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

Master-Slave Communication using I²C Interface (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, a timer, I²C bus interface, serial communication interface, and 10-bit A/D converter. These devices can be utilized as embedded microcomputers in sophisticated control systems.

These H8/300H Series H8/3687- Application Notes consist of a "Basic Edition" which describes operation examples when using the individual 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

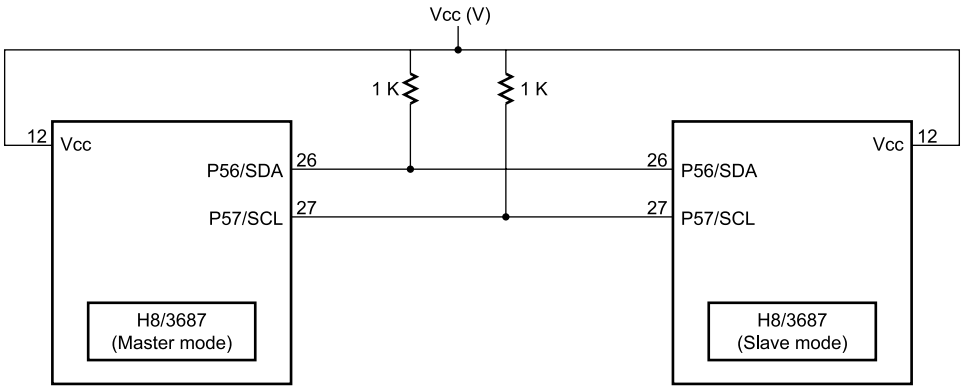
Contents

1. Specifications

Communication between microcomputers is carried out via the I²C interface of the H8/3687.

2. Configuration

Figure 2.1 shows a diagram of connection between microcomputers.



Note: Operating frequency of the H8/3687 is 16 MHz with the power supply voltage of 5 V.

Figure 2.1 Diagram of connection between microcomputers.

3. Sample Programs

3.1 Functions

The H8 microcomputer in master mode transmits four bytes of data, which is received by the H8 microcomputer in slave mode. The slave-mode microcomputer then returns the same four bytes of data to the master-mode microcomputer.

3.2 Embedding the Sample Programs

1. Sample program 2-A
   Incorporate #define directives.
   (For the microcomputer which is to operate in slave mode, #define SLAVE_MODE should be included.)
2. Sample program 2-B
   Incorporate prototype declarations.
3. Sample program 2-C
   Incorporate the source program.
4. Sample program 2-D (interrupt processing for slave mode)
   — Add the reset vector for I2C.
   — Add I2C setting initialization.
   — Add I2C interrupt processing.

3.3 Modification to the Sample Programs

Without modifications to the sample program, the system may not run. The sample programs should be modified to be suited to your program and system environment.

1. You can use the sample programs without further changes if you use the I/O register definition file available free of charge from the following Renesas web site.
   http://www.renesas.com/eng/products/mpumcu/tool/crosstool/iodef/index.html
   When creating definitions by yourself, you may modify the I/O register structures in the sample program as appropriate.
2. The sample program is designed so that timer Z is configured to start every 10 ms with timeout setting of 5 seconds in order to give timing of monitoring the state of the I2C interface. The timer processing may be modified according to your needs, 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 2-E
      - Add the timer Z reset vector.
      - Add com_timer as a common variable.
      - Add timer Z initial setting processing.
      - (The GRA setting should be changed according to the operating frequency of the microcomputer being used, so that the timer Z interrupt occurs every 10 ms. For setting values, refer to the H8/3664 Hardware Manual; for the location of modification, refer to the program notes in the sample program.)
      - Add timer Z interrupt processing.
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 of modification. In this sample program, the transfer rate is set to 200 kbps.

3.4 Method of use

Four bytes of data are transmitted from the master-mode H8 microcomputer, and after the slave-mode H8 microcomputer receives the data, it returns the same 4 bytes of data to the master-mode device. The following subroutine is executed by the master-mode device.

1. Transmit 4 bytes of data from the master mode to the slave-mode device

```
unsigned int com_i2c_master_send
(unsigned char slave_addr , unsigned int data_length , unsigned char *send_data)
```

Argument	Explanation
slave_addr	Specifies the slave-mode device address. In the sample program, this setting is 0x80.
data_length	Specifies the length of data for transmission In the sample program, this setting is 0x4.
*send_data	Specifies the address at which to store data for transmission.

Return value	Explanation
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 (halt condition detection timeout)

Example of use:

```
int ret ;
unsigned char slave_addr ;
unsigned int data_length ;
unsigned char send_data[256] ;
ret = com_i2c_master_send (slave_addr , data_length , &send_data[0] )
```

2. The 4 bytes of data returned by the slave-mode device are received by the master-mode device.

```
unsigned int com_i2c_master_recive
(unsigned char slave_addr , unsigned int data_length , unsigned char *recive_data)
```

Argument	Explanation
slave_addr	Specifies the slave-mode device address. In the sample program, this setting is 0x80.
data_length	Specifies the receive data length. In the sample program, this setting is 0x4.
*recive_data	Specifies the address where the received data is stored.

Return value	Explanation
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 (halt condition detection timeout)

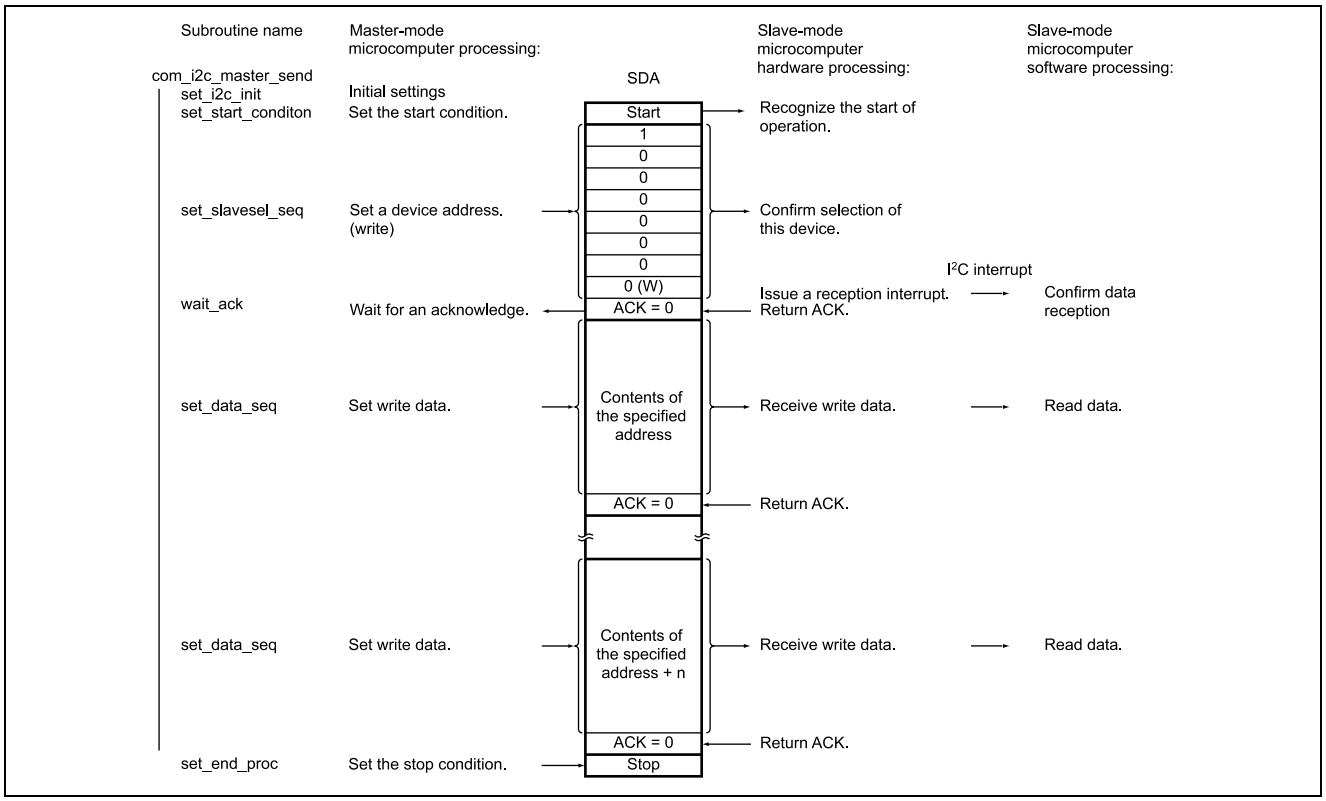
Example of use:

