H8/3687

Master-Slave Communication 2 using I^{2}C 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, I^{2}C bus interface, serial communication interface, and 10-bit A/D converter. These devices can be utilized as embedded microcomputers in sophisticated control systems.

These H8/300 H Series -H8/3687- Application Notes consist of a "Basic Edition" which describes operation examples when using the onboard 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 employs the H8/3687 I^2C interface.

2. Configuration

Figure 2.1 shows a diagram of connections between microcomputers.

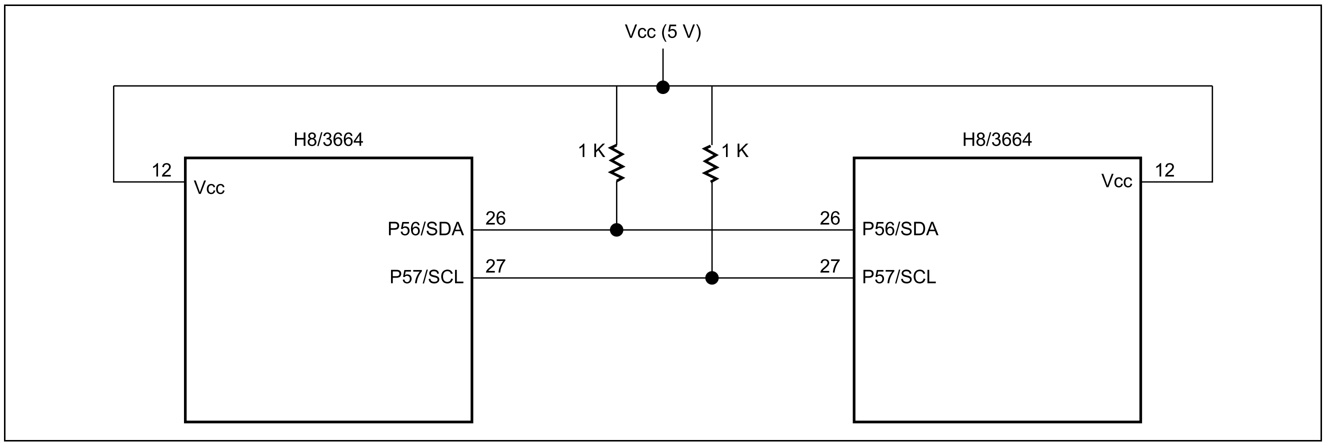


Figure 2.1 Connections between microcomputers.

H8/3687: Operating frequency 16 MHz:

Power supply voltage 5 V

3. Sample Programs

3.1 Functions

One H8 microcomputer transmits the specified length of data; when the other H8 microcomputer receives the data, it returns the same data to the microcomputer that transmitted the data.

3.2 Embedding the sample programs

1. Sample program 12-A
   Incorporate #define directives.
2. Sample program 12-B
   Incorporate prototype declarations.
3. Sample program 12-C
   Incorporate the source programs.
4. Sample program 12-D
   — 4.1 Add the I^{2}C reset vector.
   — 4.2 Add the I^{2}C initial setting processing.
   — 4.3 Add the I^{2}C common subroutine.
   — 4.4 Add the I^{2}C interrupt processing.

