

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.

H8/3664

Master-Slave Communication using I²C Interface (H8/3664)

Introduction

The H8/3664 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 H8/3664 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 H8/3664- Application Notes consist of a "Basic Edition" which describes operation examples when using the individual on-chip peripheral functions of the H8/3664 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

H8/3664

Contents

1. Specifications	2
2. Configuration	2
3. Sample Programs	3
4. Reference Documents	39

1. Specifications

Communication between microcomputers is carried out via the I²C interface of the H8/3664.

2. Configuration

Figure 2.1 shows a diagram of connection between microcomputers.

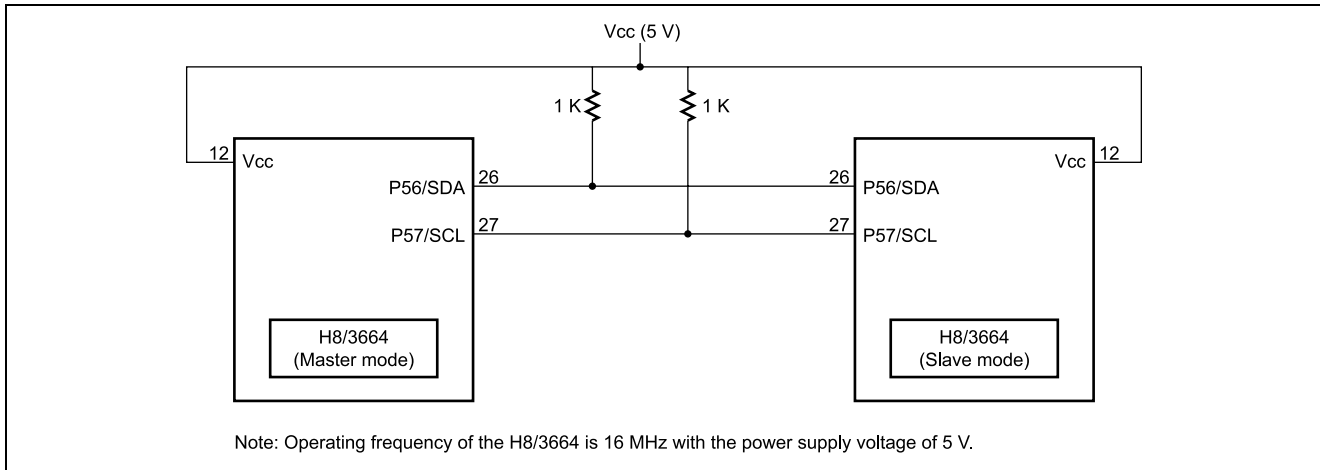


Figure 2.1 Diagram of connection between microcomputers

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 W 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 W reset vector.
 - Add com_timer as a common variable.
 - Add timer W initial setting processing.
(The GRA setting should be changed according to the operating frequency of the microcomputer being used, so that the timer W 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 W interrupt processing.
3. The I²C interface transfer rate ICMR(CKS2:0) and TSCR(IICX) should be set according to the target device specifications and the microcomputer operating frequency. Refer to the H8/3664 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 device 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 (transmit preparation completion wait timeout)
3	Abnormal termination (acknowledge timeout)
4	Abnormal termination (transmission 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 *recv_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.
*recv_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 (transmit preparation completion wait timeout)
3	Abnormal termination (acknowledge timeout)
4	Abnormal termination (transmission completion wait timeout)
5	Abnormal termination (reception completion wait timeout)
6	Abnormal termination (halt condition detection timeout)

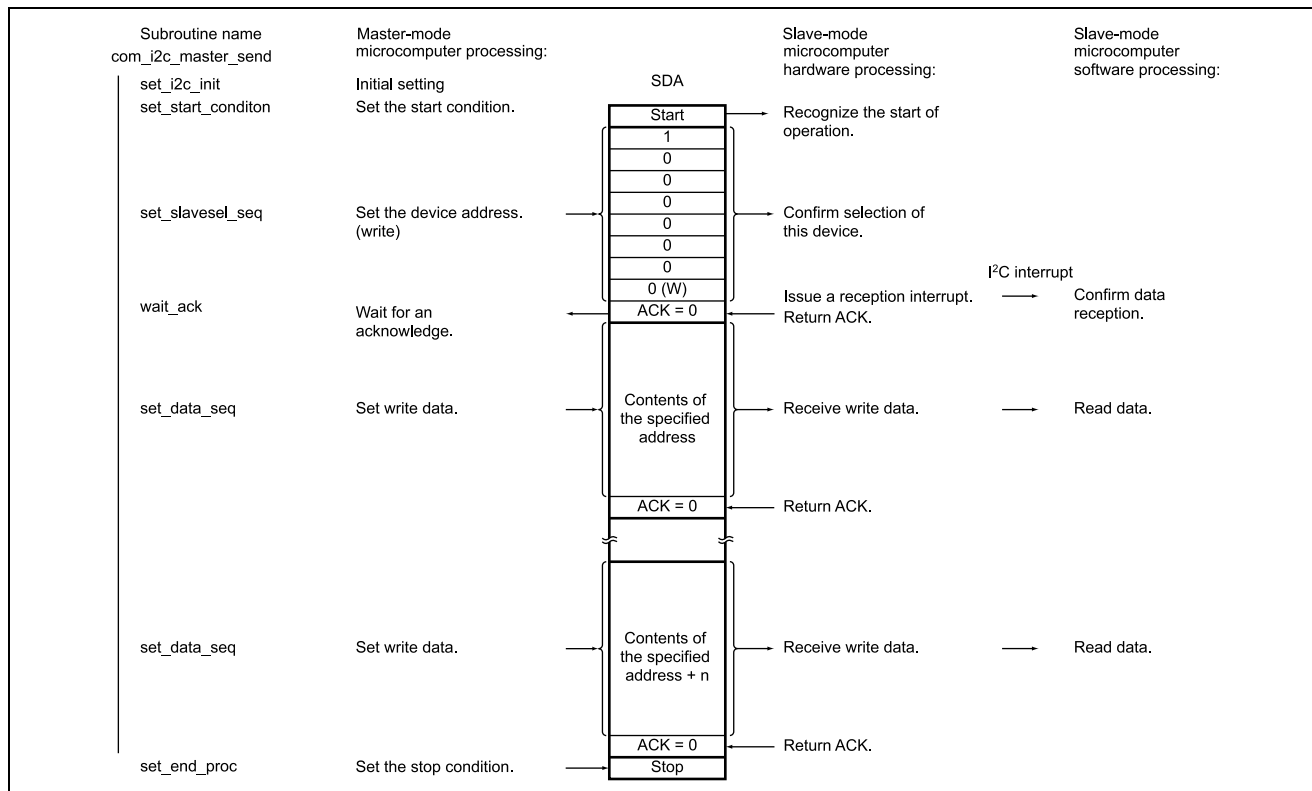
Example of use:

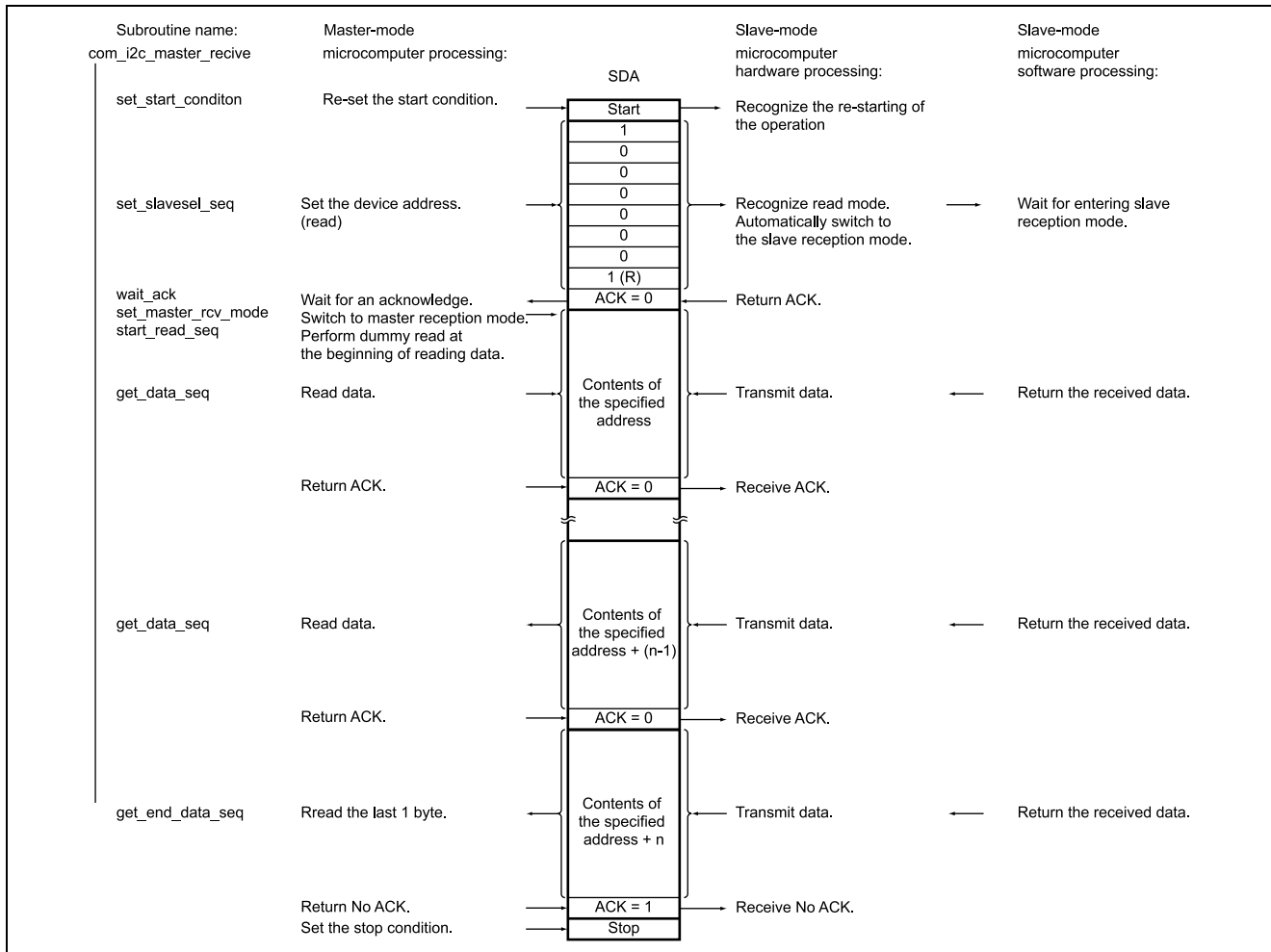
```
int ret ;
unsigned char slave_addr ;
unsigned int data_length ;
unsigned char recv_data[256] ;
ret = com_i2c_master_recive (slave_addr , data_length , &recv_data[0] )
```

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.

- 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.





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 H8/3664 Group Hardware Manual.

1. I²C-related registers

Name	Summary
I ² C bus data register (ICDR)	8-bit readable/writable register that functions as a transmission data register during transmission, and as a reception data register during reception.
Slave address register (SAR)	Sets the slave address and transfer format.
Second slave address register (SARX)	Sets the second slave address and transfer format.
I ² C bus mode register (ICMR)	Sets the transfer format and transfer rate. Can only be accessed when the ICE bit of ICCR is 1.
I ² C bus control register (ICCR)	I ² C bus interface control bits and interrupt request flags
I ² C bus status register (ICSR)	Status flags
Timer serial control register (TSCR)	8-bit readable/writable register that controls the operation mode.

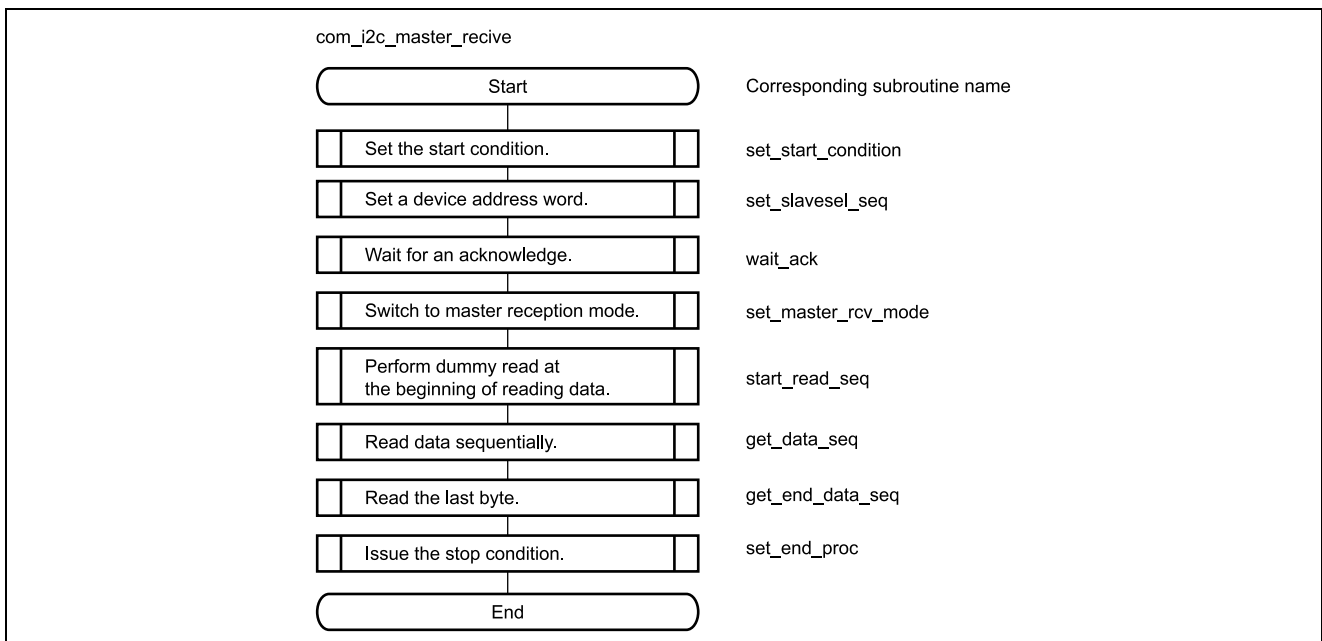
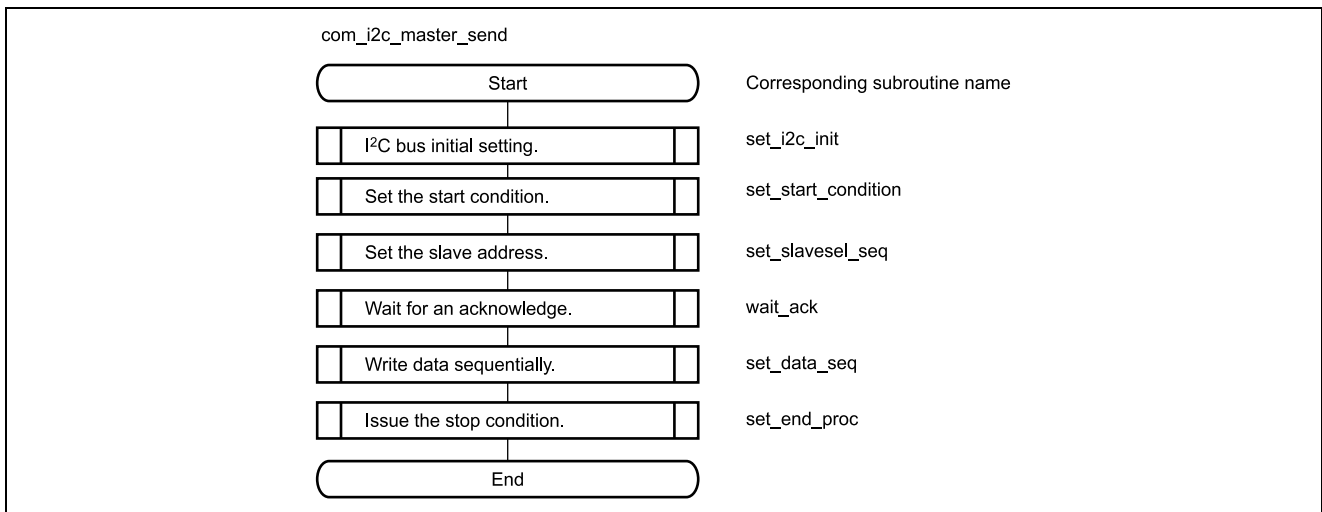
2. Timer W-related registers

Timer W 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.