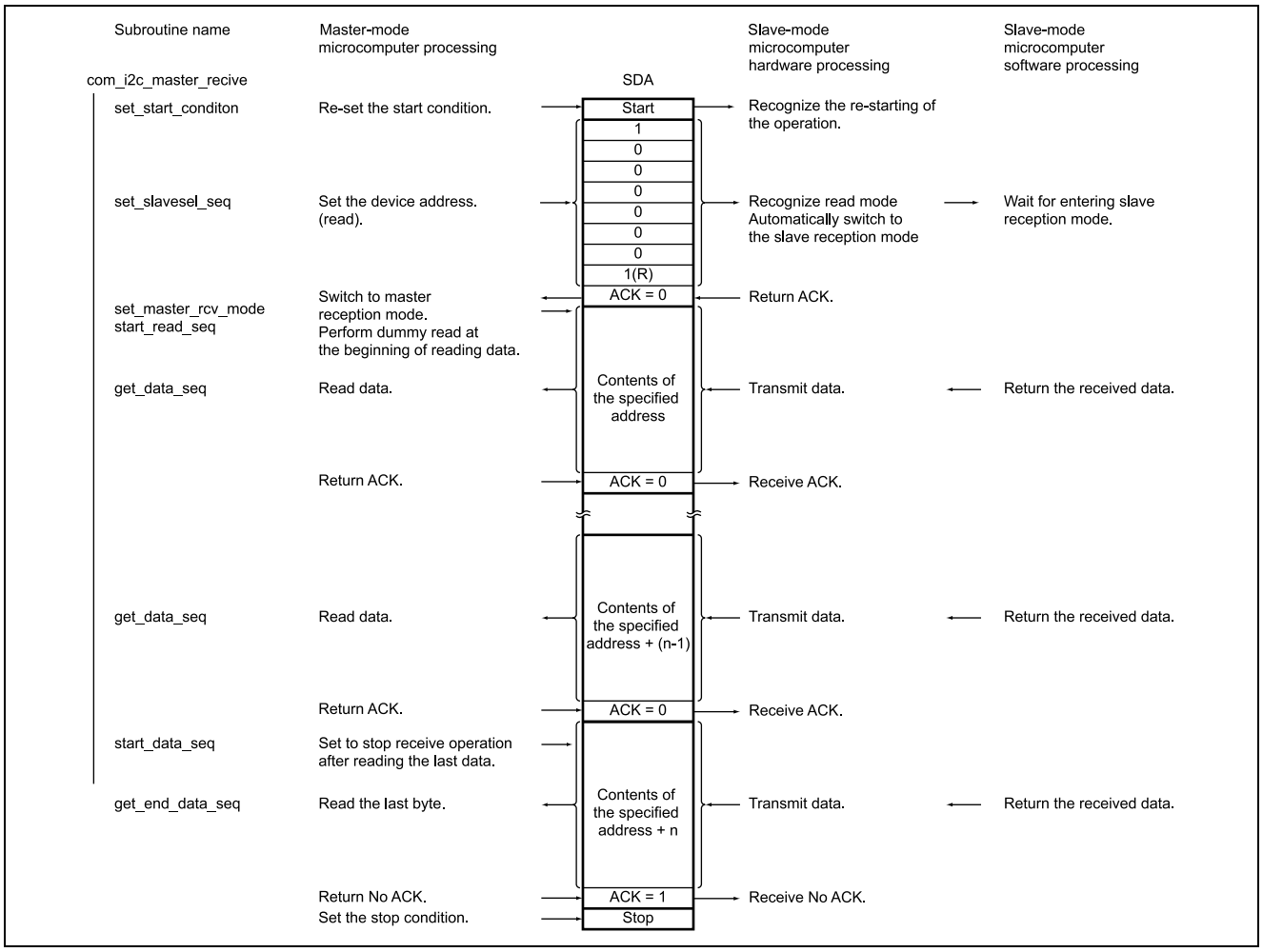
```
int ret ;
unsigned char slave_addr ;
unsigned int data_length ;
unsigned char recive_data[256] ;
ret = com_i2c_master_recive (slave_addr , data_length , &recive_data[0] )
```

3.5 Description of operation

The operation is as described below. The following figure depicts the operation of the master-mode and the slave-mode H8 microcomputers with respect to SDA data flow.

1. Four bytes of data is transmitted from the master-mode H8 microcomputer, and after the slave-mode H8 microcomputer receives the data, it returns the same 4 bytes of data to the master-mode device.




Subroutine name
Master-mode microcomputer processing
Slave-mode microcomputer hardware processing
Slave-mode microcomputer software processing
com_i2c_master_recive
SDA
set_start_conditon
Re-set the start condition.
Start
Recognize the re-starting of the operation.
1
0
0
0
0
0
0
1(R)
set_slavesel_seq
Set the device address. (read).
Recognize read mode Automatically switch to the slave reception mode
Wait for entering slave reception mode.
ACK = 0
Return ACK.
set_master_rcv_mode
start_read_seq
Switch to master reception mode. Perform dummy read at the beginning of reading data.
get_data_seq
Read data.
Contents of the specified address
Transmit data.
Return the received data.
Return ACK.
ACK = 0
Receive ACK.
get_data_seq
Read data.
Contents of the specified address + (n-1)
Transmit data.
Return the received data.
Return ACK.
ACK = 0
Receive ACK.
start_data_seq
Set to stop receive operation after reading the last data.
get_end_data_seq
Read the last byte.
Contents of the specified address + n
Transmit data.
Return the received data.
Return No ACK.
Set the stop condition.
ACK = 1
Receive No ACK.
Stop