3.3 Modifications to sample programs

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

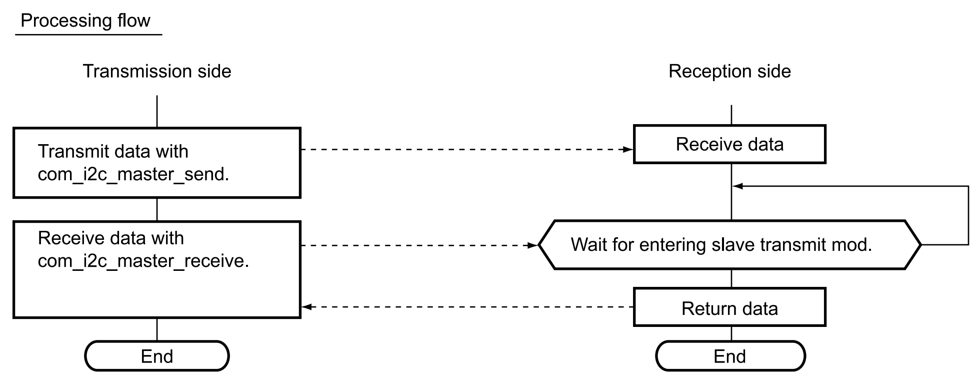
1. By using a file with definitions of IO register structures which can be obtained free of charge from the following Renesas web site,http://www.renesas.com/eng/products/mpumcu/tool/crosstool/iodef/index.html
   the sample program can be used without further changes. When creating definitions independently, the customer should modify the IO register structures used in the sample program as appropriate.
2. In order to communicate between microcomputers via I^{2}C interface, a unique device address is set for each microcomputer. Define a unique value to MICOM_ID in the sample program 12-A define definition, compile it, and write the program to each microcomputer.
3. In the sample program, TimerZ is designed to start every 10 ms and timeout after 5 seconds, in order to monitor the state of the I^{2}C interface. The timer processing can be modified according to the needs of the customer, and of course can be used without modification. When using the timer processing in the sample program without modification, the following changes should be made.
   A. Sample program 12-E
      - 5.1 The TimerZ reset vector should be added.
      - 5.2 com_timer should be added as a common variable.
      - 5.3 The TimerZ initial setting processing should be added.
        (The GRA setting should be changed according to the operating frequency of the microcomputer being used, so that the TimerZ interrupt occurs in 10 ms. For setting values, refer to the H8/3687 Hardware Manual; for the location of the setting to be changed, refer to the program notes in the sample program.)
      - 5.4 The TimerZ interrupt processing should be added.

4. The I^{2}C interface transfer rate ICCR1 (CKS3:0) should be set according to the target device specifications and the microcomputer operating frequency. Refer to the H8/3687 Hardware Manual for setting values, and to the program notes in the sample program for the location to be changed. In this sample program, the transfer rate is set to 200 kbps.

3.4 Method of use

"Data length + data" is transmitted from the transmission-side microcomputer, and the data is returned from the reception-side microcomputer. In this sample program, received data is returned without change, however, returned data can be changed by modifying the I²C reception interrupt processing. For the location to be changed, refer to the program notes in the source codes.

The transmission-side microcomputer issues com_i2c_master_send to transmit data and com_i2c_master_recive to receive response data from the reception-side device. These two are always used together as a pair.


Processing flow
Transmission side
Reception side
Transmit data with com_i2c_master_send.
Receive data with com_i2c_master_receive.
End
Receive data
Wait for entering slave transmit mod.
Return data
End

Communication packet format

2 bytes	Max. 65,535 bytes
length	Data

Example: Data length = 8 data = 0001020304050607

0008	0001020304050607

Note: This sample program does not support full-duplex transmission. Therefore, data transmission cannot be performed during data reception processing.

1. Transmit the specified length of data.

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

Argument	Explanation
slave_addr	Specifies the transmit destination device address. A unique value is set for each microcomputer as the device address.
data_length	Specifies the transmitted data length.
*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 (stop 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. Receive data returned from the transmission destination device.

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

Argument	Explanation
slave_addr	Specifies the transmission destination device address. A unique value is set for each microcomputer as the device address.
*recive_data	Specifies the address at which to store data for reception. For receive data format, refer to the first part of section 3.4, Method of use.

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 (stop condition detection timeout)

