wait timeout)
3	Abnormal termination (acknowledge timeout)
4	Abnormal termination (transfer-completion wait timeout)
5	Abnormal termination (reception-completion wait timeout)
6	Abnormal termination (stop-condition detection timeout)

Example of use:

```
int ret;
unsigned char slave_addr, *rom_data;
unsigned int rom_length, rom_addr;

ret = com_i2c_eeprom_page_write (slave_addr, rom_addr, rom_length, *rom_data)
```

4. Data is continuously read from the I²C EEPROM.

```
unsigned int com_i2c_eeprom_seq_read
(unsigned char slave_addr, unsigned int rom_addr,
unsigned int rom_length, unsigned char *rom_data)
```

Argument	Description
slave_addr	Specifies the I²C EEPROM slave address. The slave address is determined by the device code (fixed to 1010) and I²C EEPROM address pins (A2 to A0). In this sample program, A2 and A1 are low and A0 is high.

slave_addr format							
Device code				Address pin			
1	0	1	0	A2	A1	A0	0

Therefore, the setting value for slave_addr is 0xA2.

rom_addr	Specifies the ROM address where data is read from.
rom_length	Specifies the number of bytes to be read.
*rom_data	Specifies the address to store read data.

Return value	Description
0	Normal termination
1	Abnormal termination (bus busy timeout)
2	Abnormal termination (transfer-preparation completion wait timeout)
3	Abnormal termination (acknowledge timeout)
4	Abnormal termination (transfer-completion wait timeout)
5	Abnormal termination (reception-completion wait timeout)
6	Abnormal termination (stop-condition detection timeout)

Example of use:

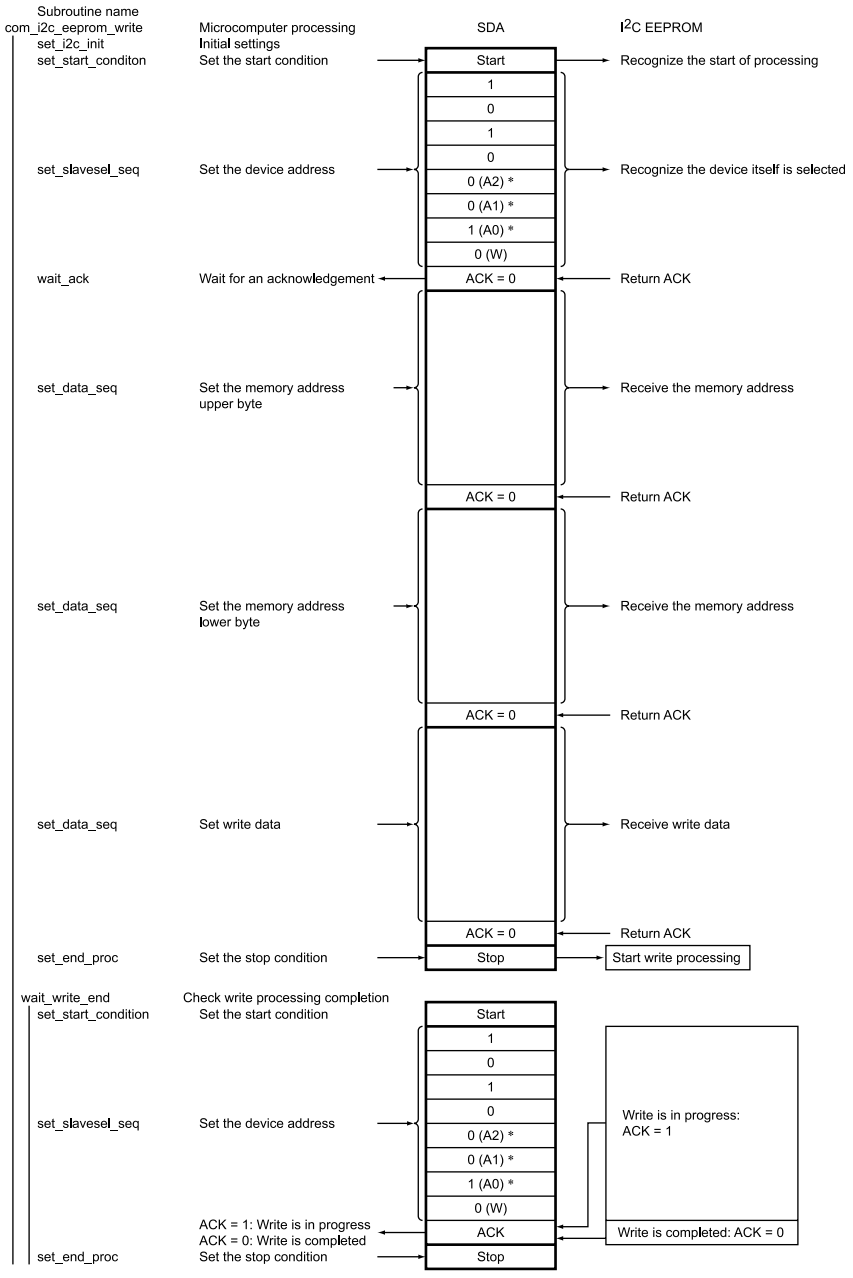
```
int ret ;
unsigned char slave_addr, *rom_data;
unsigned int  rom_length, rom_addr;

ret = com_i2c_eeprom_seq_read (slave_addr, rom_addr, rom_length, *rom_data)
```

3.5 Explanation of Operation

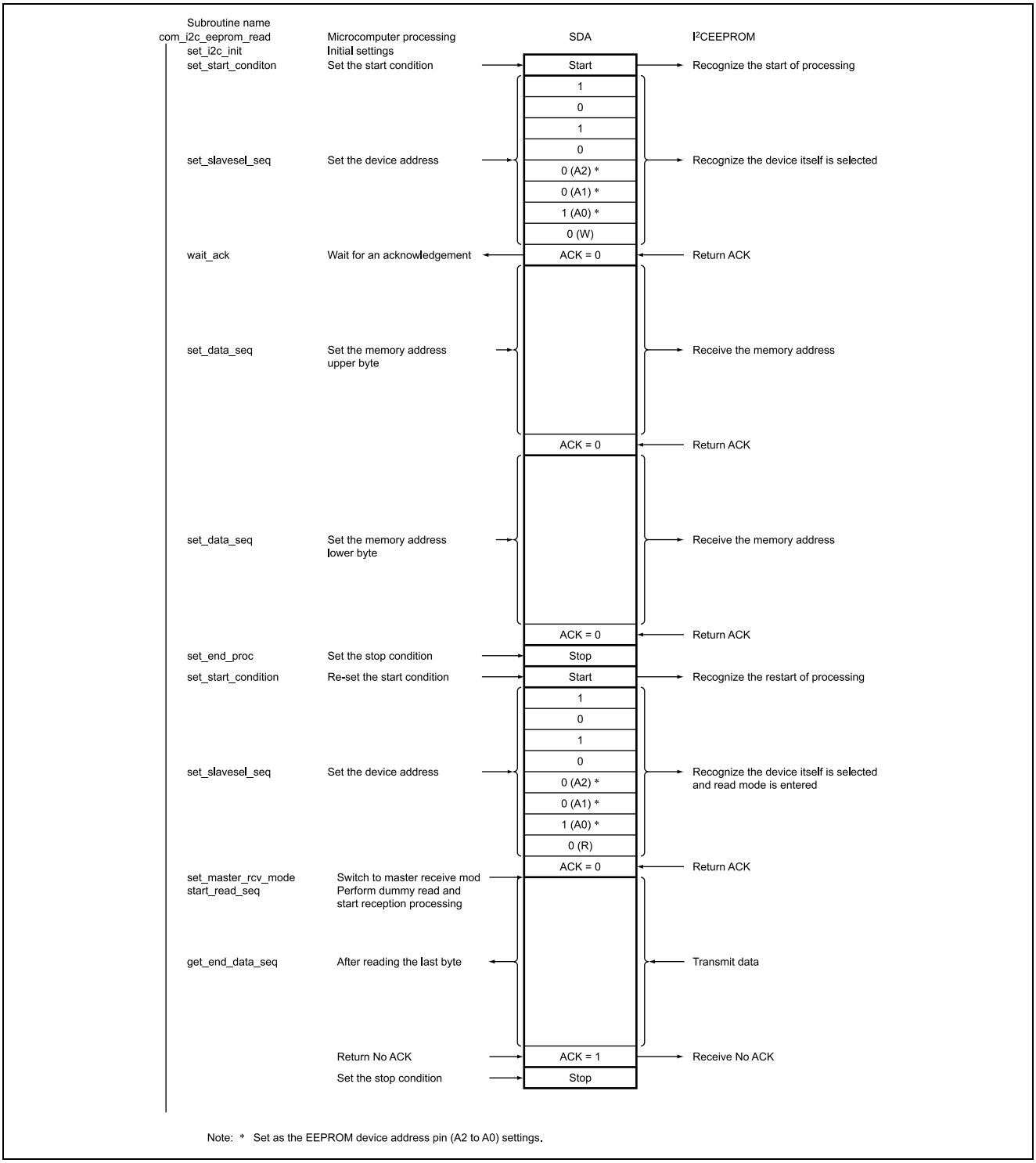
The following figure explains the operation of the H8 microcomputer and the EEPROM with respect to the SDA data flow.

1. One byte of data is written to the I2C EEPROM.



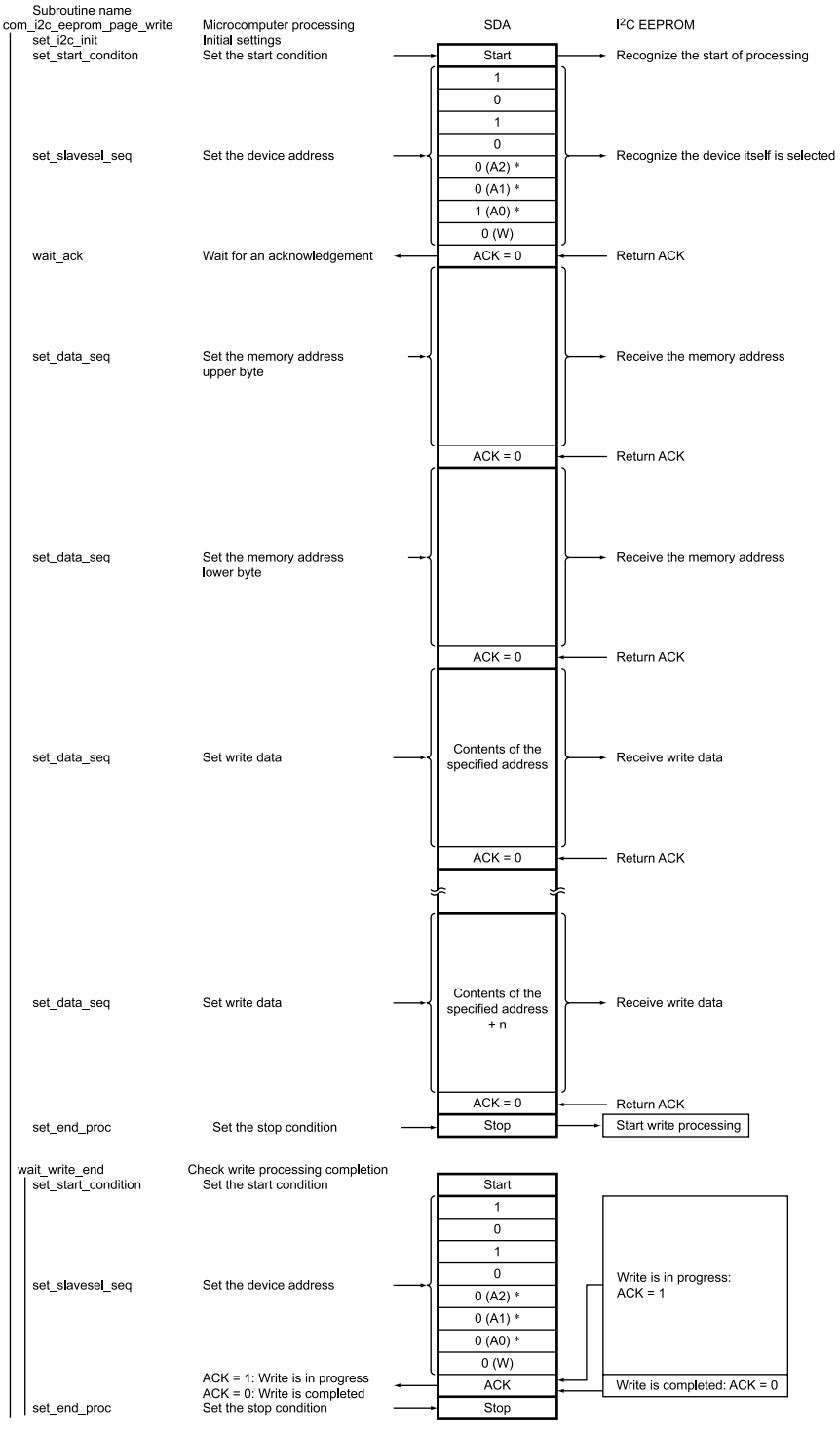
Note: * Set as the EEPROM device address pin (A2 to A0) settings.