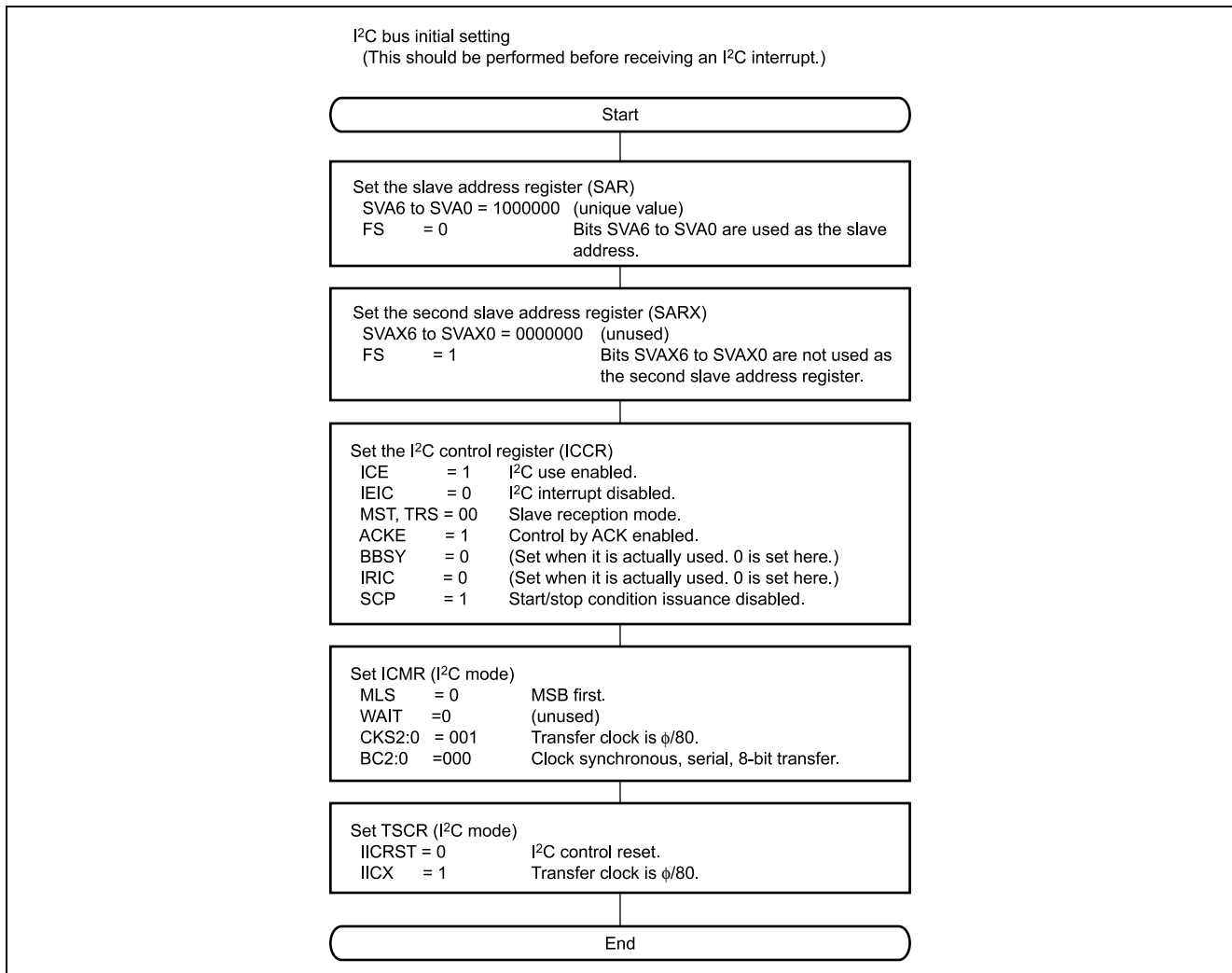
Name	Summary
Timer mode register W (TMRW)	Selects the general register functions, timer output mode, etc.
Timer control register W (TCRW)	Selects the TCNT counter clock, counter clear conditions, and timer initial output level settings.
Timer interrupt enable register W (TIERW)	Controls timer W interrupt requests.
Timer I/O control register 0 (TIOR0)	Selects the functions of the GRA and GRB and of the FTIOA and FTIOB pins.
Timer I/O control register 1 (TIOR1)	Selects the functions of the GRC and GRD and of the FTIOC and FTIOD pins. Not used in this sample program.
Timer counter (TCNT)	16-bit readable/writable upward counter
general registers A, B, C, D (GRA, GRB, GRC, GRD)	16-bit readable/writable registers which can be used as either output-compare registers or input-capture registers.

3.7 Flowcharts

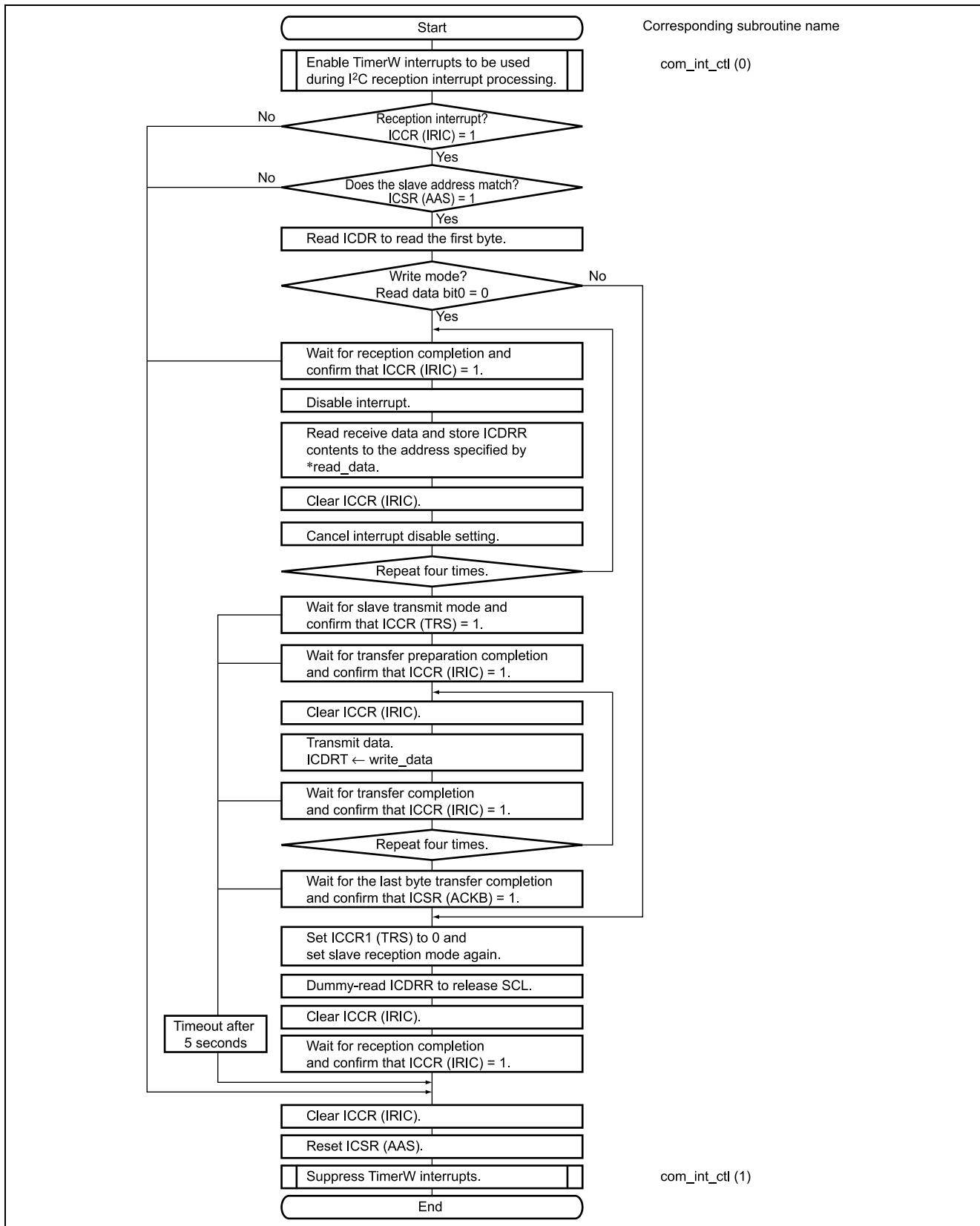
1. Master-mode H8 microcomputer processing