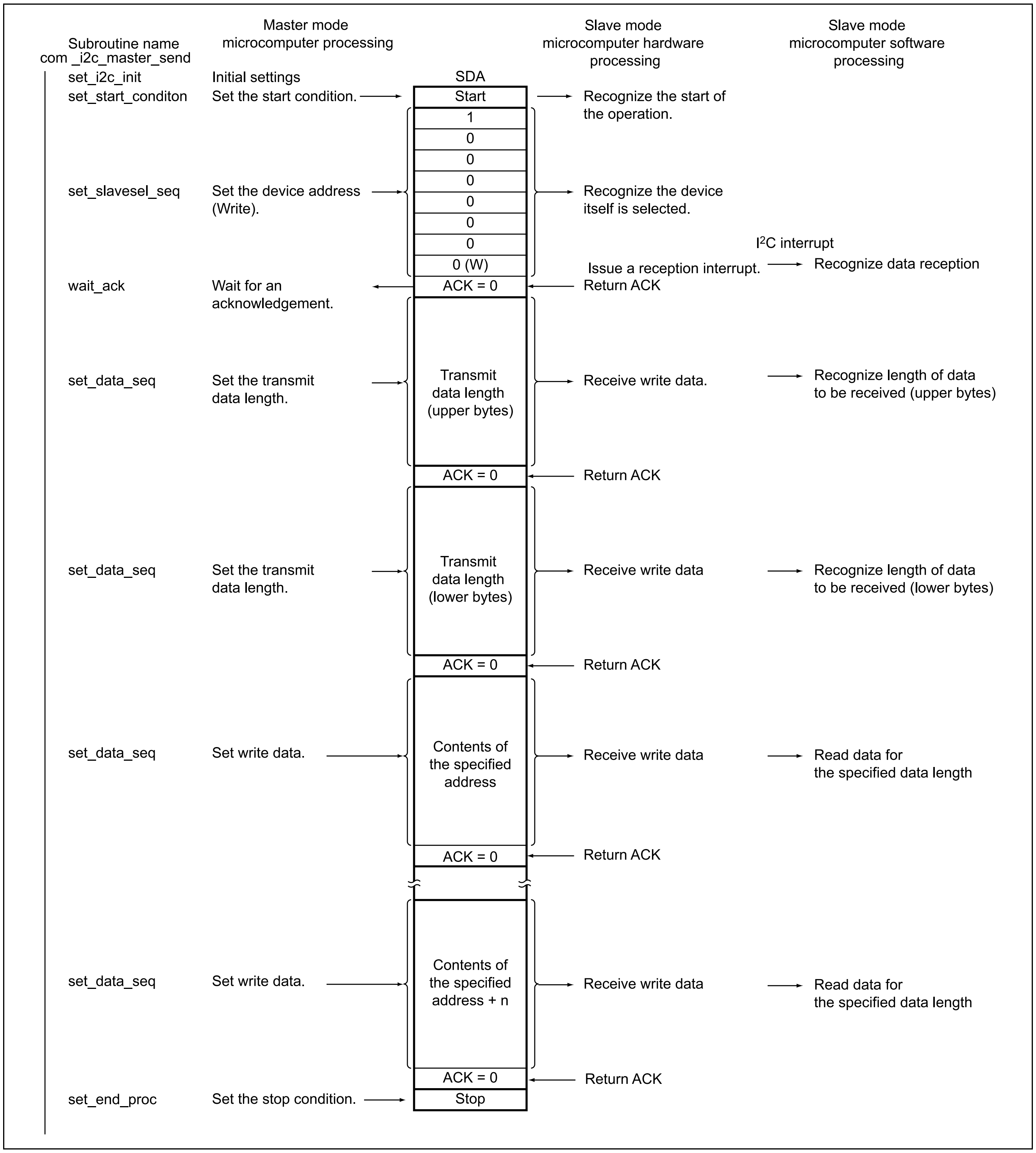
Example of use:

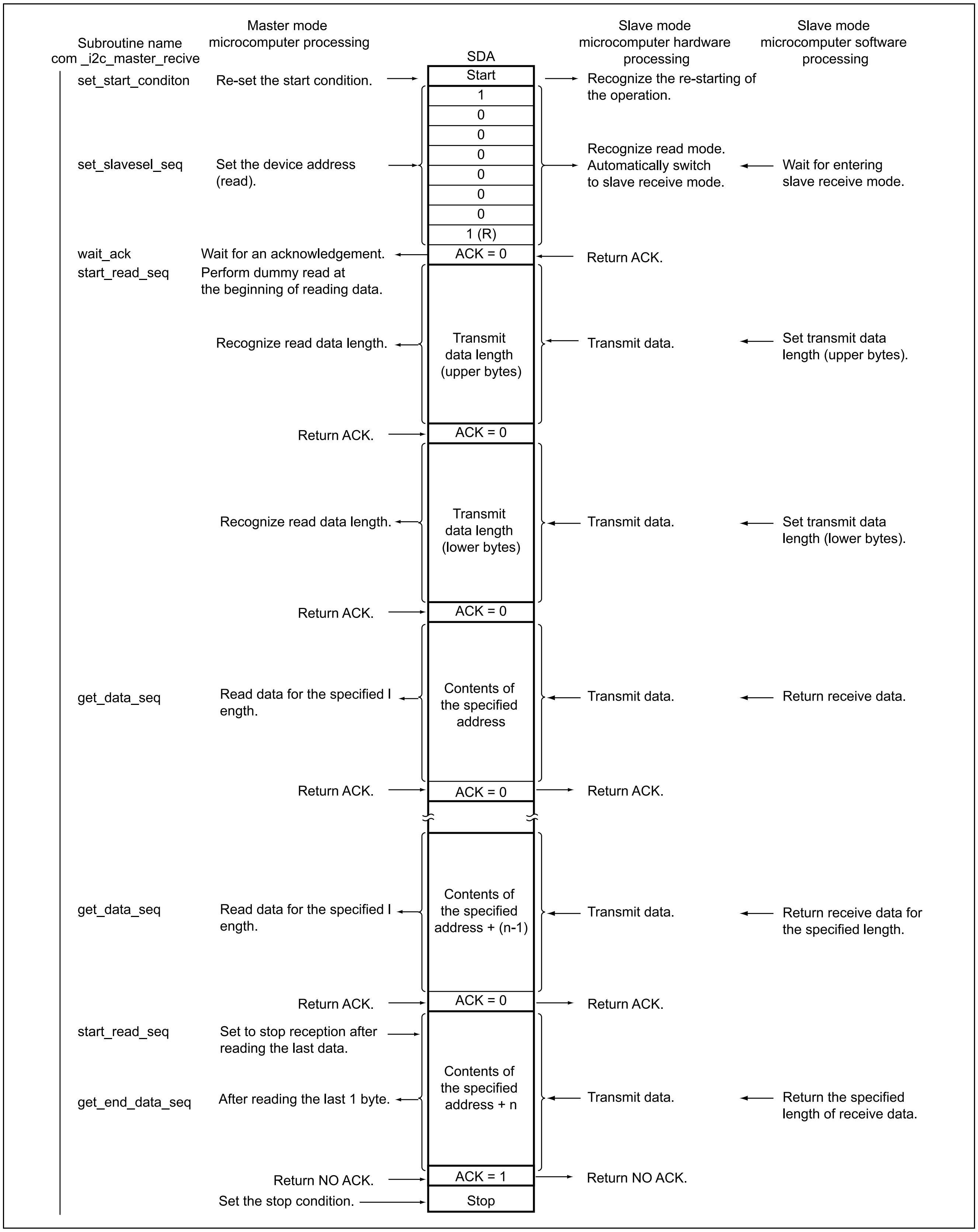
```
int ret ;
unsigned char slave_addr ;
unsigned int data_length ;
unsigned char recive_data[256] ;
ret = com_i2c_master_recive (slave_addr , data_length , &recive_data[0] )
```

3.5 Explanation of operation

Below, operation is explained.

1. Variable length data is transmitted from the transmission-side microcomputer, and after the reception-side microcomputer receives the data, it returns the same data to the transmission-side device.


Subroutine name
com _i2c_master_send
Master mode microcomputer processing
Slave mode microcomputer hardware processing
Slave mode microcomputer software processing
SDA
set_i2c_init
Initial settings
set_start_conditon
Set the start condition.
Start
Recognize the start of the operation.
1
0
0
0
0
0
0
0 (W)
set_slavesel_seq
Set the device address (Write).
Recognize the device itself is selected.
I2C interrupt
Issue a reception interrupt.
Recognize data reception
wait_ack
Wait for an acknowledgement.
ACK = 0
Return ACK
set_data_seq
Set the transmit data length.
Transmit data length (upper bytes)
Receive write data.
Recognize length of data to be received (upper bytes)
ACK = 0
Return ACK
set_data_seq
Set the transmit data length.
Transmit data length (lower bytes)
Receive write data
Recognize length of data to be received (lower bytes)
ACK = 0
Return ACK
set_data_seq
Set write data.
Contents of the specified address
Receive write data
Read data for the specified data length
ACK = 0
Return ACK
set_data_seq
Set write data.
Contents of the specified address + n
Receive write data
Read data for the specified data length
ACK = 0
Return ACK
set_end_proc
Set the stop condition.
Stop