2. One byte of data is read from the I²C EEPROM.



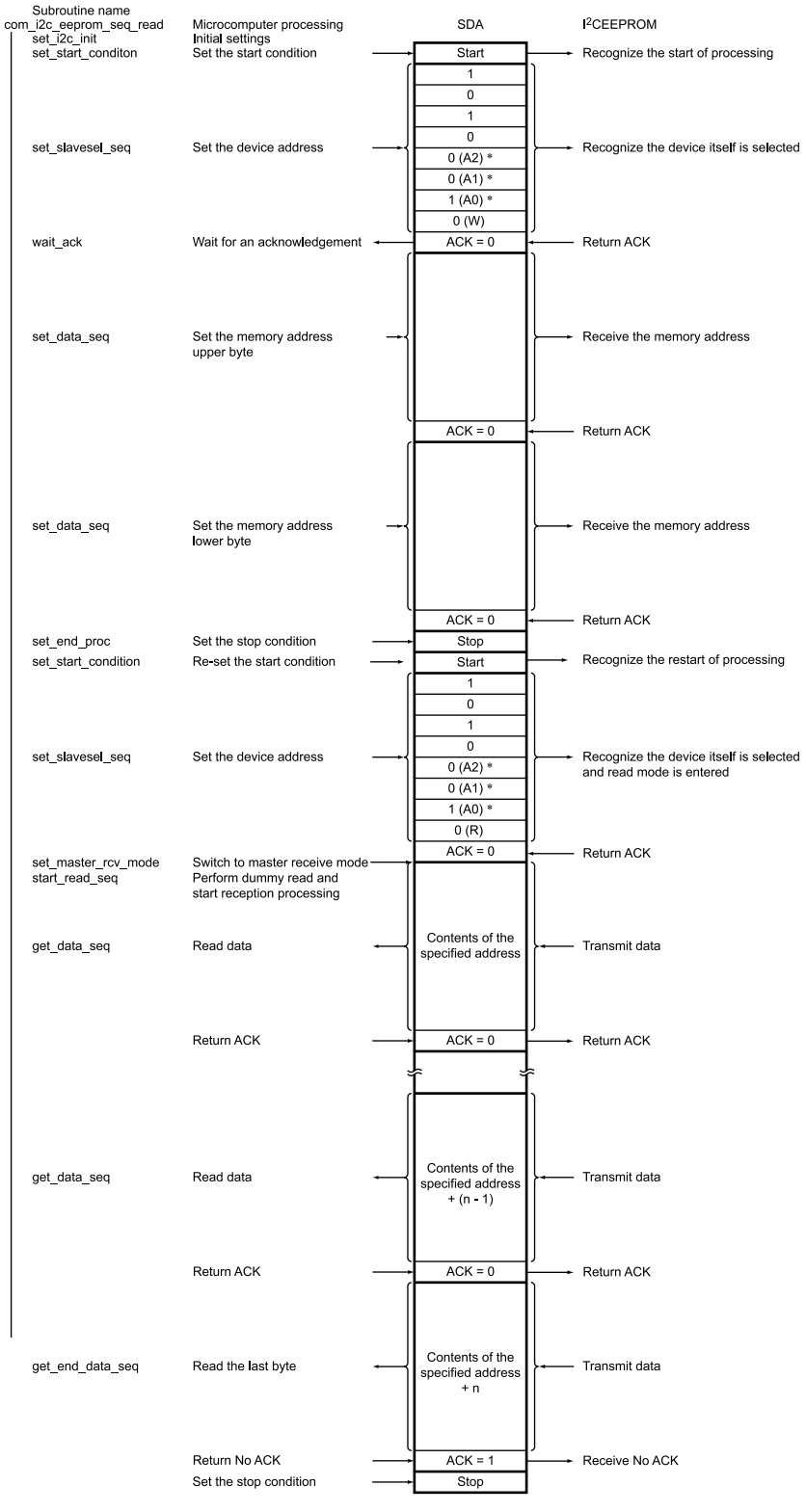
Note: * Set as the EEPROM device address pin (A2 to A0) settings.

3. The specified length of data is written to the I²C EEPROM continuously.



Note: * Set as the EEPROM device address pin (A2 to A0) settings.

4. The specified length of data is read from the I2C EEPROM continuously.



Note: * Set as the EEPROM device address pin (A2 to A0) settings.

3.6 List of Registers Used

The internal registers of the H8 microcomputer used in the sample program are listed below. For detailed information, refer to the H8/3687 Group Hardware Manual.

1. I^2C-related registers

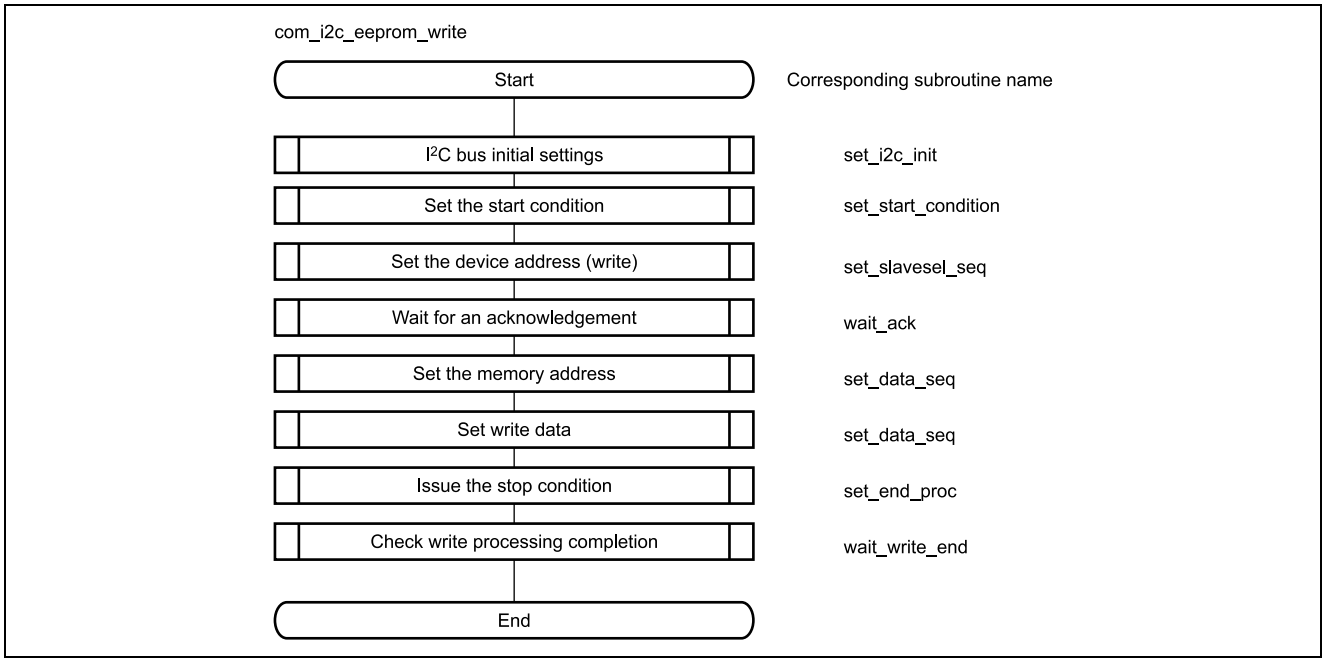
Name	Summary
I^2C bus control register 1 (ICCR1)	Starts or stops operation of the I^2C bus interface 2, controls transmission/reception, and selects master/slave mode, transmission/reception or master mode transfer clock frequency.
I^2C bus control register 2 (ICCR2)	Issues start/stop conditions, operates the SDA pin, monitors the SCL pin, and controls resets for I^2C bus interface 2 control unit.
I^2C bus mode register (ICMR)	Selects the MSB first or LSB first, performs master mode wait control, and sets the number of transfer bits.
Bus interrupt enable register (ICIER)	Enables each interrupt factor, enables/disables acknowledge, sets transmit acknowledge, and checks receive acknowledge.
I^2C bus status register (ICSR)	Checks interrupt request flags and the state.
Slave address register (SAR)	Sets the format and slave address.
I^2C bus transmit data register (ICDRT)	8-bit read/write register which stores data to be transmitted.
I^2C bus receive data register (ICDRR)	8-bit register that stores received data.

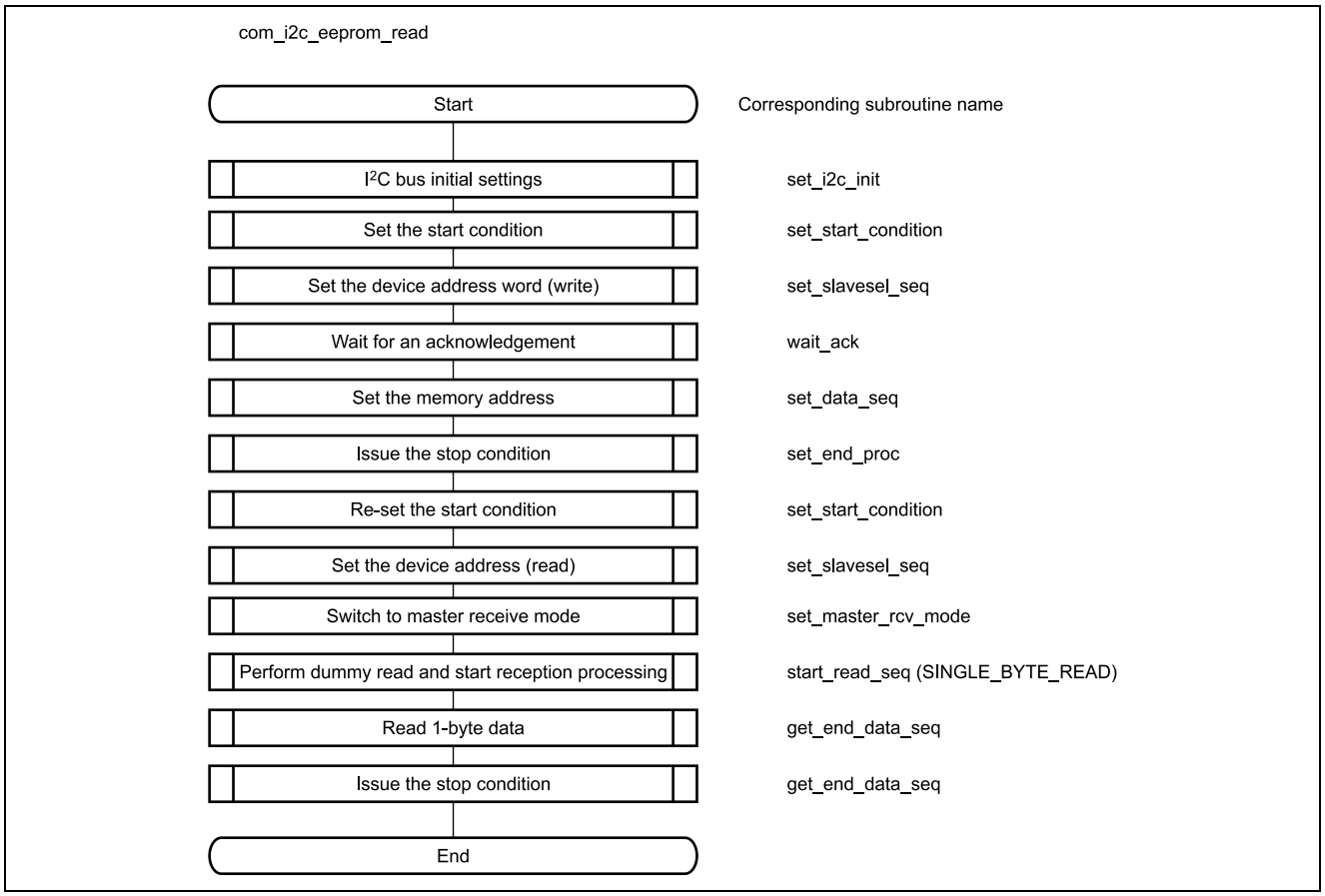
2. Timer Z-related registers
   Timer Z has various functions, but in the sample program it uses the GRA register compare-match function to generate an interrupt every 10 ms.