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. I²C-related registers

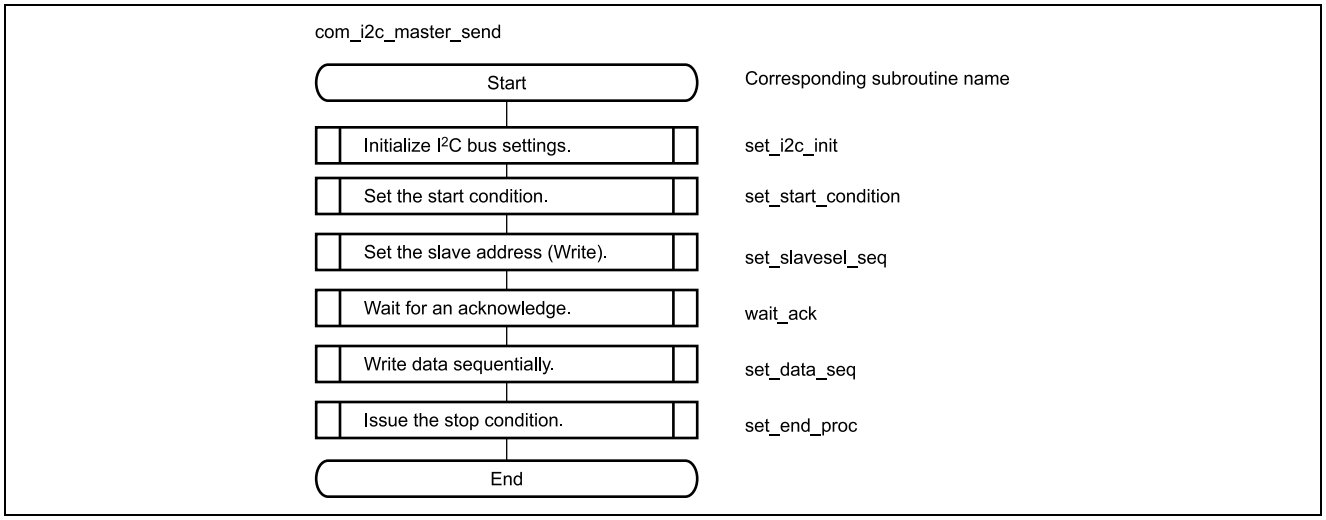
Name	Summary
I²C bus control register 1 (ICCR1)	Starts or stops operation of the I²C bus interface 2, controls transmission/reception, and selects master/slave mode, transmission/reception, and master mode transfer clock frequency.
I²C bus control register 2 (ICCR2)	Issues start/stop conditions, operates the SDA pin, monitors the SCL pin, and controls resets for I²C bus interface 2 control unit.
I²C bus mode register (ICMR)	Selects the MSB first or LSB first, controls master mode waits, and sets the number of transfer bits.
Bus interrupt enable register (ICIER)	Enables individual interrupts, validates/invalidates acknowledge, sets transmit acknowledge, and checks receive acknowledge.
I²C bus status register (ICSR)	Used for checking of interrupt request flags and the statuses.
Slave address register (SAR)	Sets the slave address and transfer format.
I²C bus transmit data register (ICDRT)	8-bit readable/writable register which stores data for transmission.
I²C bus receive data register (ICDRR)	8-bit register which stores received data.

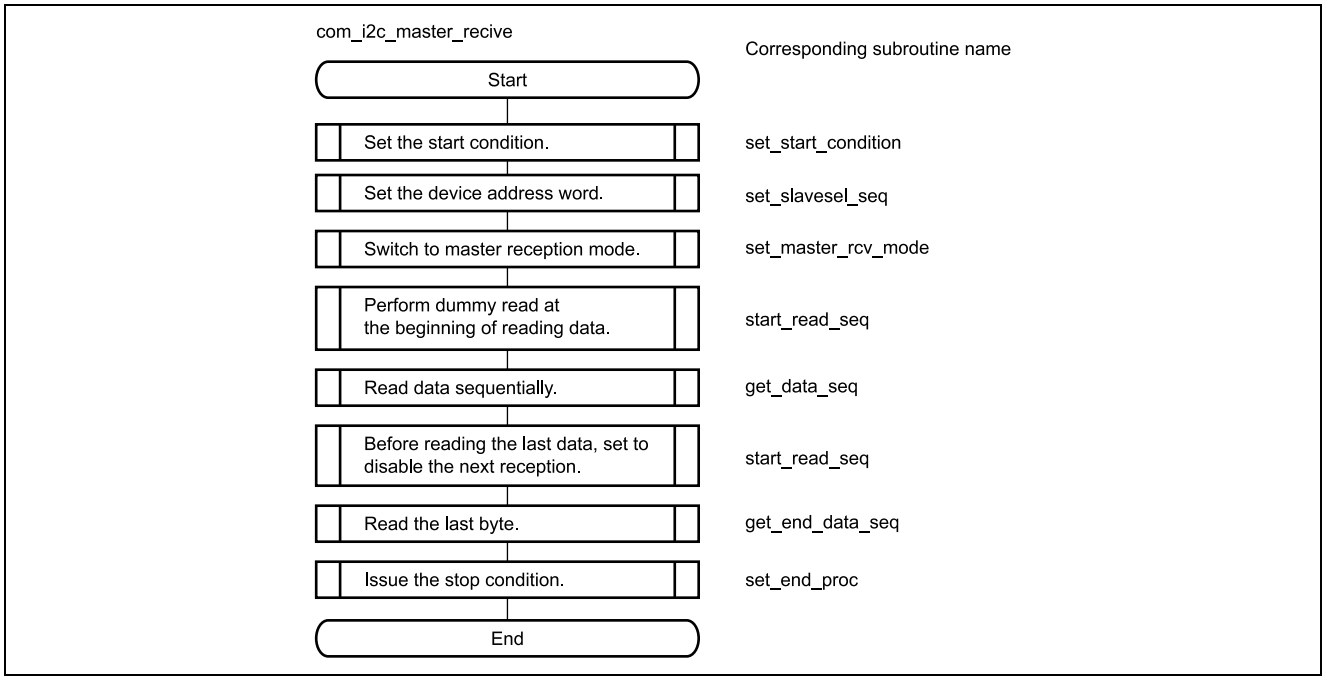
2. Timer Z-related registers
   Timer Z has various functions, but in the sample program it uses the compare-match function with the GRA register to generate an interrupt every 10 ms.

Name	Summary
Timer start register (TSTR)	Starts or stops TCNT operation.
Timer mode register W (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 operation modes and output level settings. Not used in this sample program.
Timer output master enable register (TOER)	Enables/disables outputs on channels 0 and 1.
Timer output control register (TOCR)	Selects the initial output level that is to be output until the first compare-match is generated.
Timer counter (TCNT)	16-bit readable/writable register which counts up with the input clock.
General registers A, B, C, D (GRA, GRB, GRC, GRD)	General registers are16-bit readable/writable registers. Each channel has four general registers, therefore, total of eight registers are provided. These registers can be used either as output-compare registers or input-capture registers, according to the TIORA and TIORC settings.
Timer control register (TCR)	Selects the TCNT counter's input clock, edge for an external clock (when an external clock is selected), and counter clearing 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 occurrence of TCNT overflow/underflow and compare-match or input-capture with GRA/GRB/GRC/GRD.
Timer interrupt enable register (TIER)	Enables/disables overflow interrupt requests and 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.