Subroutine name
com _i2c_master_recive
Master mode microcomputer processing
SDA
Slave mode microcomputer hardware processing
Slave mode microcomputer software processing
set_start_conditon
Re-set the start condition.
Start
Recognize the re-starting of the operation.
set_slavesel_seq
Set the device address (read).
1
0
0
0
0
0
0
1 (R)
Recognize read mode. Automatically switch to slave receive mode.
Wait for entering slave receive mode.
wait_ack
Wait for an acknowledgement.
ACK = 0
Return ACK.
start_read_seq
Perform dummy read at the beginning of reading data.
Recognize read data length.
Transmit data length (upper bytes)
Transmit data.
Set transmit data length (upper bytes).
Return ACK.
ACK = 0
Recognize read data length.
Transmit data length (lower bytes)
Transmit data.
Set transmit data length (lower bytes).
Return ACK.
ACK = 0
get_data_seq
Read data for the specified l ength.
Contents of the specified address
Transmit data.
Return receive data.
Return ACK.
ACK = 0
Return ACK.
get_data_seq
Read data for the specified l ength.
Contents of the specified address + (n-1)
Transmit data.
Return receive data for the specified length.
Return ACK.
ACK = 0
Return ACK.
start_read_seq
Set to stop reception after reading the last data.
get_end_data_seq
After reading the last 1 byte.
Contents of the specified address + n
Transmit data.
Return the specified length of receive data.
Return NO ACK.
ACK = 1
Return NO ACK.
Set the stop condition.
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. I^2C-related registers

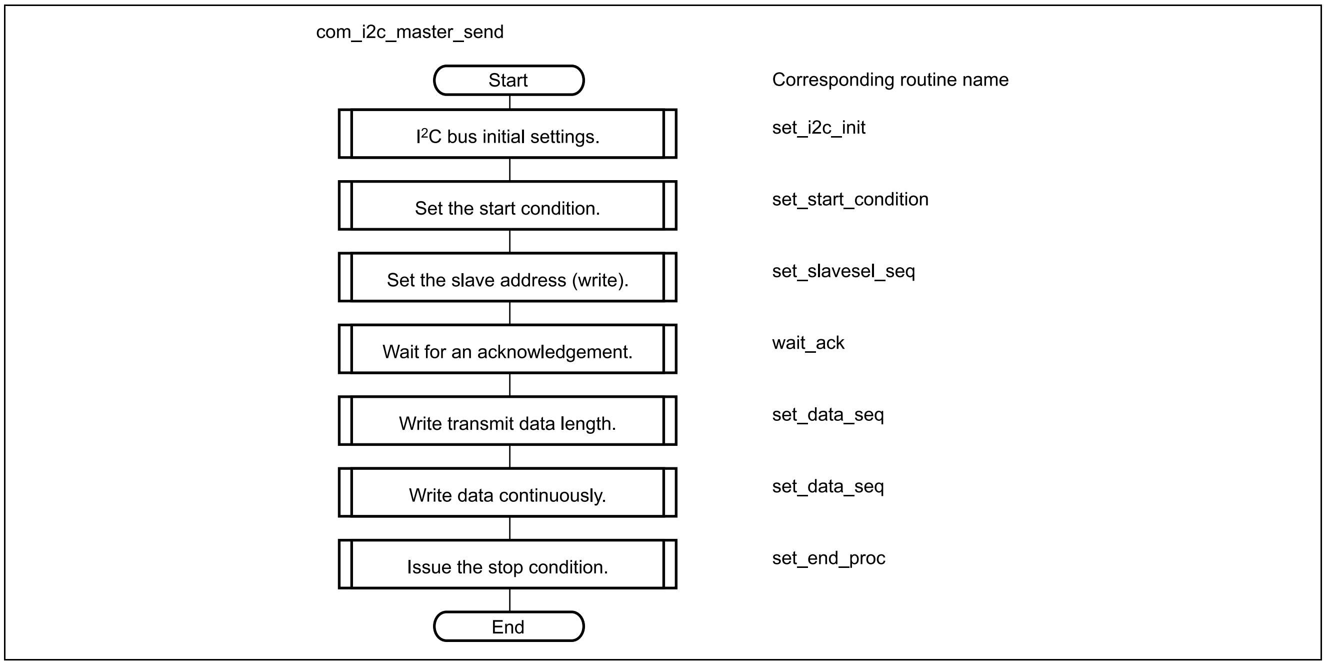
Name	Summary
I^2C bus control register 1 (ICCR1)	Starts or stops operation of the I^2C bus interface 2, controls transmission/reception, and selects master/slave mode, transmission/reception or master mode transfer clock frequency.
I^2C bus control register 2 (ICCR2)	Issues start/stop conditions, operates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