Name	Summary
Timer start register (TSTR)	Starts or stops TCNT operation.
Timer mode register (TMDR)	Sets buffer operation and selects synchronous operation.
Timer PWM mode register (TPMR)	Sets pins for PWM mode. Not used in this sample program.
Timer function control register (TFCR)	Selects the operating mode and output level. Not used in this sample program.
Timer output master enable register (TOER)	Enables/disables channel 0 and channel 1 output.
Timer output control register (TOCR)	Selects initial output settings before the first compare match occurs.
Timer counter (TCNT)	16-bit read/write register that counts up with the input clock.
General registers A, B, C, D (GRA, GRB, GRC, GRD)	GR is a 16-bit read/write register. Each channel has four GR registers, therefore, a total of eight registers are provided. These registers can be used as either output-compare registers or as input-capture registers, according to the TIORA and TIORC settings.
Timer control register (TCR)	Selects the TCNT counter clock, edge for an external clock, and counter clear conditions.
Timer I/O control register (TIORA)	Selects the functions of the GRA and GRB to be used as output-compare registers or as input-capture registers.
Timer status register (TSR)	Indicates the TCNT overflow/underflow generation and GRA/GRB/GRC/GRD compare match or input capture generation.
Timer interrupt enable register (TIER)	Enables/disables overflow interrupt requests or GR compare-match/input-capture interrupt requests.
PWM mode output level control register (POCR)	Controls the active level in PWM mode. Not used in this sample program.

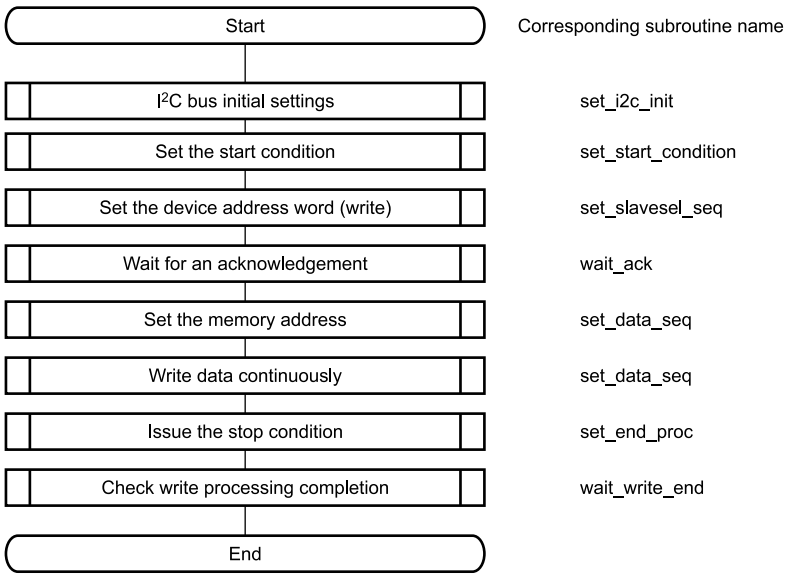
3.7 Flowcharts

The program processing flow is shown below.