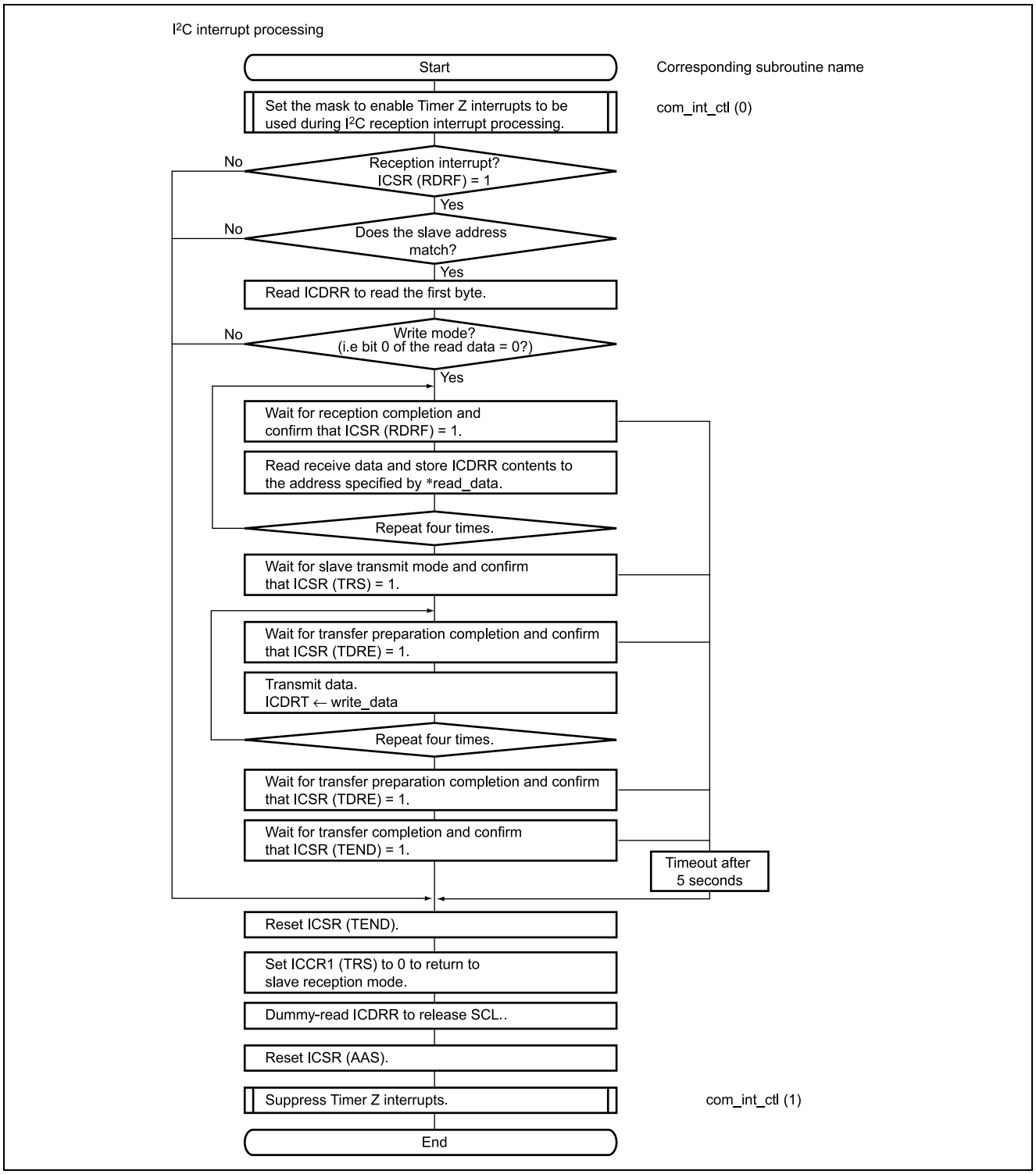
3.7 Flowcharts

1. Master mode H8 microcomputer processing

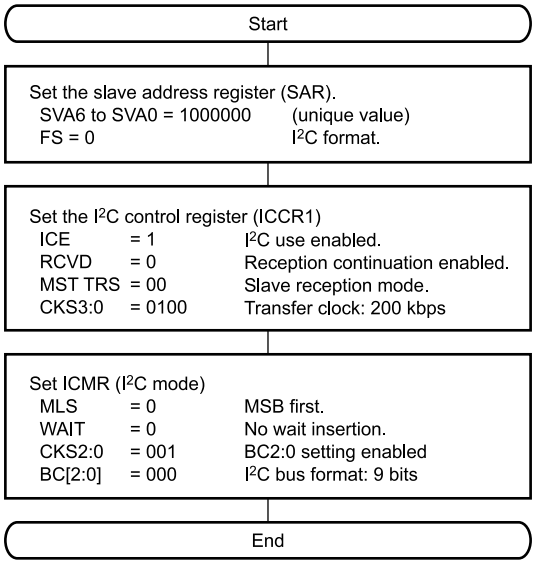




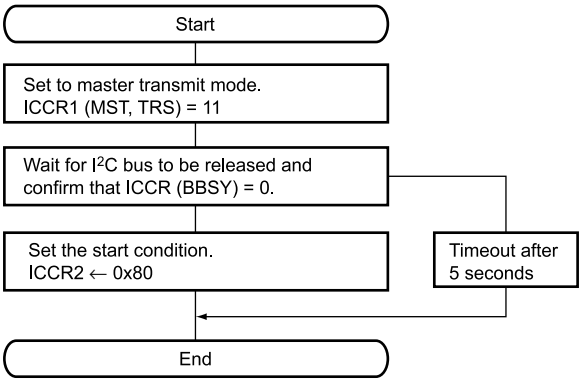
2. Slave mode H8 microcomputer processing



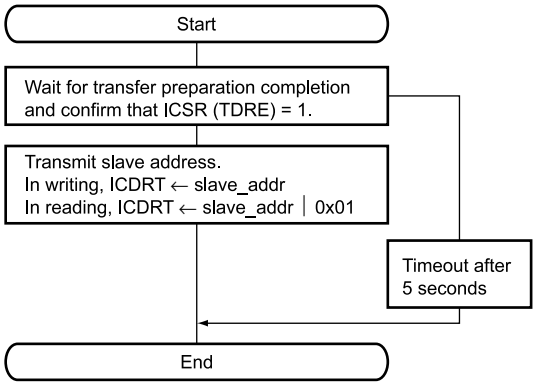
set_i2C_init
: Initializes I²C bus settings.



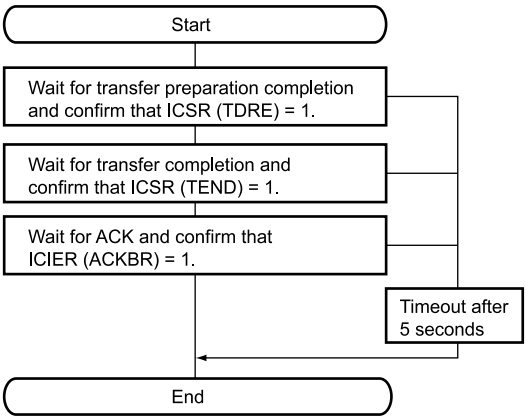
set_start_condition
: Sets the start condition.



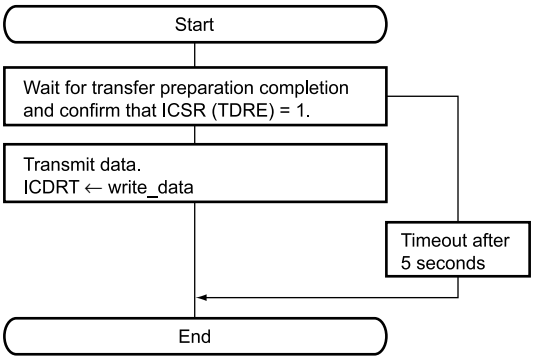
set_slavesel_seq(unsigned char mode,unsigned char slave_addr)
: Executes slave selection processing.
Mode: Write or read
0: Write, 1: Read
slave_addr: EEPROM device address