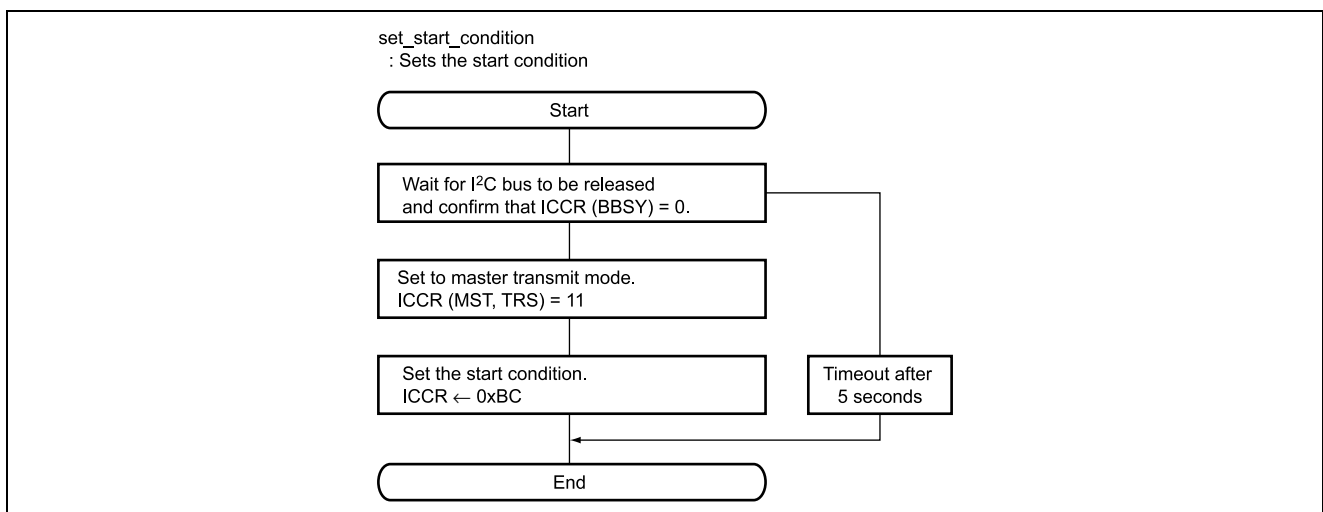
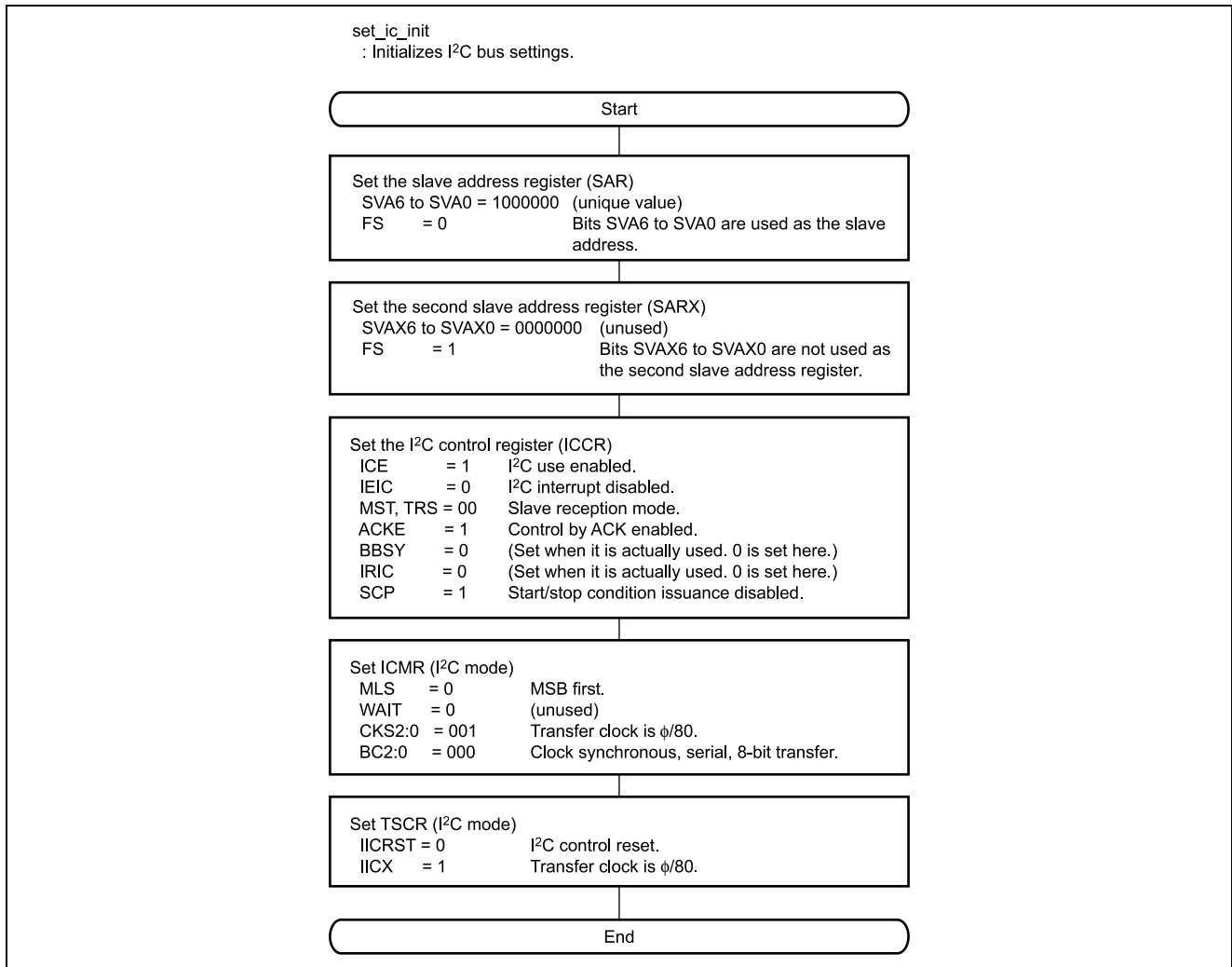


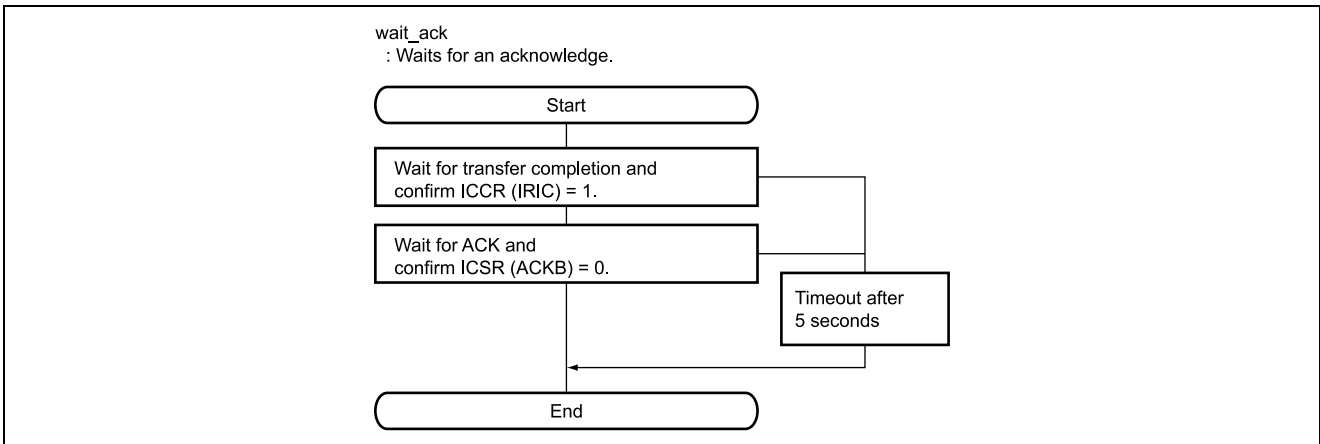
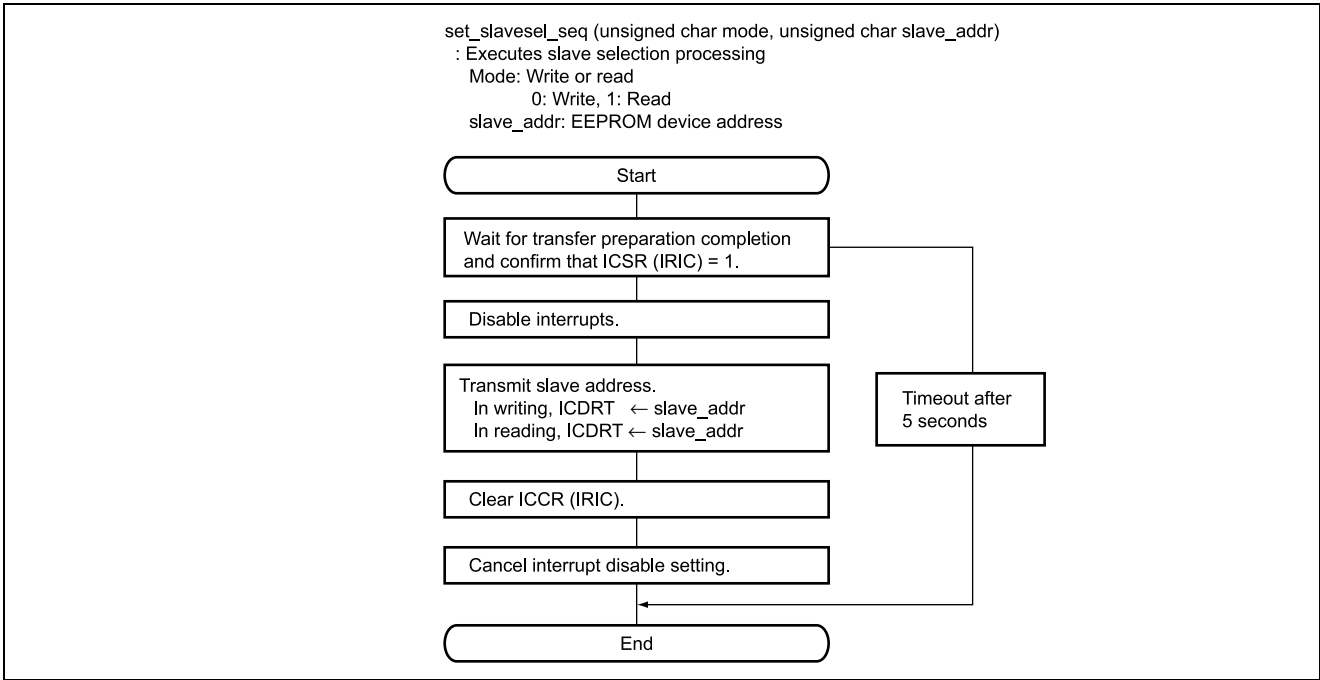
2. Slave-mode H8 microcomputer processing