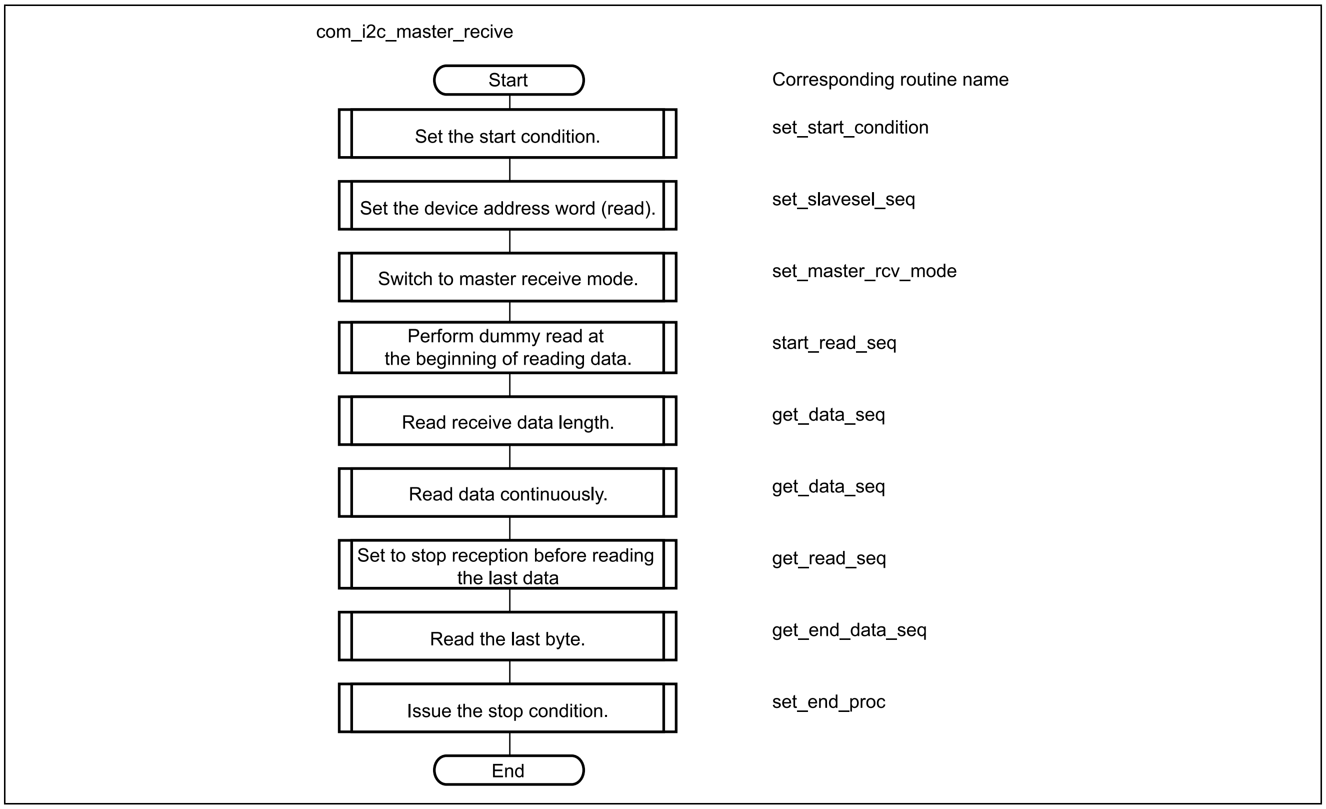
2. TimerZ-related registers
   TimerZ has various functions, but in the sample program it uses the GRA register compare-match function to generate an interrupt every 10 ms.