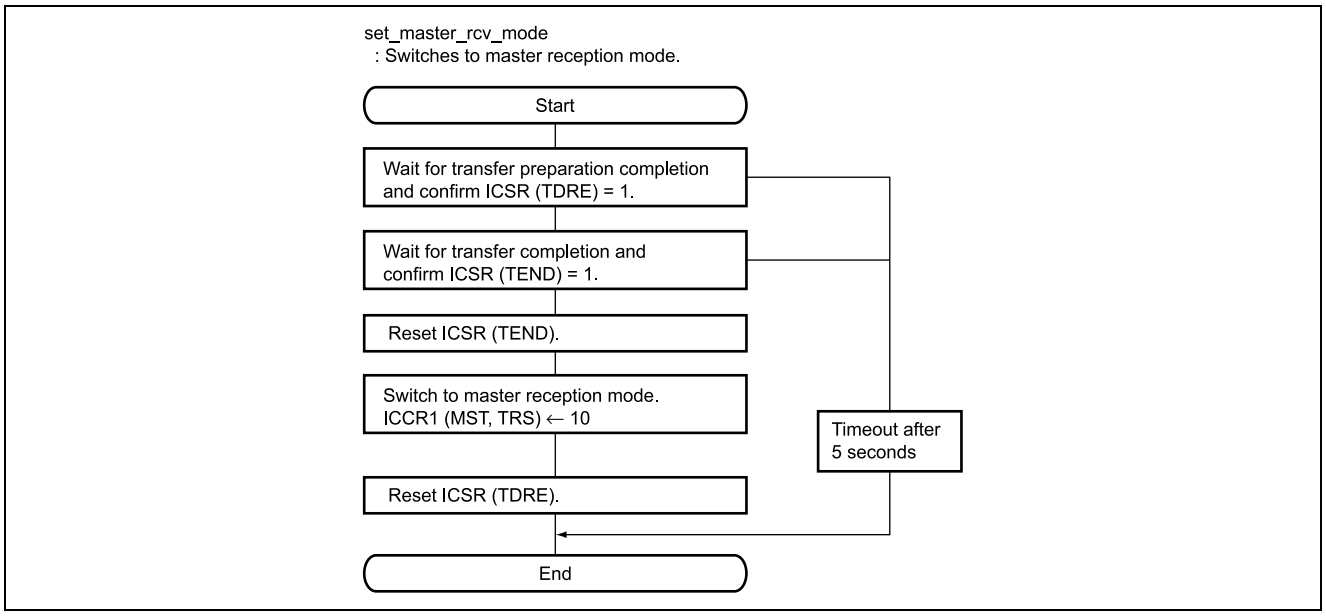


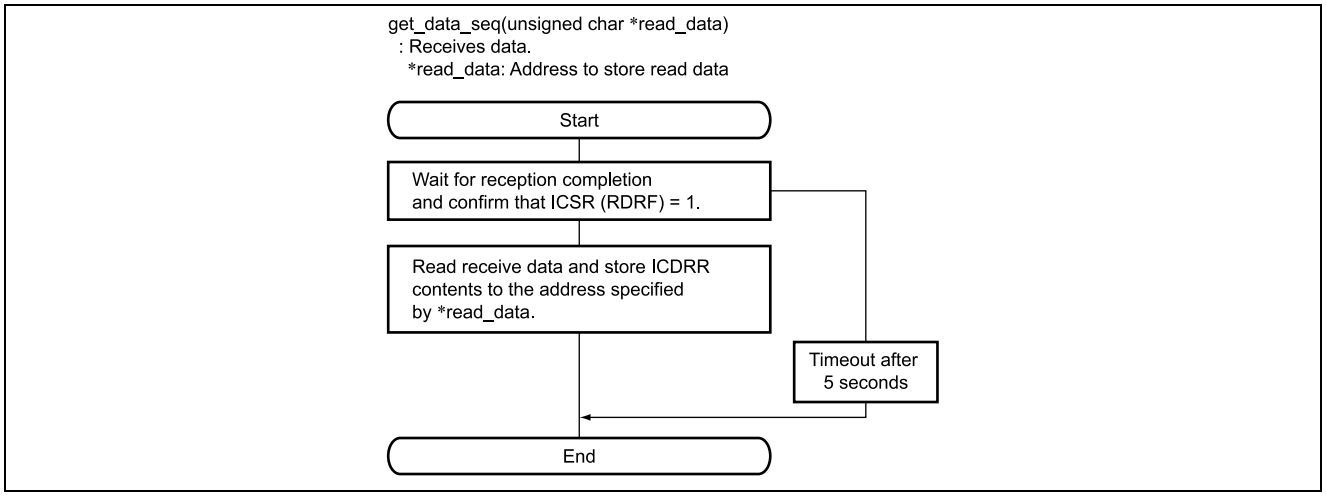
wait_ack
: Waits for an acknowledge.



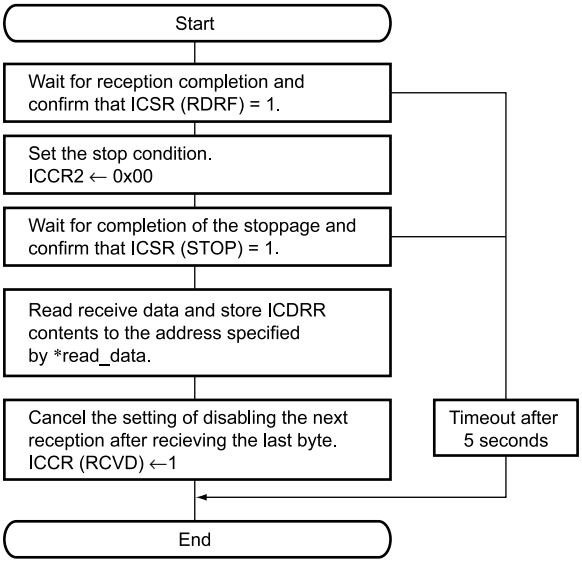
set_data_seq(unsigned char write_data)
: Sets data for transmission.
write_data: Data to be transmitted



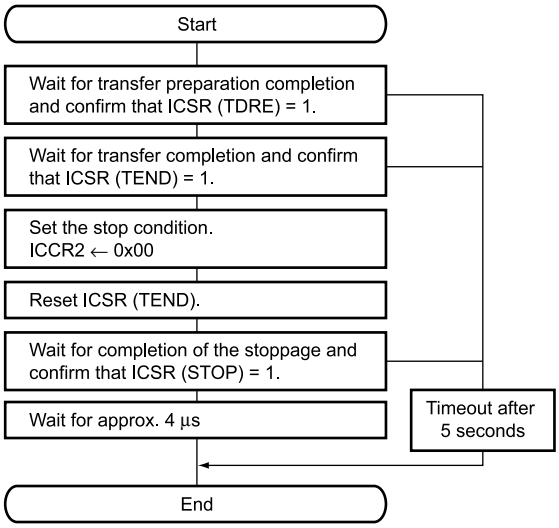

set_master_rcv_mode
: Switches to master reception mode.
Start
Wait for transfer preparation completion and confirm ICSR (TDRE) = 1.
Wait for transfer completion and confirm ICSR (TEND) = 1.
Reset ICSR (TEND).
Switch to master reception mode. ICCR1 (MST, TRS) ← 10
Reset ICSR (TDRE).
End
Timeout after 5 seconds