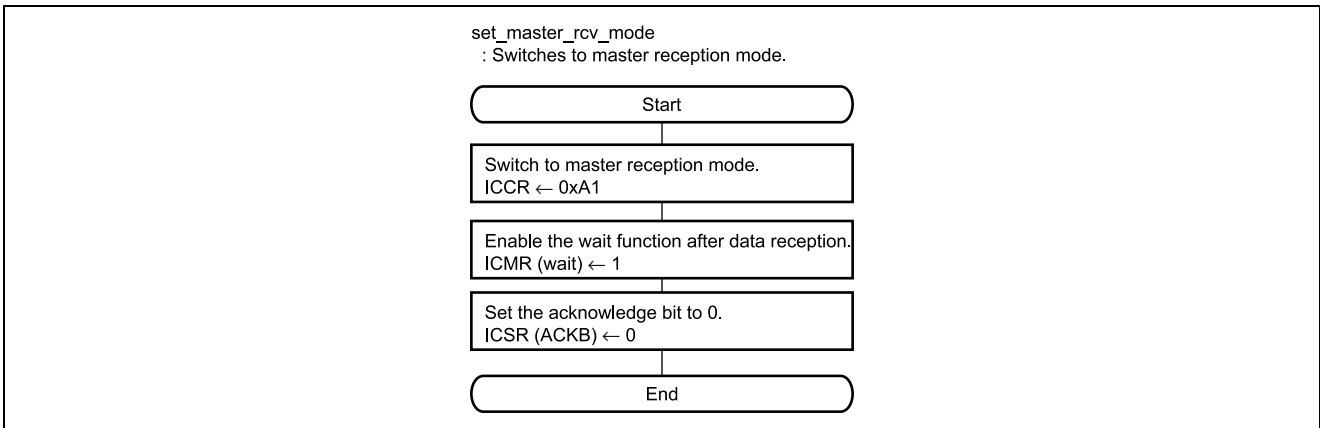
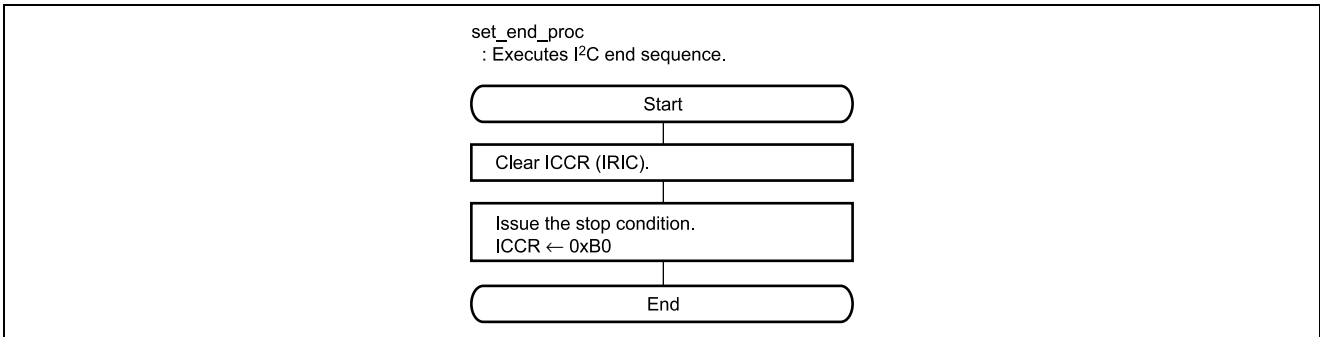
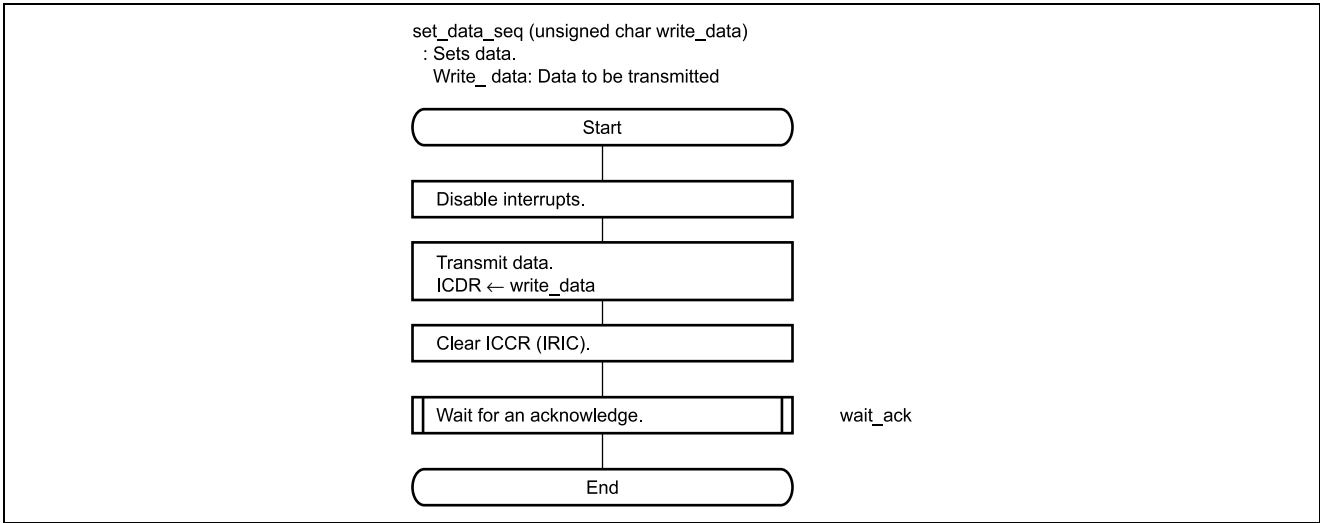