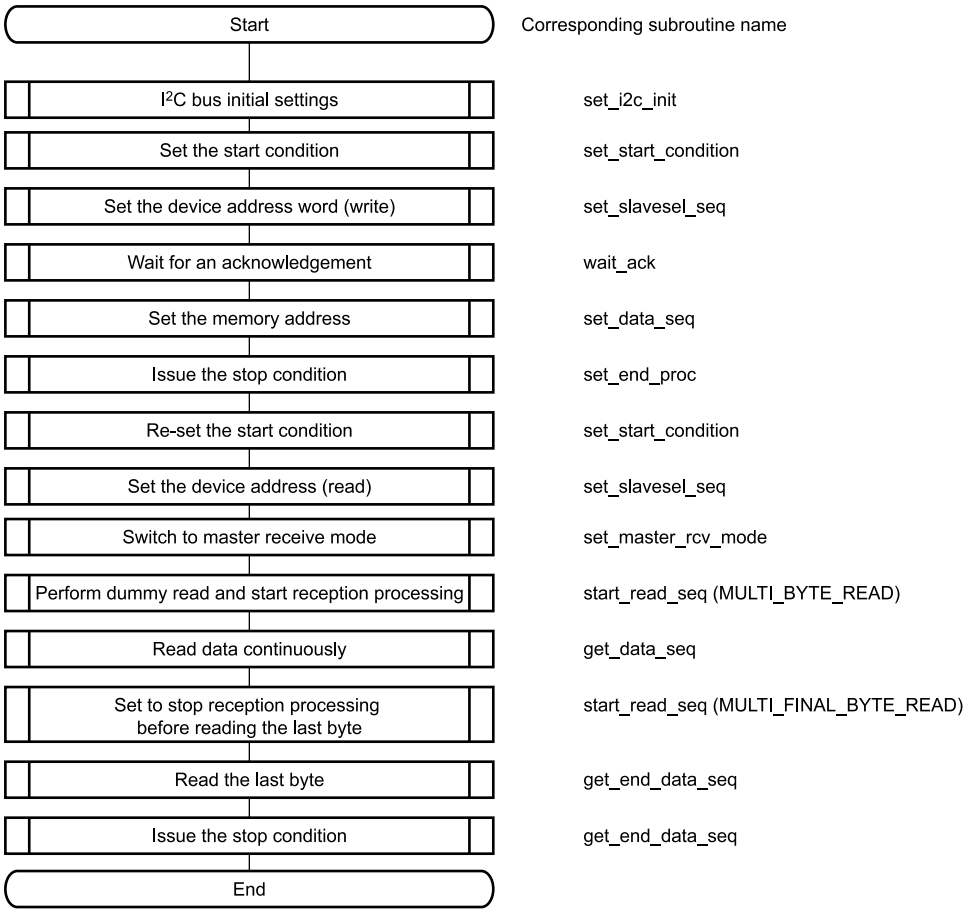


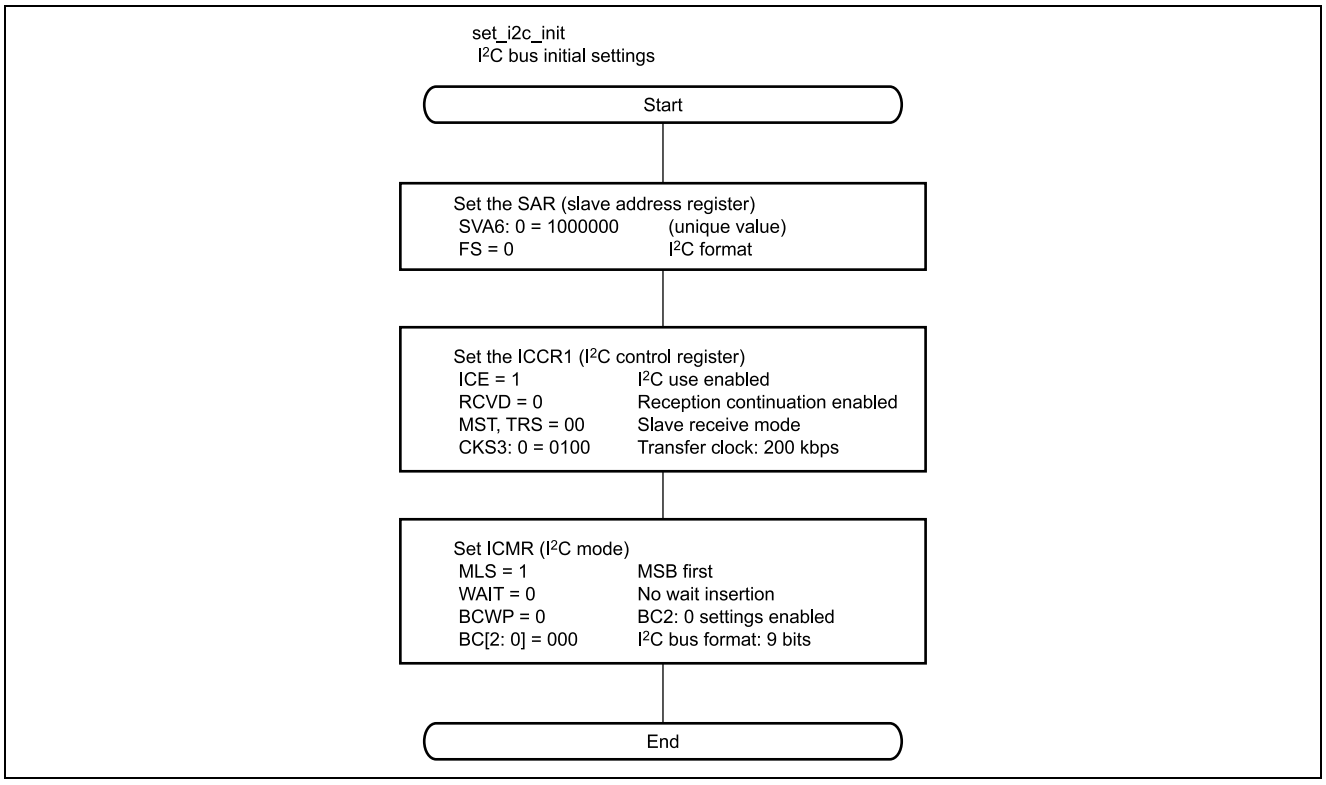


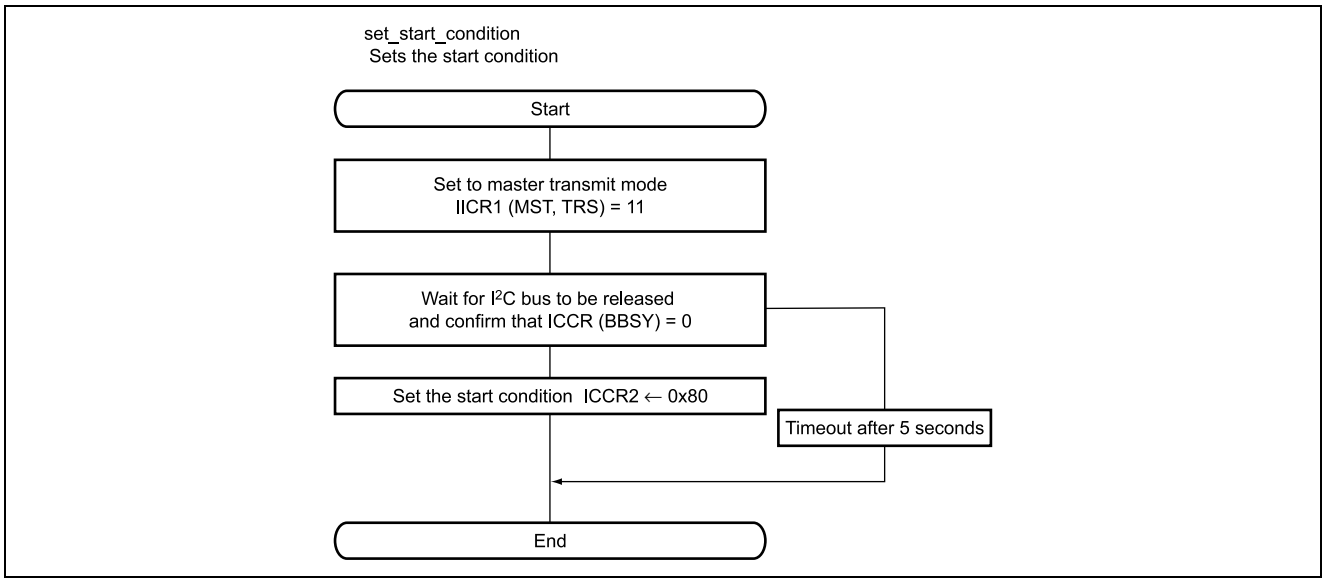
com_i2c_eeprom_page_write



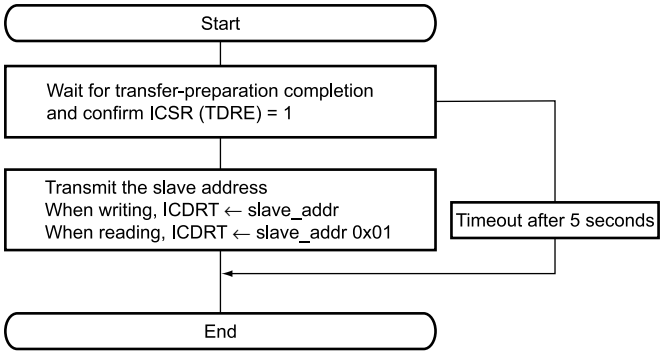
com_i2c_eeprom_seq_read



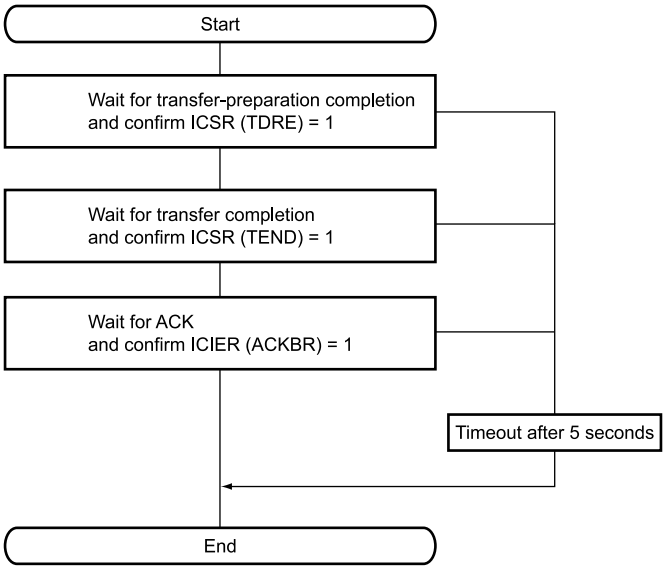

set_i2c_init
I²C bus initial settings
Start
Set the SAR (slave address register)
SVA6: 0 = 1000000 (unique value)
FS = 0 I²C format
Set the ICCR1 (I²C control register)
ICE = 1 I²C use enabled
RCVD = 0 Reception continuation enabled
MST, TRS = 00 Slave receive mode
CKS3: 0 = 0100 Transfer clock: 200 kbps
Set ICMR (I²C mode)
MLS = 1 MSB first
WAIT = 0 No wait insertion
BCWP = 0 BC2: 0 settings enabled
BC[2: 0] = 000 I²C bus format: 9 bits
End


set_start_condition
Sets the start condition
Start
Set to master transmit mode
IICR1 (MST, TRS) = 11
Wait for I²C bus to be released
and confirm that ICCR (BBSY) = 0
Set the start condition ICCR2 ← 0x80
Timeout after 5 seconds
End

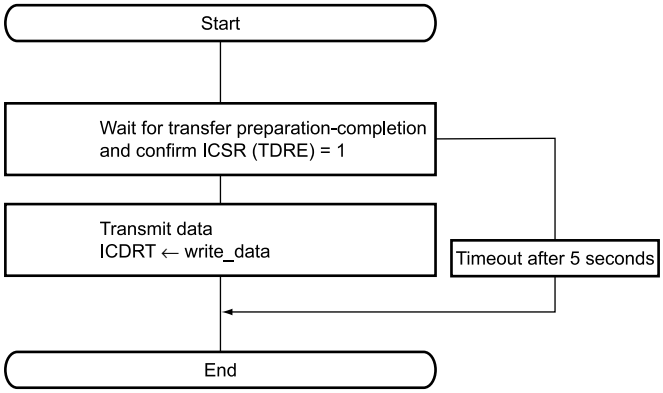
set_slavesel_seq (unsigned char mode , unsigned char slave_addr)
Executes slave selection processing
Mode: Write or read
0: Write
1: Read
slave_addr: EEPROM device address



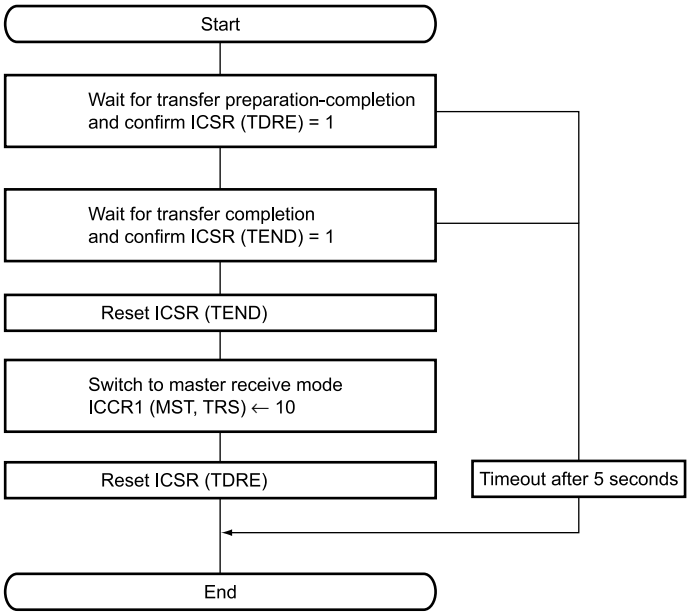
wait_ack
Waits for an acknowledgement

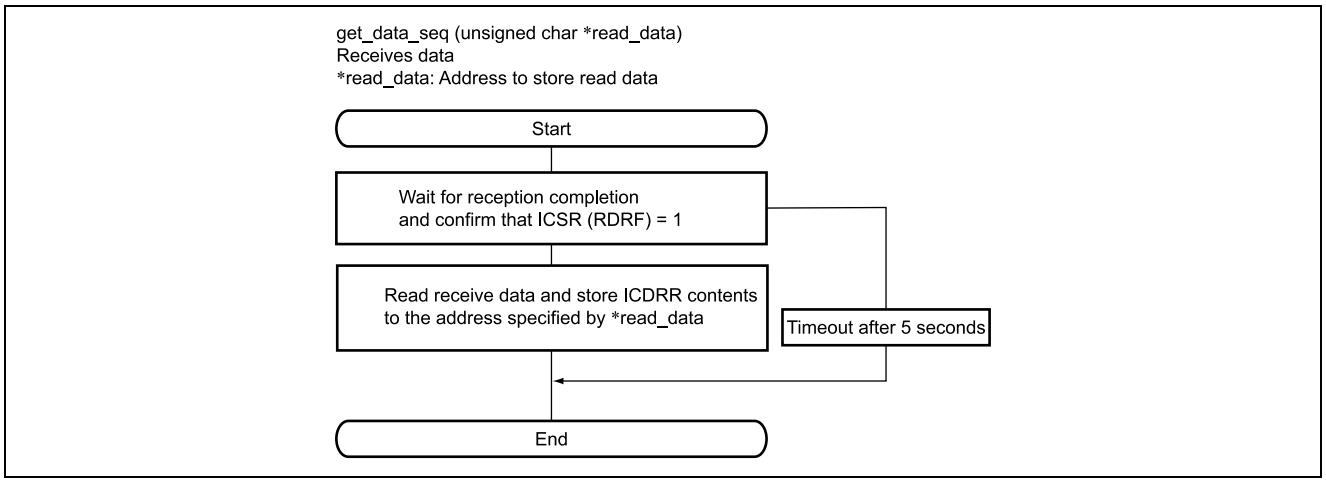


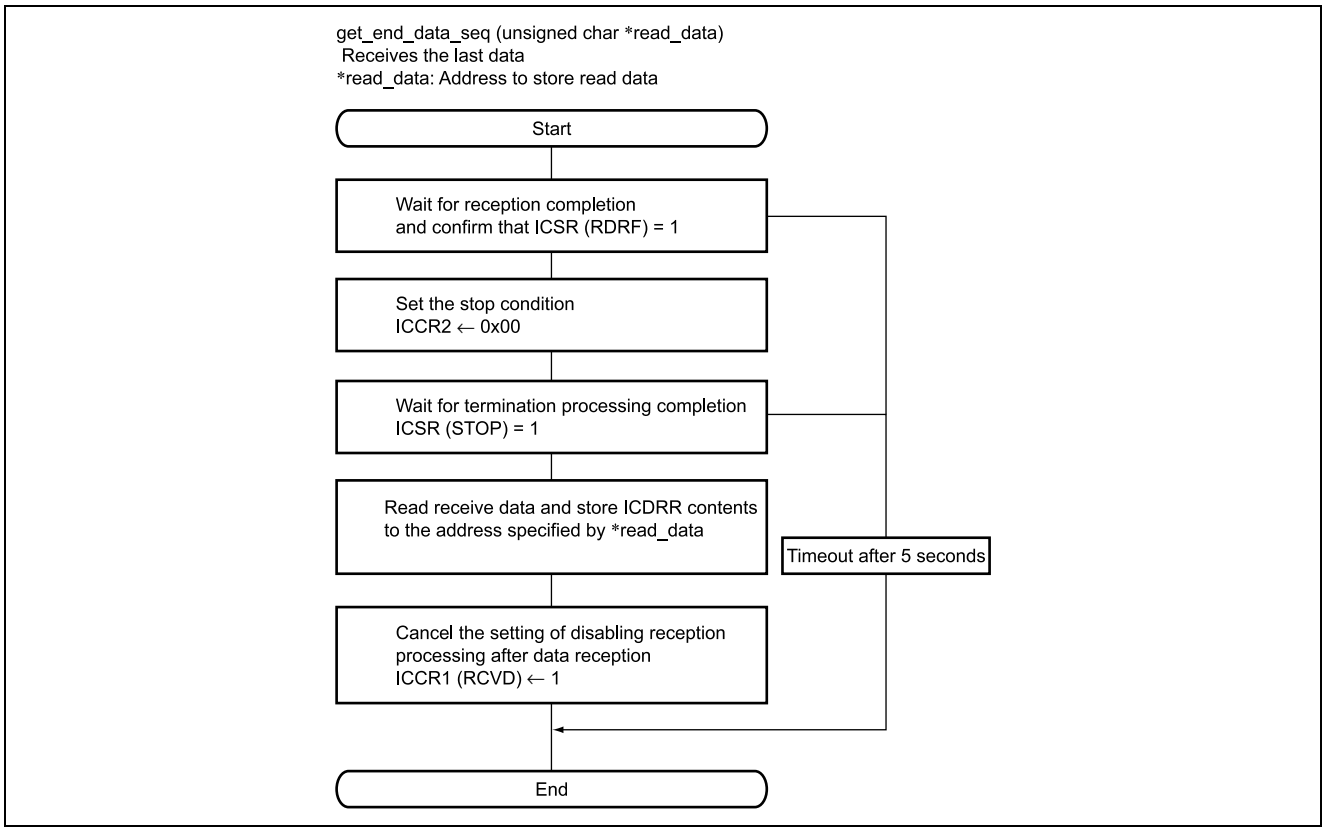
set_data_seq (unsigned char write_data)
Sets data
write_ data: Data to be transmitted



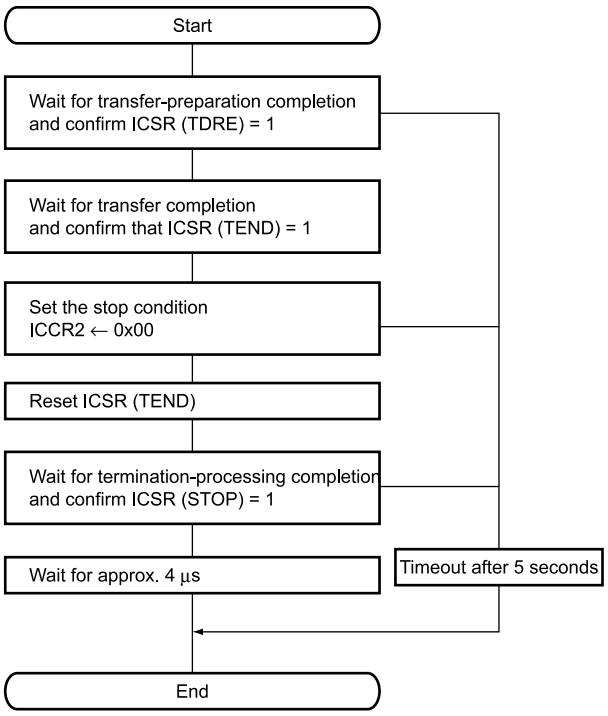
set_master_rcv_mode
Switches to master receive mode