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

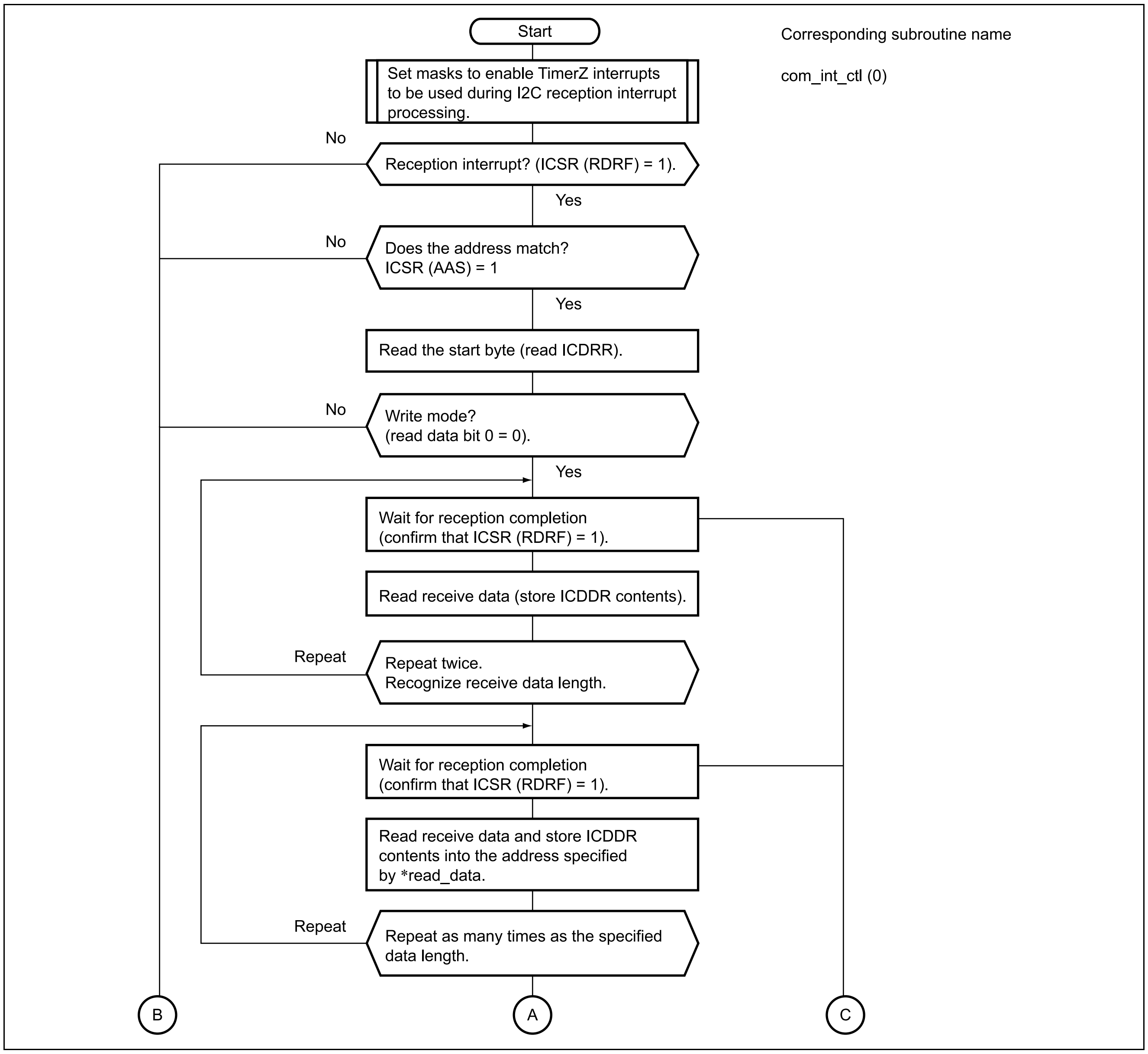