3. I²C interrupt processing

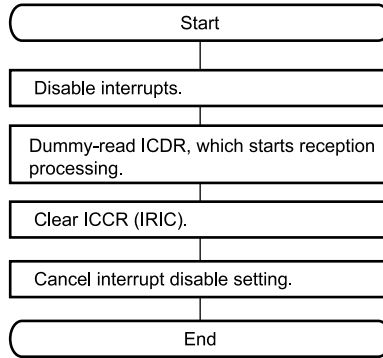




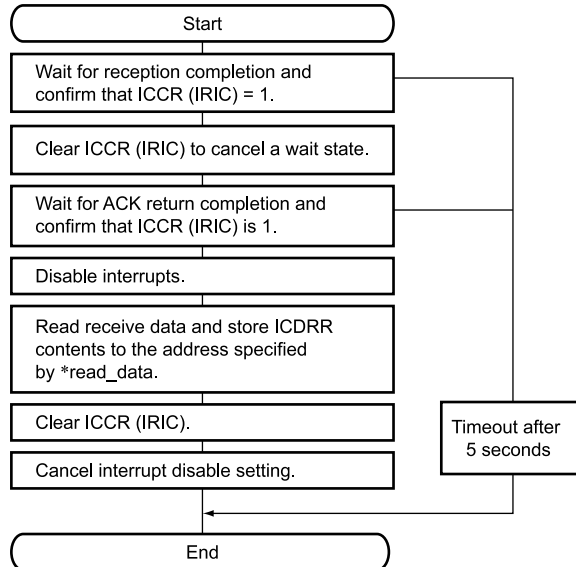


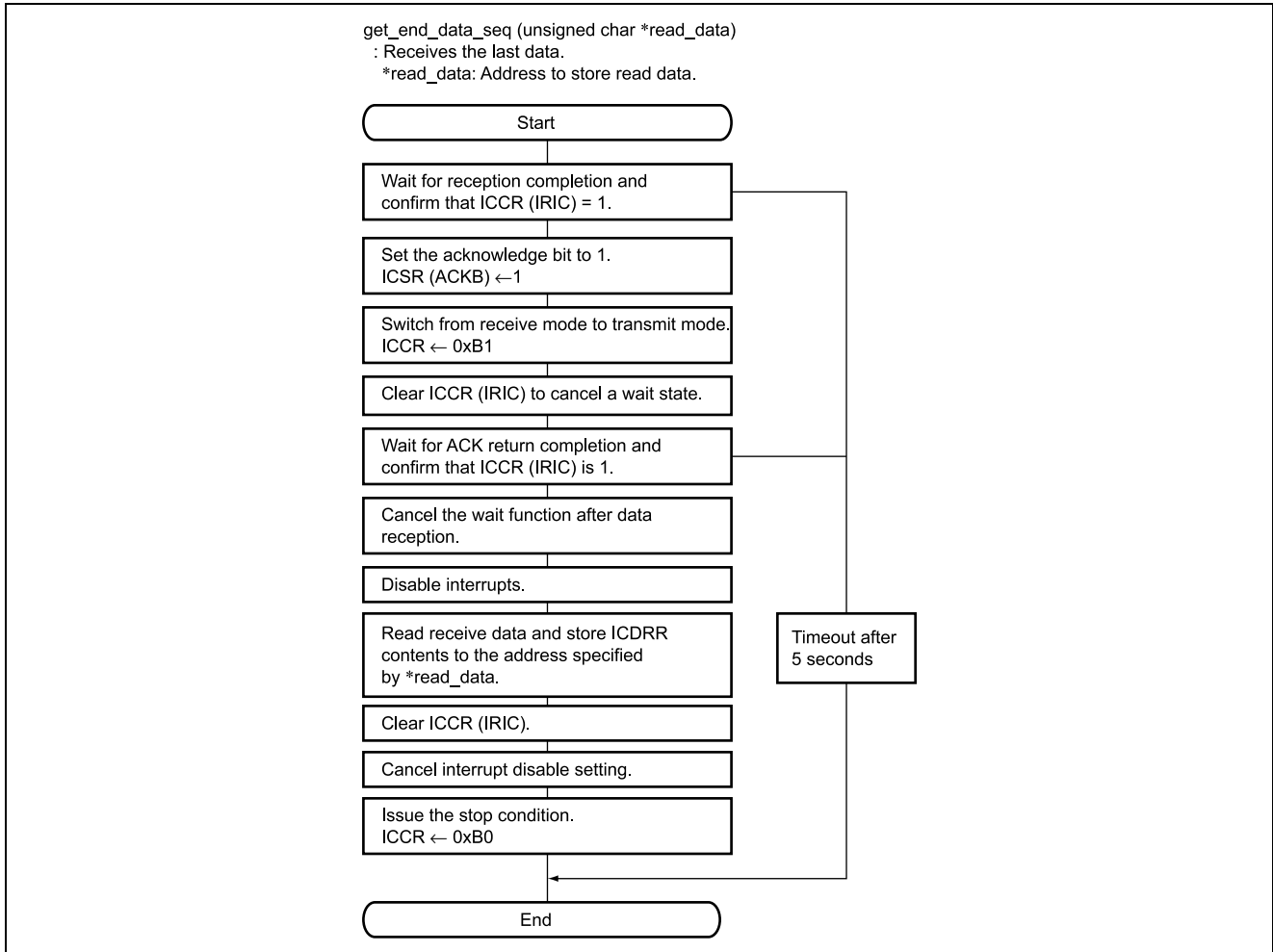


start_read_seq
: Performs dummy read at the beginning of read processing.



get_data_seq(unsigned char *read_data)
: Receives data.
*read_data: Address to store read data





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_IRIC_TOUT 2
#define I2C_ACKB_TOUT 3
#define I2C_IRTR_TOUT 4
#define I2C_TRS_TOUT 5

/*****
/* ↓ 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 int get_end_data_seq ( unsigned char *read_data );
unsigned int get_data_seq ( unsigned char *read_data );

unsigned int com_i2c_eeprom_read( unsigned char device_addr_code , unsigned int rom_addr , unsigned char *rom_data );

unsigned int com_i2c_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 *receive_data );

```

```

/* ----- */
/* ----- */
/* 3. Sample program 2-C Source code----- */
/* ----- */
/* ----- */
/*****
/* 1. Module name: com_int_ct1 */
/* 2. Function overview: Clears set_imask_ccr to 0 to enable Timer W interrupts alone. */
/* 3. History of revisions: REV Date created/revised Created/revised by Revision contents */
/*          000      2002.04.10      Ueda          New */
*****/
void com_int_ct1 (unsigned char kind)
{
    if (kind == 0){
        /*****
        /* Disables I2C reception interrupts */
        *****/
        IIC.ICCR.BIT.IEIC = 0 ;

        /*****
        /* Enables Timer W interrupts */
        *****/
        TW.TIERW.BIT.IMIEA = 1 ;          /* timerW 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 */
        *****/
        IIC.ICCR.BIT.IEIC = 1 ;

        /*****
        /* Disables TimerW interrupts */
        *****/
        TW.TIERW.BIT.IMIEA = 0 ;          /* timerW 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++;
}

```

```

/*****
/*****
/*****
/*
/*
/*          I2C EEPROM control
/*
/*
/*****
/*****
/*****

/*****
/*  1. Module name: set_i2c_init
/*  2. Function overview: Sets initial settings prior to I2 access
/*  3. History of revisions: REV  Date created/revised  Created/revised by  Revision contents
/*
/*          000          2002.12.14          Ueda          New
/*****
void set_i2c_init( )
{
/*****
/*  SAR          Sets slave address register
/*  SVA6:0      = 1000000 (unique value)
/*  FS          = 0  SVA6:0 are used as the slave address.
/*****
/*  ##(program note)#####
/*  ## SVA6:0 are used in 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  ##
/*  #####
IIC.SAR.BYTE= 0x80 ;

/*****
/*  SARX        Sets 2nd slave address register
/*  SVAX6:0     = 0000000 (not used)
/*  FS          = 1  SVAX6:0 is not used as the second slave address.
/*****
IIC.SARX.BYTE= 0x01 ;

/*****
/*  ICCR        Sets the I2C control register
/*  ICE         = 1  I2C use enabled
/*  IEIC        = 0  Interrupts not used yet
/*  MST,TRS     = 00 Slave receive mode
/*  ACKE        = 1  Makes ACK judgments valid
/*  BBSY        = 0  (This bit set during actual operation; here, set to 0)
/*  IRIC        = 1  (This bit is set during actual operation; here, set to 0)
/*  SCP         = 1  Disables issuing of start/stop conditions
/*****
IIC.ICCR.BYTE = 0x89 ;

/*****
/*  ICMR        Sets I2C mode
/*  MLS         = 0  MSB first
/*  WAIT        = 0  (Not used)
/*  CKS2:0      = 001 Transfer clock is φ/80
/*  BC2:0       = 000 Clock synchronous, serial, 8-bit transfer
/*****
IIC.ICMR.BYTE= 0x08 ;

/*  ##(program note)#####
/*  ## Setting of CKS2:0 should be changed according to the required transfer rate.  ##
/*  ## For details, please refer to the H8/3664 Hardware Manual.  ##
/*  #####

```

```

/*****
/*  TSCR      Sets I2C mode                                     */
/*      IICRST = 0 Resets the I2C control                     */
/*      IICX   = 1 Transfer clock is φ/80                    */
/*****
TSCR.BYTE= 0x01 ;
/* ##(program note)##### */
/* ## Setting of IICX should be changed according to the required transfer rate. ## */
/* ## For details, please refer to the H8/3664 Hardware Manual. ## */
/* ##### */

}

/*****
/*  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 ICCR (BBSY) = 0.                             */
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.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
        IIC.ICCR.BIT.BBSY = 0;
    #endif
}

/*****
/*  Sets master transmit mode                                 */
/*****
IIC.ICCR.BYTE = 0xB9 ;                                       /* Sets master transmit mode */
/* ##(program note)##### */
/* ## Be careful not to reset the ACKE bit (ACK judgment effective). ## */
/* ##### */

/*****
/*  Sets the start condition                                 */
/*****
IIC.ICCR.BYTE = 0xBC ;
/* ##(program note)##### */
/* ## Bits 2 and 0, which set the start condition, must 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. ## */
/* ##### */
/* ##(program note)##### */
/* ## Be careful that the ACKE (ACK judgment effective) does not end up being reset. ## */
/* ##### */

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 ICCR (IRIC)=1. */
    *****/
    com_timer.wait_100ms_scan = 50 ;
    while (IIC.ICCR.BIT.IRIC == 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_IRIC_TOUT; /* Abnormal end (timeout) */
            goto exit ;
        }

        #ifdef UT
            IIC.ICCR.BIT.IRIC = 1 ;
        #endif
    }

    /*****
    /* Confirms that ICSR (ACKB)=0. */
    *****/
    com_timer.wait_100ms_scan = 50 ;
    while (IIC.ICSR.BIT.ACKB == 1){ /* Waits for ACKB=0 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_ACKB_TOUT; /* Abnormal end (timeout) */
            goto exit ;
        }