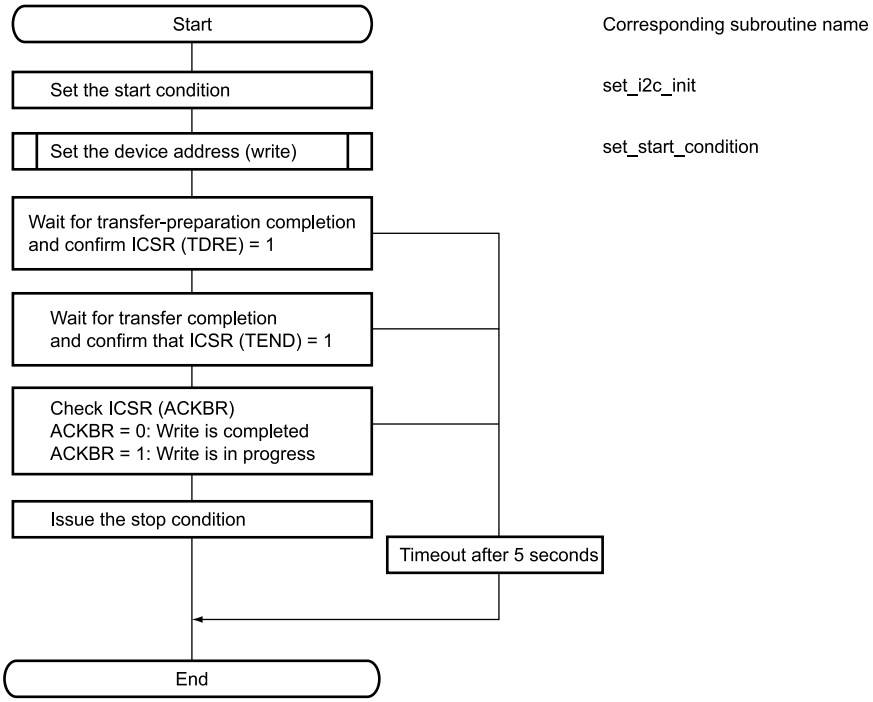

get_data_seq (unsigned char *read_data)
Receives data
*read_data: Address to store read data
Start
Wait for reception completion
and confirm that ICSR (RDRF) = 1
Read receive data and store ICDRR contents
to the address specified by *read_data
Timeout after 5 seconds
End

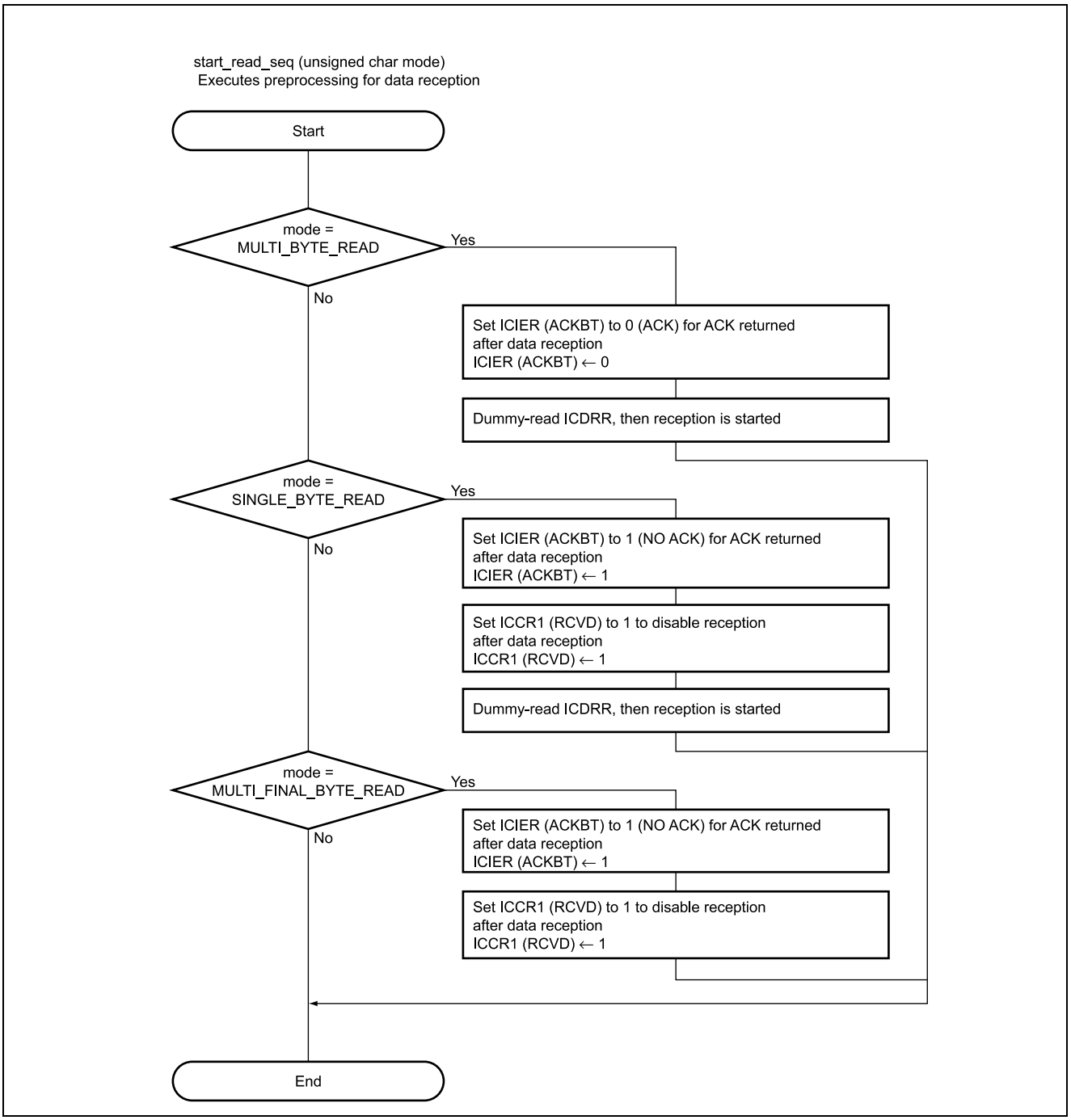

get_end_data_seq (unsigned char *read_data)
Receives the last data
*read_data: Address to store read data
Start
Wait for reception completion
and confirm that ICSR (RDRF) = 1
Set the stop condition
ICCR2 ← 0x00
Wait for termination processing completion
ICSR (STOP) = 1
Read receive data and store ICDRR contents
to the address specified by *read_data
Timeout after 5 seconds
Cancel the setting of disabling reception
processing after data reception
ICCR1 (RCVD) ← 1
End

set_end_proc
Executes termination sequence



wait_write_end (unsigned char slave_addr)
Checks writing completion
slave_addr: EEPROM device address




start_read_seq (unsigned char mode)
Executes preprocessing for data reception
Start
mode = MULTI_BYTE_READ
Yes
No
Set ICIER (ACKBT) to 0 (ACK) for ACK returned after data reception
ICIER (ACKBT) ← 0
Dummy-read ICDRR, then reception is started
mode = SINGLE_BYTE_READ
Yes
No
Set ICIER (ACKBT) to 1 (NO ACK) for ACK returned after data reception
ICIER (ACKBT) ← 1
Set ICCR1 (RCVD) to 1 to disable reception after data reception
ICCR1 (RCVD) ← 1
Dummy-read ICDRR, then reception is started
mode = MULTI_FINAL_BYTE_READ
Yes
No
Set ICIER (ACKBT) to 1 (NO ACK) for ACK returned after data reception
ICIER (ACKBT) ← 1
Set ICCR1 (RCVD) to 1 to disable reception after data reception
ICCR1 (RCVD) ← 1
End