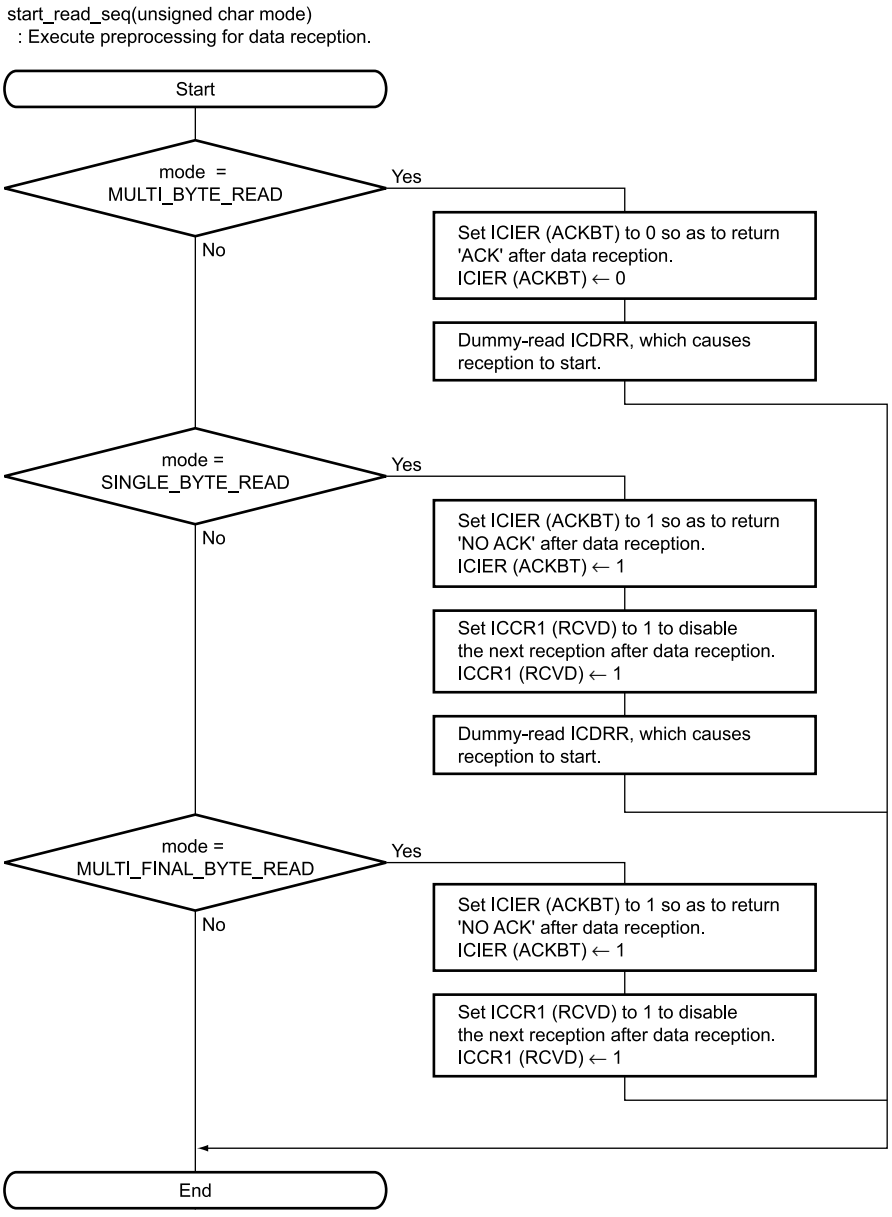

get_data_seq(unsigned char *read_data)
: Receives data.
*read_data: Address to store read data
Start
Wait for reception completion and confirm that ICSR (RDRF) = 1.
Read receive data and store ICDRR contents to the address specified by *read_data.
End
Timeout after 5 seconds

get_end_data_seq(unsigned char *read_data)
: Receives the last data.
*read_data: Address to store read data



set_end_proc
: Executes end sequence.




start_read_seq(unsigned char mode)
: Execute preprocessing for data reception.
Start
mode = MULTI_BYTE_READ
Yes
No
Set ICIER (ACKBT) to 0 so as to return 'ACK' after data reception.
ICIER (ACKBT) ← 0
Dummy-read ICDRR, which causes reception to start.
mode = SINGLE_BYTE_READ
Yes
No
Set ICIER (ACKBT) to 1 so as to return 'NO ACK' after data reception.
ICIER (ACKBT) ← 1
Set ICCR1 (RCVD) to 1 to disable the next reception after data reception.
ICCR1 (RCVD) ← 1
Dummy-read ICDRR, which causes reception to start.
mode = MULTI_FINAL_BYTE_READ
Yes
No
Set ICIER (ACKBT) to 1 so as to return 'NO ACK' after data reception.
ICIER (ACKBT) ← 1
Set ICCR1 (RCVD) to 1 to disable the next reception after data reception.
ICCR1 (RCVD) ← 1
End

3.8 Program Listing

```
/* ---------------------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------------------- */
/* 1. Sample Program 2-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

/*****************************************************************************************************************************************/
/*   I2CEEPROM access error codes (codes 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

/*****************************************************************************************************************************************/
/*   ↓ This should only be defined for the slave mode device.                                                                            */
/*****************************************************************************************************************************************/
/*   slave mode                                                                                                                          */
/*****************************************************************************************************************************************/
#define SLAVE_MODE
```

```
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/* 2. Sample program 2-B  Prototype declarations ------------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
/*   I2C BUS access processing                                                                                                                       */
/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
void com_delay( int delaytime ) ;
void com_int_ctl (unsigned char kind) ;
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 () ;

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_master_send ( unsigned char slave_addr , unsigned int data_length , unsigned char *send_data ) ;
unsigned int com_i2c_master_recive ( unsigned char slave_addr ,  unsigned int data_length , unsigned char *recive_data ) ;

/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/* 3. Sample program 2-C  Source code ------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------------------------------ */

/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
/*                                                                                                                                                   */
/*                                I2C EEPROM control                                                                                                 */
/*                                                                                                                                                   */
/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
/*****************************************************************************************************************************************************/
/*   1. Module name: com_int_ctl                                                                                                                     */
/*   2. Function overview: Clears set_imask_ccr to 0 to enable Timer Z interrupts alone.                                                             */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                                                  */
/*                            000        2002.04.10              Ueda                  New                                                        */
/*****************************************************************************************************************************************************/
void com_int_ctl (unsigned char kind)
{

     if (kind == 0){
          /*****************************************************************************************************************************************/
          /*   Disables I2C reception interrupts                                                                                                   */
          /*****************************************************************************************************************************************/
          IIC2.ICIER.BIT.RIE  = 0 ;                                    /* Disables I2C reception interrupts                                    */
```

```
        /*****************************************************************************************************************************/
        /*   Enables Timer Z interrupts                                                                                              */
        /*****************************************************************************************************************************/
        TZ0.TIER.BIT.IMIEA  = 1 ;                                   /* timerZ IMFA  enable                                         */

        /*****************************************************************************************************************************/
        /*   Cancels interrupt disable                                                                                               */
        /*****************************************************************************************************************************/
        set_imask_ccr(0);                                           /* Enables interrupts                                          */
    }
    else{
        /*****************************************************************************************************************************/
        /*   Disable interrupts (reason: to prevent other interrupts during interrupt processing)                                    */
        /*****************************************************************************************************************************/
        set_imask_ccr(1);                                           /* Disables interrupts                                         */

        /*****************************************************************************************************************************/
        /*   Enables I2C reception interrupts                                                                                        */
        /*****************************************************************************************************************************/
        IIC2.ICIER.BIT.RIE  = 1 ;                                   /* Enables I2C reception interrupt                             */

        /*****************************************************************************************************************************/
        /*   Disables TimerZ interrupts                                                                                              */
        /*****************************************************************************************************************************/
        TZ0.TIER.BIT.IMIEA  = 0 ;                                   /* timerz IMFA  enable                                         */

    }
}

/*****************************************************************************************************************************/
/*   1. Module name: com_delay                                                                                               */
/*   2. Function overview: Delay of any desired time                                                                         */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                          */
/*                            000         2002.03.25            Ueda                    New                                 */
/*****************************************************************************************************************************/
void com_delay( int delaytime )
{
    register int i,a;

    for(i=0;i<delaytime;i++)
        a++;
}

/*****************************************************************************************************************************/
/*   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            Slave address register                                                                                   */
    /*        SVA6:0    = 1000000 (unique value)                                                                                 */
    /*        FS        = 0  I2C format                                                                                          */
    /*****************************************************************************************************************************/
        /* ##(program note)########################################################################################################## */
        /* ## SVA6:0 are used in the slave mode. They 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 frequency (ϕ/80, transfer rate: 200 kbps)                                          */
    /*****************************************************************************************************************************/

    IIC2.ICCR1.BYTE     = 0x84 ;
        /* ##(program note)############################################################################################### */
        /* ## Setting of CKS3:0 should be changed according to the required transfer rate.                              ## */
        /* ## For details, please 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){                                         /* If this remains 1 for 5 seconds,                    */
                                                                    /* exits with an error.                                */
            ret = I2C_BBSY_TOUT;                                    /* Abnormal end (timeout)                              */
            goto exit ;
        }

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

    }
```