        #ifdef UT
            IIC.ICSR.BIT.ACKB = 0 ;
        #endif
    }

    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 ICCR (IRIC)=1.
    *****/
    com_timer.wait_100ms_scan = 50 ;
    while (IIC.ICCR.BIT.IRIC == 0){
        timer_wk = com_timer.wait_100ms_scan ;
        if (timer_wk == 0){
            ret = I2C_IRIC_TOUT;
            goto exit ;
        }

        #ifdef UT
            IIC.ICCR.BIT.IRIC = 1 ;
        #endif
    }

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

    /*****
    /* Disables interrupts
    *****/
    set_imask_ccr(1);

    /*****
    /* Writes data
    *****/
    IIC.ICDR = slave_addr ;

    /*****
    /* Clears ICCR (IRIC)
    *****/
    IIC.ICCR.BIT.IRIC = 0 ;
    /* ##(program note)#####
    /* ## Writing to ICDR and clearing of IRIC should take place as successive write operations within the time ##
    /* ## taken for 1-byte data transfer. ##
    /* ## For this reason, all interrupts should be disabled while these operations are being carried out. ##
    /* #####

    /*****
    /* Cancels interrupt disable
    *****/
    set_imask_ccr(0);

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 ;

    /*****
    /* Disables interrupts
    *****/
    set_imask_ccr(1);          /* Disables interrupts

    /*****
    /* Writes data
    *****/
    IIC.ICDR = write_data ;

    /*****
    /* Clears ICCR (IRIC)
    *****/
    IIC.ICCR.BIT.IRIC = 0 ;
    /* ##(program note)##### */
    /* ## Writing to ICDR and clearing of IRIC should take place as successive write operations within ## */
    /* ## the time taken for 1-byte data transfer ## */
    /* ## For this reason, all interrupts should be disabled while these operations are being carried out. ## */
    /* ##### */

    /*****
    /* Cancels interrupt disable
    *****/
    set_imask_ccr(0);          /* Cancels interrupt disable

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

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 ;

    /*****
    /* Clears ICCR (IRIC)
    *****/
    IIC.ICCR.BIT.IRIC = 0 ;

```

```

/*****
/* Issues the stop condition */
/*****
IIC.ICCR.BYTE = 0xB0 ;

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 ;

/*****
/* Switches to master receive mode */
/*****
IIC.ICCR.BYTE = 0xA1 ;
/*****
/* Sets an ICMR (WAIT) */
/*****
IIC.ICMR.BIT.WAIT = 1 ;

/*****
/* Clears ICSR (ACKB) to 0 (sets Acknowledge data) */
/*****
IIC.ICSR.BIT.ACKB = 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 dummy_data ;

/*****
/* Disables interrupts */
/*****
set_imask_ccr(1); /* Disables interrupts */

/*****
/* Initiates reception by executing a dummy read */
/*****
dummy_data = IIC.ICDR ;
/* ##(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. ## */
/* ##### */

```

```

/*****
/*  Clears ICCR (IRIC)
/*****
IIC.ICCR.BIT.IRIC = 0 ;
    /* ##(program note)##### */
    /* ## Writing to ICDR and clearing of IRIC should take place as successive write operations within the time ## */
    /* ## taken for 1-byte data transfer. ## */
    /* ## For this reason, all interrupts should be disabled while these operations are being carried out. ## */
    /* ##### */

/*****
/*  Cancels interrupt disable
/*****
set_imask_ccr(0);          /* Cancels interrupt disable */

}

/*****
/*  1. Module name: get_data_seq
/*  2. Function overview: Reads data from the I2C target device
/*  3. History of revisions: REV  Date created/revise  Created/revise 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 ICCR (IRIC) = 1
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.IRIC == 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.
        /* Abnormal end (timeout)
        ret = I2C_IRIC_TOUT;
        goto exit ;
    }

    #ifdef UT
        IIC.ICCR.BIT.IRIC = 1 ;
    #endif
}

/*****
/*  Clears ICCR (IRIC) (to cancel the wait status)
/*****
IIC.ICCR.BIT.IRIC = 0 ;

/*****
/*  Confirms that ICCR (IRIC) = 1
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.IRIC == 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.
        /* Abnormal end (timeout)
        ret = I2C_IRIC_TOUT;
        goto exit ;
    }

    #ifdef UT
        IIC.ICCR.BIT.IRIC = 1 ;
    #endif
}
}

```

```

/*****
/* Disables interrupts */
/*****
set_imask_ccr(1); /* Disables interrupts */

/*****
/* Reads received data */
/*****
*read_data = IIC.ICDR ; /* data read */

/*****
/* Clears ICCR (IRIC) */
/*****
IIC.ICCR.BIT.IRIC = 0 ;
/* ##(program note)##### */
/* ## Writing to ICDR and clearing of IRIC should take place as successive write operations within the time ## */
/* ## taken for 1-byte data transfer. ## */
/* ## For this reason, all interrupts should be disabled while these operations are being carried out. ## */
/* ##### */

/*****
/* Cancels interrupt disable */
/*****
set_imask_ccr(0); /* Cancels interrupt disable */

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 ICCR (IRIC) = 1 */
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.IRIC == 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_IRIC_TOUT; /* Abnormal end (timeout) */
goto exit ;
}

#ifdef UT
IIC.ICCR.BIT.IRIC = 1 ;
#endif
}

/*****
/* Sets value for ACK returned after data reception to "1" (NOACK) */
/*****
IIC.ICSR.BIT.ACKB = 1 ;

```

```

/*****
/* Switches from receive mode to transmit mode */
/*****
IIC.ICCR.BYTE = 0xB1 ;

/*****
/* Clears ICCR (IRIC) (to cancel the Wait state) */
/*****
IIC.ICCR.BIT.IRIC = 0 ;

/*****
/* Confirms that ICCR (IRIC) = 1 */
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.IRIC == 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_IRIC_TOUT;                                  /* Abnormal end (timeout) */
        goto exit ;
    }

    #ifdef UT
        IIC.ICCR.BIT.IRIC = 1 ;
    #endif
}

/*****
/* Resets ICMR (WAIT) */
/*****
IIC.ICMR.BIT.WAIT = 0 ;

/*****
/* Disables interrupts */
/*****
set_imask_ccr(1);                                           /* Disables interrupts */

/*****
/* Reads received data */
/*****
*read_data = IIC.ICDR ;                                     /* data read */

/*****
/* Clears ICCR (IRIC) (to cancel the Wait state) */
/*****
IIC.ICCR.BIT.IRIC = 0 ;
    /* ##(program note) ##### */
    /* ## Writing to ICDR and clearing of IRIC should take place as successive write operations within the time ## */
    /* ## taken for 1-byte data transfer. ## */
    /* ## For this reason, all interrupts should be disabled while these operations are being carried out. ## */
    /* ##### */

/*****
/* Cancels interrupt disable */
/*****
set_imask_ccr(0);                                           /* Cancels interrupt disable */

/*****
/* Clears ICCR (IRIC) and issues the stop condition */
/*****
IIC.ICCR.BYTE = 0xB0 ;

exit :

    return (ret);
}

```

```

/*****
/* 1. Module name: com_i2c_master_recv                                     */
/* 2. Function overview: Receives data of the specified data length from the slave device */
/* 3. History of revisions: REV  Date created/revise   Created/revise by   Revision contents */
/*                               000      2002.12.14      Ueda                New                */
/*****
unsigned int com_i2c_master_recv ( unsigned char slave_addr , unsigned int data_length , unsigned char *recv_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 ;}

    /*****
    /* Waits for an acknowledgement                                     */
    /*****
    ret = wait_ack() ;
    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 () ;

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

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

```

```

/*****
/* 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 ;}

    *receive_data = buf.d_byte[0] ;

    return (ret);

exit :
/*****
/* Resets the I2C interrupt request flag and issues the stop condition if an error occurs */
/*****
set_end_proc ( ) ;
    return (ret);
}

/*****
/* 1. Module name: com_i2c_master_send */
/* 2. Function overview: Transfers 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 settings */
    /*****
    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 interrupt request flag and issues the stop condition if an error occurs */
/*****
set_end_proc ( ) ;

return (ret);

}

```



```

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

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

/* ----- */
/* 4.2 i2c initial settings ----- */
/* ----- */
/* ##### */
/* ##### */
/* Sets the I2C bus */
/* ##### */
/* ##### */
#ifdef SLAVE_MODE
/* ----- */
/* SAR Sets the slave address register */
/* SVA6:0 = 1000000 (unique value) */
/* FS = 0 SVA6:0 are used as the slave address */
/* ----- */
/* ##(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 ## */
/* ##### */
IIC.SAR.BYTE= 0x80 ;