3.8 Program Listing

```
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* 1. Sample Program 1-A #define directives ------------------------------------------------------------------------------   */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/************************************************************************************************************************/
/*   For I2CEEPROM access                                                                                                   */
/************************************************************************************************************************/
#define   CMD_WRITE_OPERATION      0
#define   DATA_READ_OPERATION      1

#define   MULTI_BYTE_READ          0
#define   SINGLE_BYTE_READ         1
#define   MULTI_FINAL_BYTE_READ    2

/************************************************************************************************************************/
/*   I2C I2CEEPROM access error code (other than 0)                                                                         */
/************************************************************************************************************************/
#define   I2C_BBSY_TOUT       1
#define   I2C_TDRE_TOUT       2
#define   I2C_ACKBR_TOUT      3
#define   I2C_TEND_TOUT       4
#define   I2C_RDRF_TOUT       5
#define   I2C_STOP_TOUT       6
#define   I2C_TRS_TOUT        7


/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* 2. Sample program 1-B Prototype declaration -------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/************************************************************************************************************************/
/************************************************************************************************************************/
/*   I2C BUS access processing                                                                                              */
/************************************************************************************************************************/
/************************************************************************************************************************/
void set_i2c_init( );
unsigned int set_start_condition( );
unsigned int wait_ack();
unsigned int set_slavesel_seq (unsigned char mode ,unsigned char slave_addr ) ;
unsigned int set_data_seq(unsigned char write_data);
unsigned int set_end_proc ();
unsigned int set_master_rcv_mode () ;
unsigned int wait_write_end (unsigned char device_addr_code ) ;

void start_read_seq (unsigned char mode) ;
unsigned int get_end_data_seq (unsigned char *read_data);
unsigned int get_data_seq (unsigned char *read_data);

unsigned int com_i2c_eeprom_read( unsigned char device_addr_code , unsigned int rom_addr , unsigned char *rom_data );
unsigned int com_i2c_eeprom_write( unsigned char device_addr_code , unsigned int rom_addr , unsigned char rom_data );
unsigned int com_i2c_eeprom_seq_read
               ( unsigned char device_addr_code , unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data ) ;
unsigned int com_i2c_eeprom_page_write
               ( unsigned char device_addr_code , unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data ) ;
```

```
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* 3. Sample program 1-C Source codes ---^-------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/************************************************************************************************************************/
/************************************************************************************************************************/
/************************************************************************************************************************/
/*                                                                                                                      */
/*                                I2C EEPROM control                                                                    */
/*                                                                                                                      */
/************************************************************************************************************************/
/************************************************************************************************************************/
/************************************************************************************************************************/

/************************************************************************************************************************/
/*   1. Module name : set_i2c_init                                                                                      */
/*   2. Function overview : Sets initial settings prior to I2 access                                                    */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                     */
/*                            000         2002.12.14            Ueda                    New                             */
/************************************************************************************************************************/
void set_i2c_init( )
{
     /*******************************************************************************************************************/
     /*   SAR            Sets the slave address register                                                                */
     /*        SVA6:0    = 1000000 (Unique value)                                                                       */
     /*        FS        = 0 I2C format                                                                                 */
     /*******************************************************************************************************************/
          /* ##(program note)############################################################################################ */
          /* ## SVA6: 0 is used in the slave mode. It should be set to a unique address that is different              ## */
          /* ## from the addresses used for other slave devices connected to the I2C bus                               ## */
          /* ############################################################################################################ */

     IIC2.SAR.BYTE= 0x80 ;

     /*******************************************************************************************************************/
     /*   ICCR1          Sets the I2C control register                                                                  */
     /*        ICE       = 1  I2C use enabled                                                                           */
     /*        RCVD      = 0  Reception disabled                                                                        */
     /*        MST,TRS   = 00 Slave receive mode                                                                        */
     /*        CKS3:0  = 0100 Transfer clock kHz (ϕ/80, transfer rate: 200 kbps)                                        */
     /*******************************************************************************************************************/

     IIC2.ICCR1.BYTE     = 0x84 ;
          /* ##(program note)############################################################################################ */
          /* ## The value set for CKS3:0 should be modified depending on the necessary transfer rate.                ## */
          /* ## For details, refer to the H8/3687 Hardware Manual.                                                    ## */
          /* ############################################################################################################ */

     /*******************************************************************************************************************/
     /*   ICMR      Sets I2C mode                                                                                      */
     /*        MLS       = 0 MSB first                                                                                  */
     /*        WAIT      = 0 No wait inserted                                                                           */
     /*        BCWP      = 0 BC2:0 setting enabled                                                                      */
     /*        BC[2:0]   = 000 I2C bus format: 9 bits                                                                   */
     /*******************************************************************************************************************/
     IIC2.ICMR.BYTE= 0x00 ;

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : set_start_condition                                                                                                */
/*   2. Function overview : Sets the I2C start condition.                                                                                */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                                      */
/*                            000          2002.12.14             Ueda                    New                                           */
/*****************************************************************************************************************************************/
unsigned int set_start_condition( )
{
     int ret , Timer_wk;

     ret = NORMAL_END ;

     /**************************************************************************************************************************************/
     /*   Confirms that ICCR2 (BBSY) = 0.                                                                                                  */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICCR2.BIT.BBSY == 1){                                        /* Waits for the I2C bus to be released             */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                                 /* 5 If the value remains 1 for 5 seconds,          */
                                                                              /* exit with an error.                              */
               ret = I2C_BBSY_TOUT;                                           /* Abnormal termination  (timeout)                  */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICCR2.BIT.BBSY= 0;
          #endif

     }

     /**************************************************************************************************************************************/
     /*   Sets the master transmit mode                                                                                                    */
     /**************************************************************************************************************************************/
     IIC2.ICCR1.BYTE = 0xB4   ;                                               /* Sets the master transmit mode                    */

     /**************************************************************************************************************************************/
     /*   Sets the start condition                                                                                                         */
     /**************************************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x80 ;

          /* ##(program note)################################################################################################### */
          /* ## The settings for bits 7 and 6, which set the start condition, have to be set simultaneously,                ## */
          /* ## so they must be written in byte units                                                                        ## */
          /* ## Note that if these are set one bit at a time, the start condition may not be set properly.                   ## */
          /* ################################################################################################################### */

exit :
     return (ret);

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : set_slavesel_seq                                                                                                   */
/*   2. Function overview : Executes I2C slave selection processing.                                                                     */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                                      */
/*                            000           2002.12.14          Ueda                   New                                             */
/*****************************************************************************************************************************************/
unsigned int set_slavesel_seq (unsigned char mode ,unsigned char slave_addr )
{
     int ret , Timer_wk;
     unsigned char write_data ;

     ret = NORMAL_END ;

     /*****************************************************************************************************************************************/
     /*   Confirms that ICSR (TDRE) = 1.                                                                                                    */
     /*****************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TDRE == 0){                                        /* Waits until the preparation                          */
                                                                             /* for transfer has been completed.                     */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                                /* If the value remains 1 for 5 seconds,                */
                                                                             /* exit with an error.                                  */
               ret = I2C_TDRE_TOUT;                                          /* Abnormal termination (timeout)                       */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TDRE = 1 ;
          #endif

     }

     /*****************************************************************************************************************************************/
     /*   Sets the slave address                                                                                                            */
     /*****************************************************************************************************************************************/
     if (mode == DATA_READ_OPERATION){
          slave_addr = slave_addr | 0x01 ;
     }

     IIC2.ICDRT = slave_addr ;
exit :
     return (ret);

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : wait_ack                                                                                                           */
/*   22. Function overview : Waits for the I2C ACK.                                                                                      */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                                      */
/*                            000          2002.12.14           Ueda                    New                                             */
/*****************************************************************************************************************************************/
unsigned int wait_ack ()
{
     int ret , Timer_wk;

     ret = NORMAL_END ;

     /**************************************************************************************************************************************/
     /*   Confirms that ICSR (TDRE) = 1.                                                                                                   */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TDRE == 0){                                        /* Waits until the preparation                        */
                                                                             /* for transfer has been completed.                   */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                                /* If the value remains 1 for 5 seconds,              */
                                                                             /* exit with an error.                                */
               ret = I2C_TDRE_TOUT;                                          /* Abnormal termination (timeout)                     */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TDRE = 1 ;
          #endif

     }

     /**************************************************************************************************************************************/
     /*   Confirms that ICSR (TEND) = 1.                                                                                                   */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TEND == 0){                                        /* Waits until the transfer has been completed.       */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                                /* If the value remains 1 for 5 seconds,              */
                                                                             /* exit with an error.                                */
               ret = I2C_TEND_TOUT;                                          /* Abnormal termination (timeout)                     */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TEND = 1 ;
          #endif

     }
```

```
     /*****************************************************************************************************************************/
     /*   Confirms that ICIER (ACKBR) = 1.                                                                                        */
     /*****************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICIER.BIT.ACKBR == 1){                                  /* Waits for ACK to be returned.                  */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,          */
                                                                         /* exit with an error.                            */
               ret = I2C_ACKBR_TOUT;                                     /* Abnormal termination (timeout)                 */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICIER.BIT.ACKBR = 0 ;
          #endif

     }

exit :
     return (ret);

}

/*****************************************************************************************************************************/
/*   1. Module name : set_data_seq                                                                                           */
/*   2. Function overview : Executes I2C data setting processing                                                             */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                        */
/*                            000        2002.12.14             Ueda                      New                               */
/*****************************************************************************************************************************/
unsigned int set_data_seq (unsigned char write_data)
{
     int ret , Timer_wk;

     ret = NORMAL_END ;

     /*****************************************************************************************************************************/
     /*   Confirms that ICSR (TDRE) = 1.                                                                                          */
     /*****************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TDRE == 0){                                    /* Waits until the preparation                    */
                                                                         /* for transfer has been completed.               */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,          */
                                                                         /* exit with an error.                            */
               ret = I2C_TDRE_TOUT;                                      /* Abnormal termination  (timeout)                */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TDRE = 1 ;
          #endif

     }

     /*****************************************************************************************************************************/
     /*   Sets data                                                                                                               */
     /*****************************************************************************************************************************/
     IIC2.ICDRT = write_data ;                                           /* dummy write                                    */

exit :
     return (ret);

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : set_master_rcv_mode                                                                                                */
/*   2. Function overview : Switches to master receive mode                                                                              */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                                     */
/*                             000          2002.12.14           Ueda                    New                                            */
/*****************************************************************************************************************************************/
unsigned int set_master_rcv_mode ()
{
     int ret , Timer_wk;
     unsigned char dummy_data ;

     ret = NORMAL_END ;

     /**************************************************************************************************************************************/
     /*   Confirms that ICSR (TDRE) = 1.                                                                                                   */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TDRE == 0){                                    /* Waits until the preparation                    */
                                                                         /* for transfer has been completed.               */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,          */
                                                                         /* exit with an error.                            */
               ret = I2C_TDRE_TOUT;                                      /* Abnormal termination (timeout)                 */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TDRE = 1 ;
          #endif

     }

     /**************************************************************************************************************************************/
     /*   Confirms that ICSR (TEND) = 1.                                                                                                   */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TEND == 0){                                    /* Waits until the preparation                    */
                                                                         /* for transfer has been completed.               */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,          */
                                                                         /* exit with an error.                            */
               ret = I2C_TEND_TOUT;                                      /* Abnormal termination  (timeout)                */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TEND = 1 ;
          #endif
     }

     /**************************************************************************************************************************************/
     /*   Resets ICSR (TEND)                                                                                                               */
     /**************************************************************************************************************************************/
     IIC2.ICSR.BIT.TEND = 0 ;

     /**************************************************************************************************************************************/
     /*   Switches to master receive mode                                                                                                  */
     /**************************************************************************************************************************************/
     IIC2.ICCR1.BYTE = 0xA4   ;
```

```
     /*****************************************************************************************************************************/
     /*   Resets ICSR (TDRE)                                                                                                      */
     /*****************************************************************************************************************************/
     IIC2.ICSR.BIT.TDRE = 0 ;

exit :
     return (ret);
}

/*****************************************************************************************************************************/
/*   1. Module name : start_read_seq                                                                                         */
/*   2. Function overview : Carries out a dummy read at the start of read processing                                         */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                        */
/*                             000          2002.12.14           Ueda                    New                                  */
/*****************************************************************************************************************************/
void start_read_seq (unsigned char mode)
{
     int ret , Timer_wk;
     unsigned char dummy_data ;

     ret = NORMAL_END ;

     if (mode == MULTI_BYTE_READ) {
          /*****************************************************************************************************************************/
          /*   Sets value for ACK returned after data reception to "0" (ACK)                                                         */
          /*****************************************************************************************************************************/
          IIC2.ICIER.BIT.ACKBT = 0 ;

          /*****************************************************************************************************************************/
          /*   Initiates reception when a dummy read is carried out                                                                  */
          /*****************************************************************************************************************************/
          dummy_data = IIC2.ICDRR ;
               /* ##(program note)############################################################################################ */
               /* ## Reception begins when a dummy read is carried out, and data is sent                                        ## */
               /* ## from the device synchronized to the SCL.                                                                   ## */
               /* ## A low-level signal is sent to the device synchronized to the ninth SCL,                                    ## */
               /* ## in response to the ICSR (ACKB) previously set to 0.                                                        ## */
               /* ############################################################################################################ */

     }
     if (mode == SINGLE_BYTE_READ) {
          /*****************************************************************************************************************************/
          /*   Sets value for ACK returned after data reception to "1" (NOACK)                                                       */
          /*****************************************************************************************************************************/