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

     /*****************************************************************************************************************************/
     /*   Sets the start condition                                                                                                */
     /*****************************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x80 ;
          /* ##(program note)########################################################################################################## */
          /* ## Bits 7 and 6, which set the start condition, have to be set simultaneously, so they must be                  ## */
          /* ## written in byte units                                                                                         ## */
          /* ## Be aware 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 this remains 1 for 5 seconds,                      */
                                                                       /* exits with an error.                                  */
               ret = I2C_TDRE_TOUT;                                    /* Abnormal end (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                                                                                                            */
/*   2. 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 this remains 1 for 5 seconds,                     */
                                                                       /* exits with an error.                                 */
               ret = I2C_TDRE_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                     */
                                                                       /* exits with an error.                                 */
               ret = I2C_TEND_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                    */
                                                                /* exits with an error.                                */
            ret = I2C_ACKBR_TOUT;                               /* Abnormal end (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 this remains 1 for 5 seconds,                    */
                                                                /* exits with an error.                                */
            ret = I2C_TDRE_TOUT;                                /* Abnormal end (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 this remains 1 for 5 seconds,                     */
                                                                       /* exits with an error.                                 */
               ret = I2C_TDRE_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                     */
                                                                       /* exits with an error.                                 */
               ret = I2C_TEND_TOUT;                                    /* Abnormal end (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 by executing a dummy read                                                                         */
          /*****************************************************************************************************************************/
          dummy_data = IIC2.ICDRR ;
               /* ##(program note)############################################################################################ */
               /* ## Reception begins when a dummy read is carried out, and data is sent from the device synchronized.       ## */
               /* ## with the SCL                                                                                            ## */
               /* ## A low level signal is sent to the device synchronized with the ninth SCL, in response to the ICSR (ACKB)## */
               /* ## set to 0 previously.                                                                                    ## */
               /* ############################################################################################################ */

     }
     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 the current data reception                                                          */
          /*****************************************************************************************************************************/
          IIC2.ICCR1.BIT.RCVD = 1  ;

          /*****************************************************************************************************************************/
          /*   Initiates reception by executing a dummy read                                                                         */
          /*****************************************************************************************************************************/
          dummy_data = IIC2.ICDRR ;
```

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

        }

        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 the current 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 this remains 1 for 5 seconds,                 */
                                                                        /* exits with an error.                             */
                ret = I2C_RDRF_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                     */
                                                                       /* exits with an error.                                 */
               ret = I2C_RDRF_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                     */
                                                                       /* exits with an error.                                 */
               ret = I2C_STOP_TOUT;                                    /* Abnormal end (timeout)                               */
               goto exit ;
          }

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

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

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

exit :

     return (ret);
}

/*****************************************************************************************************************************/
/*   1. Module name: set_end_proc                                                                                            */
/*   2. Function overview: Executes an I2C end 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 this remains 1 for 5 seconds,                    */
                                                                       /* exits with an error.                                */
               ret = I2C_TDRE_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                    */
                                                                       /* exits with an error.                                */
               ret = I2C_TEND_TOUT;                                    /* Abnormal end (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 this remains 1 for 5 seconds,                      */
                                                                       /* exits with an error.                                  */
               ret = I2C_STOP_TOUT;                                    /* Abnormal end (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: com_i2c_master_recive                                                                                       */
/*   2. Function overview: Receives data of the specified length from the slave device                                           */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                          */
/*                             000          2002.12.14          Ueda                    New                                    */
/*****************************************************************************************************************************/
unsigned int com_i2c_master_recive ( unsigned char slave_addr , unsigned int data_length , unsigned char *recive_data )
{
     int ret , i  ;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;

     ret = NORMAL_END ;

     /*****************************************************************************************************************************/
     /*   Sets the start condition.                                                                                                */
     /*****************************************************************************************************************************/
    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 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< (data_length-1) ; i++){
        ret = get_data_seq ( &buf.d_byte[0] ) ;
            if (ret !=0) { goto exit ;}

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

    /*****************************************************************************************************************************************/
    /*   Makes 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 ;}

        *recive_data = buf.d_byte[0] ;

        return (ret);

exit :
    /*****************************************************************************************************************************************/
    /*   Resets the I2C control 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_master_send                                                                                                 */
/*   2. Function overview: Transmits data of the specified length from the master to a slave device                                      */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                                    */
/*                             000         2002.12.14             Ueda                    New                                          */
/*****************************************************************************************************************************************/
unsigned int com_i2c_master_send ( unsigned char slave_addr , unsigned int data_length , unsigned char *send_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 ;}
     /*****************************************************************************************************************************************/
     /*   Writes data continuously                                                                                                            */
     /*****************************************************************************************************************************************/
     for (i=0; i< data_length ; i++){
          buf.d_byte[0] = *send_data  ;
          ret = set_data_seq ( buf.d_byte[0] ) ;
               if (ret !=0) { goto exit ;}
               *send_data ++ ;
     }

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

     return (ret);

exit :
     /*****************************************************************************************************************************************/
     /*   Resets the I2C control 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);

}
```

```
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4 . Sample Program 2-D  Slave Mode Processing -------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4.1 Addition of the reset vector --------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/*   Set the jump destination to h8_i2c.                                                                                    */

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4.2 i2C initial settings ----------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
     /* ###################################################################################################################### */
     /* ###################################################################################################################### */
     /*                                                                                                                        */
     /*        Sets the I2C bus                                                                                                */
     /*                                                                                                                        */
     /* ###################################################################################################################### */
     /* ###################################################################################################################### */

#ifdef SLAVE_MODE
     /************************************************************************************************************************/
     /*   SAR            Slave address register                                                                              */
     /*        SVA6:0    = 1000000 (unique value)                                                                            */
     /*        FS        = 0  I2C format                                                                                     */
     /************************************************************************************************************************/
          /* ##(program note)################################################################################################ */
          /* ## SVA6:0 are used in the slave mode. They 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 frequency (ϕ/80, transfer rate: 200 kbps)                                      */
     /************************************************************************************************************************/

     IIC2.ICCR1.BYTE     = 0x84 ;
          /* ##(program note)################################################################################################ */
          /* ## Setting of CKS3:0 should be changed according to the required transfer rate.                             ## */
          /* ## For detailed information, please 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 ;
```

```
    /*****************************************************************************************************************************/
    /*   ICIER          Sets I2C interrupts                                                                                      */
    /*        TIE       = 0 Transmit interrupts disabled                                                                         */
    /*        TEIE      = 0  Transmit-end interrupts disabled                                                                    */
    /*        RIE       = 1  Receive interrupts enabled                                                                          */
    /*        NAKIE     = 0  NACK-receive interrupts disabled                                                                    */
    /*        STIE      = 0  Stop-condition-detection interrupts disabled                                                        */
    /*        ACKE      = 0  Acknowledgement judgments not used                                                                  */
    /*        ACKBR     = 0  Reception acknowledgement                                                                           */
    /*        ACKBT     = 0  Transmission acknowledgement (ACK = 0)                                                              */
    /*****************************************************************************************************************************/
    IIC2.ICIER.BYTE= 0x20 ;                                         /* Enables receive interrupts                            */
#endif

/* ------------------------------------------------------------------------------------------------------------------------------ */
/* 4.4 i2C interrupt processing ------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/*****************************************************************************************************************************/
/*   1. Module name: h8_i2c                                                                                                      */
/*   2. Function overview: Processing executed in response to an interrupt from the I2C bus                                      */
/*   3. History of revisions: REV  Date created/revised    Created/revised by     Revision contents                            */
/*                            000          2002.12.14           Ueda                  New                                       */
/*****************************************************************************************************************************/
#pragma interrupt( h8_i2c )
void h8_i2c ( void )
{
    int i , j , timer_wk;
    unsigned int ret ;
    unsigned char        slave_addr , dummy_data ;
    unsigned char        read_data[5]  ;


    ret = NORMAL_END ;
    /*****************************************************************************************************************************/
    /*   Clears set_imask_ccr to 0 to mask IREQ0-3 and SCI rcvint interrupts.                                                     */
    /*   Enable timer Z interrupts alone                                                                                         */
    /*****************************************************************************************************************************/
    com_int_ctl(0) ;                                                /* Clears ccr to 0 to enable timer Z interrupts alone   */

    /*****************************************************************************************************************************/
    /*   Checks if it is a receive interrupt                                                                                     */
    /*****************************************************************************************************************************/
    if (IIC2.ICSR.BIT.RDRF == 1){                                   / Reception                                             */
        if (IIC2.ICSR.BIT.AAS == 1){                                /* Slave address matching                                */
            /*************************************************************************************************************************/
            /*   Receives salve_addr and r/w                                                                                     */
            /*************************************************************************************************************************/
            slave_addr = IIC2.ICDRR ;

            if ((slave_addr & 0x01) == 0){                          /* write                                                 */
                /*********************************************************************************************************************/
                /*   Receives 4-byte data                                                                                        */
                /*********************************************************************************************************************/
                for (i=0; i< 4 ; i++){
```

```
        /*****************************************************************************************************/
        /*   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 this remains 1 for 5 seconds,                    */
                                                        /* exits with an error.                                */
                ret = I2C_RDRF_TOUT;                    /* Abnormal end (timeout)                              */
                goto exit ;
            }

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

        /*****************************************************************************************************/
        /*   Reads received data                                                                             */
        /*****************************************************************************************************/
        read_data[i] = IIC2.ICDRR ;                     /* data read                                           */
}

/*****************************************************************************************************/
/*   Confirms that ICCR1(TRS)=1.                                                                     */
/*****************************************************************************************************/
com_timer.wait_100ms_scan = 50 ;
while (IIC2.ICCR1.BIT.TRS == 0){                        /* Waits until the system enters                       */
                                                        /* the slave transmit mode                             */
    timer_wk = com_timer.wait_100ms_scan  ;
    if (timer_wk == 0){                                 /* If this remains 1 for 5 seconds,                    */
                                                        /* exits with an error.                                */
        ret = I2C_TRS_TOUT;                             /* Abnormal end (timeout)                              */
        goto exit ;
    }

    #ifdef UT
        IIC2.ICCR1.BIT.TRS = 1 ;
    #endif
}
    /* ##(program note)################################################################################ */
    /* ## In the data of the 5th byte, because the 8th-bit data (R/W) is "1", the system automatically  ## */
    /* ## switches to the slave transmit mode, so there is no need to set IIC.ICCR.BIT.TRS to 1.        ## */
    /* ################################################################################################ */

for (i=0; i< 4 ; i++){
    /*****************************************************************************************************/
    /*   Confirms that ICSR (TDRE)=1.                                                                    */
    /*****************************************************************************************************/
    com_timer.wait_100ms_scan = 50 ;
    while (IIC2.ICSR.BIT.TDRE == 0){                    /* Waits until preparation                             */
                                                        /* for transfer has been completed */
        timer_wk = com_timer.wait_100ms_scan  ;
        if (timer_wk == 0){                             /* If this remains 1 for 5 seconds,                    */
                                                        /* exits with an error.                                */
            ret = I2C_TDRE_TOUT;                        /* Abnormal end (timeout)                              */
            goto exit ;
        }
```