/* ----- */
/* SARX Sets the second slave address register */
/* SVAX6:0 = 0000000 (Not used) */
/* FS = 1 SVAX6:0 are not used as the second slave address. */
/* ----- */
IIC.SARX.BYTE= 0x01 ;

/* ----- */
/* ICCR Sets the I2C control register */
/* ICE = 1 I2C use enabled */
/* IEIC = 1 Interrupt enabled */
/* MST,TRS = 00 Slave receive mode */
/* ACKE = 1 ACK is used */
/* BBSY = 0 (This bit is set during actual operation; here, set to 0) */
/* IRIC = 0 (This bit is set during actual operation; here, set to 0) */
/* SCP = 1 Disables issuing of the start/stop conditions */
/* ----- */
IIC.ICCR.BYTE = 0xC9 ;

```

```

/*****
/*  ICMR      Sets I2C mode
/*      MLS      = 0  MSB first
/*      WAIT     = 0  (Not used)
/*      CKS2:0   = 001 Transfer clock is φ/80
/*      BC2:0    = 000 Clock synchronous, serial, 8-bit transfer
/*****
IIC.ICMR.BYTE= 0x08 ;
/* ##(program note)##### */
/* ## The setting of CKS2:0 should be changed according to the required transfer rate.      ## */
/* ## For detailed information, please refer to the H8/3664 Hardware Manual.                ## */
/* ##### */

/*****
/*  TSCR      Sets I2C mode
/*      IICRST   = 0  Resets I2C control
/*      IICX     = 1  Transfer clock is φ/80
/*****
TSCR.BYTE= 0x01 ;
/* ##(program note)##### */
/* ## The setting of CKS2:0 should be changed according to the required transfer rate.      ## */
/* ## For detailed information, please refer to the H8/3664 Hardware Manual.                ## */
/* ##### */

#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      2002.02.12      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 W interrupts alone
/*****
com_int_ctl(0) ;                               /* Clears ccr to 0 to enable timer W interrupts alone */

```

```

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

        /* Clears ICCR (IRIC) */
        IIC.ICCR.BIT.IRIC = 0 ;

        if ((slave_addr & 0x01) == 0){
            /* write */
            /* eives 4-byte data */
            for (i=0; i< 4 ; i++){
                /* Confirms that ICCR (IRIC) = 1 */
                com_timer.wait_100ms_scan = 50 ;
                while (IIC.ICCR.BIT.IRIC == 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_IRIC_TOUT;
                        /* Abnormal end (timeout) */
                        goto exit ;
                    }

                    #ifdef UT
                        IIC.ICCR.BIT.IRIC = 1 ;
                    #endif
                }

                /* Disables interrupts */
                set_imask_ccr(1);

                /* Reads received data */
                read_data[i] = IIC.ICDR ;
                /* data read */

                /* Clears ICCR (IRIC) */
                IIC.ICCR.BIT.IRIC = 0 ;

                /* ##(program note)##### */
                /* ## Writing to ICDR and clearing of IRIC should take place as successive write ## */
                /* ## operations within the time taken for 1-byte data transfer. ## */
                /* ## For this reason, all interrupts should be disabled while these operations are ## */
                /* ## being carried out. ## */
                /* ##### */

                /* Cancels interrupt disable */
                set_imask_ccr(0);
                /* Cancels interrupt disable */
            }
        }
    }
}

```

```

/*****
/*  Confirms that ICCR (TRS) = 1
*/
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.TRIS == 0){
    timer_wk = com_timer.wait_100ms_scan ;
    if (timer_wk == 0){
        ret = I2C_TRS_TOUT;
        goto exit ;
    }

    #ifndef UT
        IIC.ICCR.BIT.TRIS = 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.TRIS to 1. ## */
/* ##### */

/*****
/*  Confirms that ICCR (IRIC) = 1
*/
/*****
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.IRIC == 0){
    timer_wk = com_timer.wait_100ms_scan ;
    if (timer_wk == 0){
        ret = I2C_IRIC_TOUT;
        goto exit ;
    }

    #ifndef UT
        IIC.ICCR.BIT.IRIC = 1 ;
    #endif
}

for (i=0; i< 4 ; i++){
    /*****
    /*  Clears ICCR (IRIC)
    */
    /*****
    IIC.ICCR.BIT.IRIC = 0 ;

    /*****
    /*  Sets data
    */
    /*****
    IIC.ICDR = read_data[i] ;

    /*****
    /*  Confirms that ICCR (IRIC) = 1.
    */
    /*****
    com_timer.wait_100ms_scan = 50 ;
    while (IIC.ICCR.BIT.IRIC == 0){
        timer_wk = com_timer.wait_100ms_scan ;
        if (timer_wk == 0){
            ret = I2C_IRIC_TOUT;
            goto exit ;
        }

        #ifndef UT
            IIC.ICCR.BIT.IRIC = 1 ;
        #endif
    }
}
}

```

```

/* ##(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.                  ## */
/* ##### */

/*****/
/* End processing */
/*****/
/*****/
/* Confirms that ICSR (ACKB) = 1 */
/*****/
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICSR.BIT.ACKB == 0){
    timer_wk = com_timer.wait_100ms_scan ;
    if (timer_wk == 0){
        ret = I2C_ACKB_TOUT;
        goto exit ;
    }

    #ifdef UT
        IIC.ICSR.BIT.ACKB = 1 ;
    #endif
}

/*****/
/* Resets ICCR1 (TRS) (slave receive mode) */
/*****/
IIC.ICCR.BIT.TRIS = 0 ;

/*****/
/* dummy read */
/*****/
dummy_data = IIC.ICDR ;

/*****/
/* Clears ICCR (IRIC) */
/*****/
IIC.ICCR.BIT.IRIC = 0 ;

/*****/
/* Confirms that ICCR (IRIC) = 1 */
/*****/
com_timer.wait_100ms_scan = 50 ;
while (IIC.ICCR.BIT.IRIC == 0){
    timer_wk = com_timer.wait_100ms_scan ;
    if (timer_wk == 0){
        ret = I2C_IRIC_TOUT;
        goto exit ;
    }

    #ifdef UT
        IIC.ICCR.BIT.IRIC = 1 ;
    #endif
}

}

exit :
/*****/
/* Clears ICCR (IRIC) */
/*****/
IIC.ICCR.BIT.IRIC = 0 ;

```

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

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

}
```

```

/* ----- */
/* ----- */
/* 5. Sample Program 2-E TimerW Processing ----- */
/* ----- */
/* ----- */

/* ----- */
/* 5.1 Adding the reset vector ----- */
/* ----- */
/* Set the jump destination to h8_timerw. */

/* ----- */
/* 5.2 Common variable definitions for Timer W ----- */
/* ----- */
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 TimerW initial settings ----- */
/* ----- */
/* ##### */
/* ##### */
/* Sets TimerW */
/* ##### */
/* ##### */
/* Sets TimerW initial settings */
/* ##### */
TW.TCRW.BIT.CKS = 3 ; /* Counts using internal clock φ/8 */
TW.TCRW.BIT.CCLR = 1 ; /* Clears the counter on a GRA compare-match */
TW.TIOR0.BIT.IOA = 0 ; /* GRA is used as an output compare register. */
TW.TIERW.BIT.IMIEA = 1 ; /* Enables IMFA */
TW.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 H8/3664 Hardware Manual. ## */
/* ##### */

TW.TCNT = 0 ; /* Clears the timer counter */

/* ----- */
/* Cancels interrupt disable */
/* ----- */
set_imask_ccr(0); /* Enables interrupts */

/* ----- */
/* Starts Timer W */
/* By starting the timer before executing "com_change_frq", recovers from the sleep state using the Timer W interrupt */
/* without using direct transition interrupts, and changes the frequency. */
/* ----- */
TW.TMRW.BIT.CTS = 1 ; /* timer start */

```

```

/* ----- */
/* 5.4 Timer W interrupt processing ----- */
/* ----- */
/*****/
/* 1. Module name: h8_timerw */
/* 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_timerw )
void h8_timerw( void )
{

    /*****/
    /* Clears the interrupt source */
    /*****/
    com_global.dummy = TW.TSRW.BYTE;          /* dummy read */
    TW.TSRW.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 100 msec */
        /*****/
        if( com_timer.wait_100ms>0 )
            com_timer.wait_100ms --;
        if( com_timer.wait_100ms_scan>0 )
            com_timer.wait_100ms_scan --;
        if( com_timer.wait_100ms_sci3>0 )
            com_timer.wait_100ms_sci3 --;

        com_timer.counter = 0;
    }
}

```


4. Reference Documents

- H8/3664 Group Hardware Manual (published by Renesas Technology Corp.)
- I²C Bus Usage (published by Phillips)

Revision Record

Rev.	Date	Description	
		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.