          IIC2.ICIER.BIT.ACKBT = 1 ;

          /*****************************************************************************************************************************/
          /*   Disables the next reception after data reception                                                                      */
          /*****************************************************************************************************************************/
          IIC2.ICCR1.BIT.RCVD = 1  ;

          /*****************************************************************************************************************************/
          /*   Initiates reception when a dummy read is carried out                                                                  */
          /*****************************************************************************************************************************/
          dummy_data = IIC2.ICDRR ;
               /* ##(program note)############################################################################################ */
               /* ## Reception begins when a dummy read is carried out, and data is sent                                      ## */
               /* ##  from the device synchronized to the SCL.                                                                  ## */
               /* ## A high level signal is sent to the device synchronized to the ninth SCL,                                 ## */
               /* ## Ain response to IIC2.ICIER.BIT.ACKBT previously set to 1.                                                 ## */
               /* ## The SCL clock for the next reception is not sent in response                                             ## */
               /* ## to IIC2.ICIER.BIT.ACKBT previously set to 1.                                                             ## */
               /* ############################################################################################################ */

     }
```

```
    if (mode ==MULTI_FINAL_BYTE_READ) {
        /*****************************************************************************************************************/
        /*   Sets value for ACK returned after data reception to "1" (NOACK)                                             */
        /*****************************************************************************************************************/
        IIC2.ICIER.BIT.ACKBT = 1 ;

        /*****************************************************************************************************************/
        /*   Disables the next reception after data reception                                                            */
        /*****************************************************************************************************************/
        IIC2.ICCR1.BIT.RCVD = 1  ;
    }
}

/*****************************************************************************************************************/
/*   1. Module name : get_data_seq                                                                               */
/*   2. Function overview : Reads data from the I2C target device                                                */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents             */
/*                             000          2002.12.14          Ueda                    New                     */
/*****************************************************************************************************************/
unsigned int get_data_seq (unsigned char *read_data)
{
    int ret , Timer_wk;
    unsigned char dummy_data ;

    ret = NORMAL_END ;

    /*****************************************************************************************************************/
    /*   Confirms that ICSR (RDRF) = 1.                                                                              */
    /*****************************************************************************************************************/
    com_timer.wait_100ms_scan = 50 ;
    while (IIC2.ICSR.BIT.RDRF == 0){                                    /* Waits until the reception has been completed.    */
        Timer_wk = com_timer.wait_100ms_scan ;
        if (Timer_wk == 0){                                             /* If the value remains 1 for 5 seconds,            */
                                                                        /* exit with an error.                              */
            ret = I2C_RDRF_TOUT;                                        /* Abnormal termination  (timeout)                  */
            goto exit ;
        }

        #ifdef UT
            IIC2.ICSR.BIT.RDRF = 1 ;
        #endif
    }

    /*****************************************************************************************************************/
    /*   Reads received data.                                                                                        */
    /*****************************************************************************************************************/
    *read_data = IIC2.ICDRR ;                                           /* data read                                        */


exit :
    return (ret);
}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : get_end_data_seq                                                                                                   */
/*   2. Function overview : Reads data from the I2C target device                                                                        */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                                     */
/*                             000          2002.12.14            Ueda                    New                                           */
/*****************************************************************************************************************************************/
unsigned int get_end_data_seq (unsigned char *read_data)
{
     int ret , Timer_wk;
     unsigned char dummy_data ;

     ret = NORMAL_END ;

     /**************************************************************************************************************************************/
     /*   Confirms that ICSR(RDRF) = 1.                                                                                                    */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.RDRF == 0){                                    /* Waits until the reception has been completed.    */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,            */
                                                                         /* exit with an error.                              */
               ret = I2C_RDRF_TOUT;                                      /* Abnormal termination  (timeout)                  */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.RDRF = 1 ;
          #endif
     }


     /**************************************************************************************************************************************/
     /*   Sets the stop condition                                                                                                          */
     /**************************************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x00 ;

     /**************************************************************************************************************************************/
     /*   Confirms that ICSR (STOP) = 1.                                                                                                   */
     /**************************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.STOP == 0){                                    /* Waits until the stop condition is detected.      */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,            */
                                                                         /* exit with an error.                              */
               ret = I2C_STOP_TOUT;                                      /* Abnormal termination  (timeout)                  */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.RDRF = 1 ;
          #endif
     }

     /**************************************************************************************************************************************/
     /*   Reads received data                                                                                                              */
     /**************************************************************************************************************************************/
     *read_data = IIC2.ICDRR ;                                           /* data read                                        */
```

```
     /*****************************************************************************************************************************/
     /*   Cancels the disable setting of the next reception after data reception                                                  */
     /*****************************************************************************************************************************/
     IIC2.ICCR1.BIT.RCVD = 0  ;                                          /* Cancels the reception disable setting           */

exit :

     return (ret);
}

/*****************************************************************************************************************************/
/*   1. Module name : set_end_proc                                                                                           */
/*   2. Function overview : Executes an I2C exit sequence                                                                    */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                        */
/*                             000         2002.12.14             Ueda                    New                                */
/*****************************************************************************************************************************/
unsigned int set_end_proc ()
{
     int ret , Timer_wk;

     ret = NORMAL_END ;

     /*****************************************************************************************************************************/
     /*   Confirms that ICSR (TDRE) = 1.                                                                                          */
     /*****************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TDRE == 0){                                    /* Waits until the preparation                     */
                                                                         /* for transfer has been completed.                */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,           */
                                                                         /* exit with an error.                             */
               ret = I2C_TDRE_TOUT;                                      /* Abnormal termination  (timeout)                 */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TDRE = 1 ;
          #endif

     }

     /*****************************************************************************************************************************/
     /*   Confirms that ICSR (TEND) = 1.                                                                                          */
     /*****************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TEND == 0){                                    /* Waits until the transfer has been completed.    */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                            /* If the value remains 1 for 5 seconds,           */
                                                                         /* exit with an error.                             */
               ret = I2C_TEND_TOUT;                                      /* Abnormal termination  (timeout)                 */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.TEND = 1 ;
          #endif
     }
```

```
     /*****************************************************************************************************************************/
     /*   Sets the stop condition                                                                                                 */
     /*****************************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x00 ;                                        /* Sets the stop condition.                          */

     /*****************************************************************************************************************************/
     /*   Resets ICSR (TEND).                                                                                                     */
     /*****************************************************************************************************************************/
     IIC2.ICSR.BIT.TEND = 0 ;

     /*****************************************************************************************************************************/
     /*   Confirms that ICSR (STOP) = 1.                                                                                          */
     /*****************************************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.STOP == 0){                                  /* Waits until the stop condition is detected.       */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                          /* If the value remains 1 for 5 seconds,             */
                                                                       /* exit with an error.                               */
               ret = I2C_STOP_TOUT;                                    /* Abnormal termination  (timeout)                   */
               goto exit ;
          }

          #ifdef UT
               IIC2.ICSR.BIT.RDRF = 1 ;
          #endif
     }

/*   * Waits for approx. 4 µs.     *
     com_delay(5) ;
*/

exit :
     return (ret);

}
/*****************************************************************************************************************************/
/*   1. Module name : wait_write_end                                                                                         */
/*   2. Function overview : Checks write completion of I2C                                                                   */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                        */
/*                             000        2002.12.14              Ueda                    New                                */
/*****************************************************************************************************************************/
unsigned int wait_write_end (unsigned char slave_addr )
{
     int ret , Timer_wk;

     ret = NORMAL_END ;

     com_timer.wait_100ms = 50 ;

     do{
          /*****************************************************************************************************************************/
          /*   Sets the start condition.                                                                                               */
          /*****************************************************************************************************************************/
          ret = set_start_condition() ;                                /* Sets the start condition                          */
               if (ret !=0) { goto exit ;}

          /*****************************************************************************************************************************/
          /*   Sets the device address word (write)                                                                                    */
          /*****************************************************************************************************************************/
          ret = set_slavesel_seq ( CMD_WRITE_OPERATION , slave_addr ) ;
               if (ret !=0) { goto exit ;}
```

```
        /*****************************************************************************************************************/
        /*   Confirms that ICSR (TDRE) = 1.                                                                              */
        /*****************************************************************************************************************/
        com_timer.wait_100ms_scan = 50 ;
        while (IIC2.ICSR.BIT.TDRE == 0){                                /* Waits until the preparation                   */
                                                                        /* for transfer has been completed.              */
             Timer_wk = com_timer.wait_100ms_scan ;
             if (Timer_wk == 0){                                        /* If the value remains 1 for 5 seconds,         */
                                                                        /* exit with an error.                           */
                  ret = I2C_TDRE_TOUT;                                  /* Abnormal termination  (timeout)               */
                  goto exit ;
             }

             #ifdef UT
                  IIC2.ICSR.BIT.TDRE = 1 ;
             #endif

        }

        /*****************************************************************************************************************/
        /*   Confirms that ICSR (TEND) = 1.                                                                              */
        /*****************************************************************************************************************/
        com_timer.wait_100ms_scan = 50 ;
        while (IIC2.ICSR.BIT.TEND == 0){                                /* Waits until the transfer has been completed.  */
             Timer_wk = com_timer.wait_100ms_scan ;
             if (Timer_wk == 0){                                        /* If the value remains 1 for 5 seconds,         */
                                                                        /* exit with an error.                           */
                  ret = I2C_TEND_TOUT;                                  /* Abnormal termination  (timeout)               */
                  goto exit ;
             }

             #ifdef UT
                  IIC2.ICSR.BIT.TEND = 1 ;
             #endif

        }

        /*****************************************************************************************************************/
        /*   Checks ICIER (ACKBR): ACK = 0  Write is completed  ACK = 1  Write is in progress.                           */
        /*****************************************************************************************************************/
        if (com_timer.wait_100ms == 0){                                 /* If the value remains 1 for 5 seconds,         */
                                                                        /* exit with an error.                           */
             ret = I2C_ACKBR_TOUT;                                      /* Abnormal termination  (timeout                */
             goto exit ;
        }

        #ifdef UT
             IIC2.ICIER.BIT.ACKBR = 1 ;
        #endif

        /*****************************************************************************************************************/
        /*   Issues the stop condition                                                                                   */
        /*****************************************************************************************************************/
        ret = set_end_proc ( ) ;
             if (ret !=0) { goto exit ;}

     } while (IIC2.ICIER.BIT.ACKBR == 1) ;                              /* Iterate the loop while ACK = 1.               */

exit :
     return (ret);

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : com_i2c_eeprom_read                                                                                                */
/*   2. Function overview : Reads 1-byte data from I2CEEPROM.                                                                            */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                                     */
/*                             000        2002.12.14             Ueda                    New                                            */
/*****************************************************************************************************************************************/
unsigned int com_i2c_eeprom_read ( unsigned char slave_addr , unsigned int rom_addr , unsigned char *rom_data )
{
     int ret ;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;


     ret = NORMAL_END ;

     /*****************************************************************************************************************************************/
     /*   Initializes the I2C bus                                                                                                             */
     /*****************************************************************************************************************************************/
     set_i2c_init () ;

     /*****************************************************************************************************************************************/
     /*   Sets the start condition                                                                                                            */
     /*****************************************************************************************************************************************/
     ret = set_start_condition() ;                                        /* Sets the start condition                        */
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the device address word (write).                                                                                               */
     /*****************************************************************************************************************************************/
     ret = set_slavesel_seq ( CMD_WRITE_OPERATION , slave_addr ) ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Waits for an acknowledgement                                                                                                        */
     /*****************************************************************************************************************************************/
     ret = wait_ack() ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the memory address.                                                                                                            */
     /*****************************************************************************************************************************************/
     buf.d_int = rom_addr ;

     ret = set_data_seq ( buf.d_byte[0]) ;                                /* 1st Memory address                              */
          if (ret !=0) { goto exit ;}

     ret = set_data_seq ( buf.d_byte[1]) ;                                /* 2nd Memory address                              */
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Issues the stop condition to bring SDA to high.                                                                                     */
     /*****************************************************************************************************************************************/
     ret = set_end_proc ( ) ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the start condition again.                                                                                                     */
     /*****************************************************************************************************************************************/
     ret = set_start_condition() ;                                        /* Sets the start condition                        */
          if (ret !=0) { goto exit ;}
```

```
     /*****************************************************************************************************************************/
     /*   Sets the device address word (read).                                                                                    */
     /*****************************************************************************************************************************/
     ret = set_slavesel_seq ( DATA_READ_OPERATION , slave_addr ) ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************/
     /*   Switches to the master receive mode.                                                                                    */
     /*****************************************************************************************************************************/
     ret = set_master_rcv_mode () ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************/
     /*   Carries out a dummy read at the start of data reading.                                                                  */
     /*****************************************************************************************************************************/
     start_read_seq ( SINGLE_BYTE_READ ) ;

     /*****************************************************************************************************************************/
     /*   Issues the stop condition after data (1 byte) has been read.                                                            */
     /*****************************************************************************************************************************/
     ret = get_end_data_seq ( &buf.d_byte[0] ) ;
          if (ret !=0) { goto exit ;}

     *rom_data = buf.d_byte[0] ;

     return (ret);

exit :
     /*****************************************************************************************************************************/
     /*   Resets the I2C and issues the stop condition if an error occurs                                                         */
     /*****************************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x02 ;                                        /* Resets I2C control                                   */
     IIC2.ICCR2.BYTE     = 0x00 ;                                        /* Sets the stop condition                              */
     return (ret);

}
/*****************************************************************************************************************************/
/*   1. Module name : com_i2c_eeprom_write                                                                                   */
/*   2. Function overview : Writes 1-byte data to I2CEEPROM.                                                                 */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                        */
/*                             000       2002.12.14              Ueda                    New                                */
/*****************************************************************************************************************************/
unsigned int com_i2c_eeprom_write ( unsigned char slave_addr , unsigned int rom_addr , unsigned char rom_data )
{
     int ret ;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;


     ret = NORMAL_END ;

     /*****************************************************************************************************************************/
     /*   Initializes the I2C bus.                                                                                                */
     /*****************************************************************************************************************************/
     set_i2c_init () ;