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

                }
                /*****************************************************************************************************/
                /*   Sets data                                                                                       */
                /*****************************************************************************************************/
                IIC2.ICDRT = read_data[i] ;
            }
            /* ##(program note)################################################################################### */
            /* ## This example shows a case in which 4-byte data from the master device is received.              ## */
            /* ## If the received data is configured as a packet and sent together with the packet length,        ## */
            /* ##  transmission and reception of variable-length data is also possible.                          ## */
            /* ##################################################################################################### */


            /*****************************************************************************************************/
            /*   Confirms that ICSR (TDRE) = 1                                                                   */
            /*****************************************************************************************************/
            com_timer.wait_100ms_scan = 50 ;
            while (IIC2.ICSR.BIT.TDRE == 0){                    /* Waits until preparation                     */
                                                                /* for transfer has been completed             */
                timer_wk = com_timer.wait_100ms_scan  ;
                if (timer_wk == 0){                             /* If this remains 1 for 5 seconds,            */
                                                                /* exits with an error.                        */
                    ret = I2C_TDRE_TOUT;                        /* Abnormal end (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 this remains 1 for 5 seconds,            */
                                                                /* exits with an error.                        */
                    ret = I2C_TEND_TOUT;                        /* Abnormal end (timeout)                      */
                    goto exit ;
                }

                #ifdef UT
                    IIC2.ICSR.BIT.TEND = 1 ;
                #endif
            }
        }
    }
exit :
    /*****************************************************************************************************/
    /*   Resets ICSR (TEND)                                                                              */
    /*****************************************************************************************************/
    IIC2.ICSR.BIT.TEND = 0 ;
```

```
    /*****************************************************************************************************************************************/
    /*   Resets ICCR1 (TRS) (slave receive mode)                                                                                             */
    /*****************************************************************************************************************************************/
    IIC2.ICCR1.BYTE = 0x84 ;

    /*****************************************************************************************************************************************/
    /*   SCL is released when a dummy read is carried out.                                                                                   */
    /*****************************************************************************************************************************************/
    dummy_data = IIC2.ICDRR ;                                       /* data read                                                           */

    /*****************************************************************************************************************************************/
    /*   Resets the interrupt source.                                                                                                        */
    /*****************************************************************************************************************************************/
    IIC2.ICSR.BIT.AAS   = 0 ;                                       /* Slave address matching                                              */

    /*****************************************************************************************************************************************/
    /*   Outputs an error message to the console.                                                                                            */
    /*****************************************************************************************************************************************/
    sprintf(com_debug_info,"recive data =%02X%02X%02X%02X \n",
    read_data[0],read_data[1],read_data[2],read_data[3]);
    com_write_sireal_data(&com_debug_info[0]) ;

    if (ret != NORMAL_END){                                         /* Abnormal end (timeout                                               */
         sprintf(com_debug_info,"  i2c_int_exec err(%02X)\n",ret);
         com_write_sireal_data(&com_debug_info[0]) ;
    }

    /*****************************************************************************************************************************************/
    /*   Unmasks the IREQ0-3 and SCI rcvint interrupts                                                                                       */
    /*****************************************************************************************************************************************/
    com_int_ctl(1) ;

}
```

```
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* 5 . Sample Program 2-E  TimerZ Processing ------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------ */

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 5.1 Addition of the reset vector ---------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/* Set the jump destination to h8_timerz.                                                                                   */

/* ------------------------------------------------------------------------------------------------------------------------ */
/*  5.2 Common variable definitions for TimerZ ----------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
     struct    {
          int counter;                                          /* 100 ms counter                                       */
          int wait_10ms;                                        /* For wait time of 10 ms                               */
          int wait_100ms;                                       /* For wait time in 100 ms units (common)               */
          int wait_100ms_scan;                                  /* For wait time in 100 ms units (for I2C)              */
     }com_timer;

/* ------------------------------------------------------------------------------------------------------------------------ */
/* 5.3 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 on a GRA compare-match */
                                                           /* CKEG[1:0] = 00 Counts up at the rising edge          */
                                                           /* TPSC[2:0] = 011 Counts using internal clock ϕ/8      */
     TZ0.TIORA.BYTE = 0x00 ;
                                                           /* IOA[2:0] = 000                                       */
                                                           /* GRA functions as an output-compare register          */
     TZ0.TIER.BYTE  = 0x01 ;
                                                           /* IMIEA = 1 Enables IMFA                               */

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

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

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


/* ------------------------------------------------------------------------------------------------------------------------- */
/* 5.4 TimerZ interrupt processing ----------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------- */
/************************************************************************************************************************/
/*   1. Module name: h8_TimerZ                                                                                              */
/*   2. Function overview: 10-msec interval timer processing                                                                */
/*   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 interrupt source                                                                                        */
     /************************************************************************************************************************/
     com_global.dummy = TZ0.TSR.BYTE;                               /* dummy read                                           */

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

     /************************************************************************************************************************/
     /*   Decrement by 1 every 10 msec                                                                                       */
     /************************************************************************************************************************/
     if( com_timer.wait_10ms>0 )
          com_timer.wait_10ms --;

     /************************************************************************************************************************/
     /*   Counting up                                                                                                        */
     /************************************************************************************************************************/
     com_timer.counter++;
     if( com_timer.counter >= 10 ){
          /************************************************************************************************************************/
          /*   Decrement by 1 every 10 msec                                                                                   */
          /************************************************************************************************************************/
          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.)
- I²C 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.