     /*****************************************************************************************************************************/
     /*   Sets the start condition.                                                                                               */
     /*****************************************************************************************************************************/
     ret = set_start_condition() ;                                       /* Sets the start condition.                            */
          if (ret !=0) { goto exit ;}
```

```
    /*****************************************************************************************************************************/
    /*   Sets the device address word (write).                                                                                   */
    /*****************************************************************************************************************************/
    ret = set_slavesel_seq ( CMD_WRITE_OPERATION , slave_addr ) ;
         if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Waits for an acknowledgement.                                                                                           */
    /*****************************************************************************************************************************/
    ret = wait_ack() ;
         if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Sets the memory address.                                                                                                */
    /*****************************************************************************************************************************/
    buf.d_int = rom_addr ;

    ret = set_data_seq ( buf.d_byte[0]) ;                              /* 1st Memory address                                   */
         if (ret !=0) { goto exit ;}

    ret = set_data_seq ( buf.d_byte[1]) ;                              /* 2nd Memory address                                   */
         if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Sets the write data.                                                                                                    */
    /*****************************************************************************************************************************/
    ret = set_data_seq ( rom_data) ;
         if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Issues the stop condition.                                                                                              */
    /*****************************************************************************************************************************/
    ret = set_end_proc ( ) ;
         if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Check the end of write.                                                                                                 */
    /*****************************************************************************************************************************/
    ret = wait_write_end ( slave_addr ) ;
         if (ret !=0) { goto exit ;}
         /* ##(program note)############################################################################################### */
         /* ## The I2CEEPROM starts a write operation after receiving the stop condition.                                ## */
         /* ## Since the write operation takes a maximum of 15 ms,                                                        ## */
         /* ## the I2CEEPROM checks write completion using the Acknowledge Pooling method.                               ## */
         /* ############################################################################################################### */

    return (ret);

exit :
    /*****************************************************************************************************************************/
    /*   Resets the I2C and issues the stop condition if an error occurs                                                         */
    /*****************************************************************************************************************************/
    IIC2.ICCR2.BYTE     = 0x02 ;                                       /* Resets I2C control                                   */
    IIC2.ICCR2.BYTE     = 0x00 ;                                       /* Sets the stop condition                              */
    return (ret);

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : com_i2c_eeprom_seq_read                                                                                            */
/*   2. Function overview : Reads the specified length of data from I2CEEPROM.                                                           */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                                     */
/*                             000        2002.12.14             Ueda                    New                                           */
/*****************************************************************************************************************************************/
unsigned int com_i2c_eeprom_seq_read
               ( unsigned char slave_addr , unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data )
{
     int ret , i  ;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;


     ret = NORMAL_END ;

     /*****************************************************************************************************************************************/
     /*   Initializes the I2C bus                                                                                                             */
     /*****************************************************************************************************************************************/
     set_i2c_init () ;

     /*****************************************************************************************************************************************/
     /*   Sets the start condition                                                                                                            */
     /*****************************************************************************************************************************************/
     ret = set_start_condition() ;                                          /* Sets the start condition                          */
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the device address word (write).                                                                                               */
     /*****************************************************************************************************************************************/
     ret = set_slavesel_seq ( CMD_WRITE_OPERATION , slave_addr ) ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Waits for an acknowledgement                                                                                                        */
     /*****************************************************************************************************************************************/
     ret = wait_ack() ;
     if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the memory address.                                                                                                            */
     /*****************************************************************************************************************************************/
     buf.d_int = rom_addr ;

     ret = set_data_seq ( buf.d_byte[0]) ;                                  /* 1st Memory address                                */
          if (ret !=0) { goto exit ;}

     ret = set_data_seq ( buf.d_byte[1]) ;                                  /* 2nd Memory address                                */
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Issues the stop condition to bring SDA to high.                                                                                     */
     /*****************************************************************************************************************************************/
     ret = set_end_proc ( ) ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the start condition again.                                                                                                     */
     /*****************************************************************************************************************************************/
     ret = set_start_condition() ;                                          /* Sets the start condition                          */
          if (ret !=0) { goto exit ;}
```

```
    /*****************************************************************************************************************************/
    /*   Sets the device address word (read).                                                                                    */
    /*****************************************************************************************************************************/
    ret = set_slavesel_seq ( DATA_READ_OPERATION , slave_addr ) ;
        if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Switches to the master receive mode.                                                                                    */
    /*****************************************************************************************************************************/
    ret = set_master_rcv_mode () ;
        if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Carries out a dummy read at the start of data reading.                                                                  */
    /*****************************************************************************************************************************/
    start_read_seq ( MULTI_BYTE_READ ) ;

    /*****************************************************************************************************************************/
    /*   Reads data continuously.                                                                                                */
    /*****************************************************************************************************************************/
    for (i=0; i< (rom_length-1) ; i++){
        ret = get_data_seq ( &buf.d_byte[0] ) ;
            if (ret !=0) { goto exit ;}

            *rom_data = buf.d_byte[0] ;
            *rom_data ++ ;
    }

    /*****************************************************************************************************************************/
    /*   Specifies settings before reading the last data                                                                         */
    /*****************************************************************************************************************************/
    start_read_seq ( MULTI_FINAL_BYTE_READ ) ;

    /*****************************************************************************************************************************/
    /*   Issues the stop condition after the last data (1 byte) has been read                                                    */
    /*****************************************************************************************************************************/
    ret = get_end_data_seq ( &buf.d_byte[0] ) ;
        if (ret !=0) { goto exit ;}

        *rom_data = buf.d_byte[0] ;

        return (ret);

exit :
    /*****************************************************************************************************************************/
    /*   Resets the I2C and issues the stop condition if an error occurs                                                         */
    /*****************************************************************************************************************************/
    IIC2.ICCR2.BYTE     = 0x02 ;                                    /* Resets I2C control                                   */
    IIC2.ICCR2.BYTE     = 0x00 ;                                    /* Sets the stop condition                              */
    return (ret);

}
```

```
/*****************************************************************************************************************************************/
/*   1. Module name : com_i2c_eeprom_write                                                                                               */
/*   2. Function overview : Writes data to I2CEEPROM continuously.                                                                       */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                                     */
/*                             000        2002.12.14             Ueda                    New                                           */
/*****************************************************************************************************************************************/
unsigned int com_i2c_eeprom_page_write
               ( unsigned char slave_addr , unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data )
{
     int ret , i ;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;


     ret = NORMAL_END ;

     /*****************************************************************************************************************************************/
     /*   Initializes the I2C bus.                                                                                                            */
     /*****************************************************************************************************************************************/
     set_i2c_init () ;

     /*****************************************************************************************************************************************/
     /*   Sets the start condition.                                                                                                           */
     /*****************************************************************************************************************************************/
     ret = set_start_condition() ;                                          /* Sets the start condition                          */
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the device address word (write).                                                                                               */
     /*****************************************************************************************************************************************/
     ret = set_slavesel_seq ( CMD_WRITE_OPERATION , slave_addr ) ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Waits for an acknowledgement.                                                                                                       */
     /*****************************************************************************************************************************************/
     ret = wait_ack() ;
          if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Sets the memory address.                                                                                                            */
     /*****************************************************************************************************************************************/
     buf.d_int = rom_addr ;

     ret = set_data_seq ( buf.d_byte[0]) ;                                  /* 1st Memory address                                */
          if (ret !=0) { goto exit ;}

    ret = set_data_seq ( buf.d_byte[1]) ;                                   /* 2nd Memory address                                */
     if (ret !=0) { goto exit ;}

     /*****************************************************************************************************************************************/
     /*   Writes data continuously.                                                                                                           */
     /*****************************************************************************************************************************************/
     for (i=0; i< rom_length ; i++){
          buf.d_byte[0] = *rom_data  ;
          ret = set_data_seq ( buf.d_byte[0] ) ;
               if (ret !=0) { goto exit ;}
               *rom_data ++ ;
     }
```

```
    /*****************************************************************************************************************************/
    /*   Issues the stop condition                                                                                               */
    /*****************************************************************************************************************************/
    ret = set_end_proc ( ) ;
        if (ret !=0) { goto exit ;}

    /*****************************************************************************************************************************/
    /*   Checks write completion.                                                                                                */
    /*****************************************************************************************************************************/
    ret = wait_write_end ( slave_addr ) ;
        if (ret !=0) { goto exit ;}
        /* ##(program note)############################################################################################### */
        /* ## The I2CEEPROM starts the write operation after receiving the stop condition.                                ## */
        /* ## Since the write operation takes a maximum of 15 ms,                                                          ## */
        /* ## the I2CEEPROM checks write completion using the Acknowledge Pooling method.                                 ## */
        /* ############################################################################################################### */

    return (ret);

exit :
    /*****************************************************************************************************************************/
    /*   Resets the I2C and issues the stop conditions if an error occurs                                                        */
    /*****************************************************************************************************************************/
    IIC2.ICCR2.BYTE     = 0x02 ;                                    /* Resets I2C control                                    */
    IIC2.ICCR2.BYTE     = 0x00 ;                                    /* Sets the stop condition                               */
    return (ret);

}
```

```
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4. Sample Program 1-D TimerZ Processing ---------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4.1 Addition of reset vectors -------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/*   Set the jump destination to h8_timerz.                                                                                 */

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4.2 Common variable definitions for TimerZ ------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
     struct    {
          int counter;                                          /* 100 ms counter                                 */
          int wait_10ms;                                        /* Sets a wait time of 10 ms                      */
          int wait_100ms;                                       /* Sets the wait time in 100 ms units (common)    */
          int wait_100ms_scan;                                  /* Sets the wait time in 100 ms units (for I2C)   */
     }com_timer;

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4.3 TimerZ initial settings ---------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
     /* ###################################################################################################################### */
     /* ###################################################################################################################### */
     /*                                                                                                                        */
     /*        Sets TimerZ                                                                                                     */
     /*                                                                                                                        */
     /* ###################################################################################################################### */
     /* ###################################################################################################################### */
     /************************************************************************************************************************/
     /*   Sets TimerZ initial settings                                                                                         */
     /************************************************************************************************************************/
     TZ.TSTR.BYTE = 0x00 ;
     TZ.TMDR.BYTE = 0x00 ;
     TZ.TPMR.BYTE = 0x00 ;
     TZ.TFCR.BYTE = 0x00 ;
     TZ.TOER.BYTE = 0xFF ;
     TZ.TOCR.BYTE = 0x00 ;

     TZ0.TCR.BYTE   = 0x23 ;
                                                           /* CCLR[2:0] = 001 Clears the counter               */
                                                           /* when a GRA compare match occurs.                 */

                                                           /* CKEG[1:0] = 00 Counts up at the rising edges     */
                                                           /* TPSC[2:0] = 011                                  */
                                                           /* Counts using internal clock ϕ/8                  */
     TZ0.TIORA.BYTE = 0x00 ;
                                                           /* IOA[2:0] = 000                                   */
                                                           /* GRA is used as an output compare register        */
     TZ0.TIER.BYTE  = 0x01 ;
                                                           /* IMIEA = 1 Enables IMFA                           */

     TZ0.GRA        = 20000 ;                              /* Issues an interrupt every 10 ms                  */
          /* ##(program note)################################################################################################ */
          /* ## The set values differ depending on the operating frequency of the microcomputer.                          ## */
          /* ## Refer to the H8/3687 Hardware Manual.                                                                      ## */
          /* ###################################################################################################################### */

     TZ0.TCNT       = 0 ;                                  /* Clears the timer counter                         */
```

```
     /*****************************************************************************************************************************/
     /*   Starts TimerZ                                                                                                           */
     /*****************************************************************************************************************************/
     TZ.TSTR.BYTE   = 0x01 ;                                          /* timer start                                            */
                                                                      /* STR0 = 1 Starts TCNT_0                                 */


/* ------------------------------------------------------------------------------------------------------------------------------ */
/* 4.4 TimerZ interrupt processing ---------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/*****************************************************************************************************************************/
/*   1. Module name : h8_TimerZ                                                                                                  */
/*   2. Function overview: Interval timer processing every 10 ms                                                                 */
/*   3. History of revisions : REV  Date created/revised    Created/revised by     Revision contents                           */
/*                             000        2002.02.11             Ueda                    New                                    */
/*****************************************************************************************************************************/
#pragma interrupt( h8_timerz )
void h8_timerz( void )
{

     /*****************************************************************************************************************************/
     /*   Clears the source                                                                                                       */
     /*****************************************************************************************************************************/
     com_global.dummy = TZ0.TSR.BYTE;                                 /* dummy read                                             */

     TZ0.TSR.BIT.IMFA = 0;                                            /* IMFA clear                                             */

     /*****************************************************************************************************************************/
     /*   -1 in units of 10 ms                                                                                                    */
     /*****************************************************************************************************************************/
     if( com_timer.wait_10ms>0 )
          com_timer.wait_10ms --;

     /*****************************************************************************************************************************/
     /*   Increments the counter                                                                                                  */
     /*****************************************************************************************************************************/
     com_timer.counter++;
     if( com_timer.counter >= 10 ){
          /*************************************************************************************************************************/
          /*   -1 in units of 100 ms                                                                                               */
          /*************************************************************************************************************************/
          if( com_timer.wait_100ms>0 )
               com_timer.wait_100ms --;
          if( com_timer.wait_100ms_scan>0 )
               com_timer.wait_100ms_scan --;

          com_timer.counter = 0;
     }

}
```

4. Reference Documents

- H8/3687 Group Hardware Manual (published by Renesas Technology Corp.)
- HN58X2464FPIAG Data Sheet (published by Renesas Technology Corp.)
- I^2C Bus Usage (published by Phillips)

Revision Record

		Description	
Rev.	Date	Page	Summary
1.00	Sep.29.03	—	First edition issued

Keep safety first in your circuit designs!

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

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
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 Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
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 Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation 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 Corporation for further details on these materials or the products contained therein.