3.7 Flowcharts

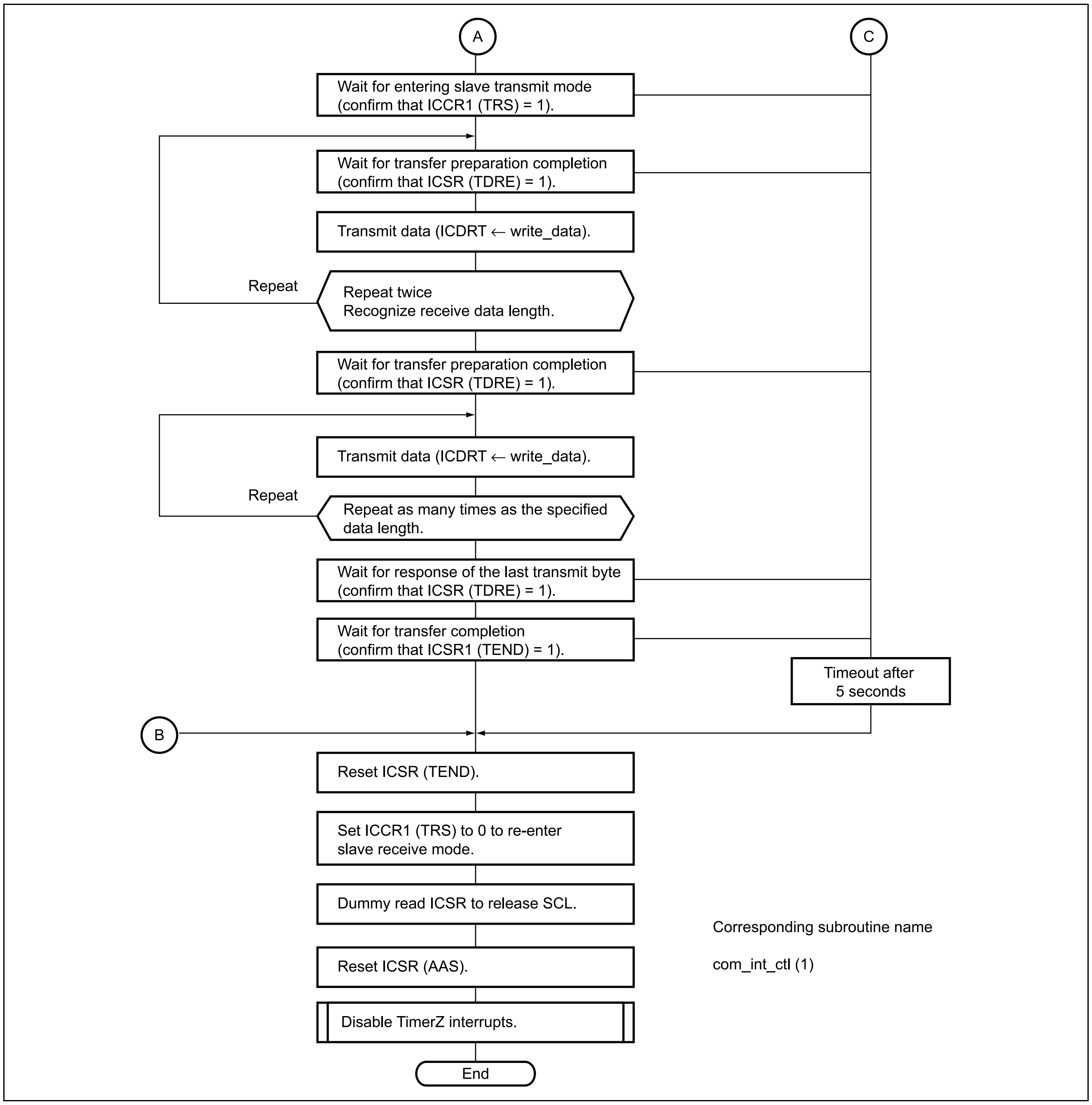
1. Transmission-side H8 microcomputer processing

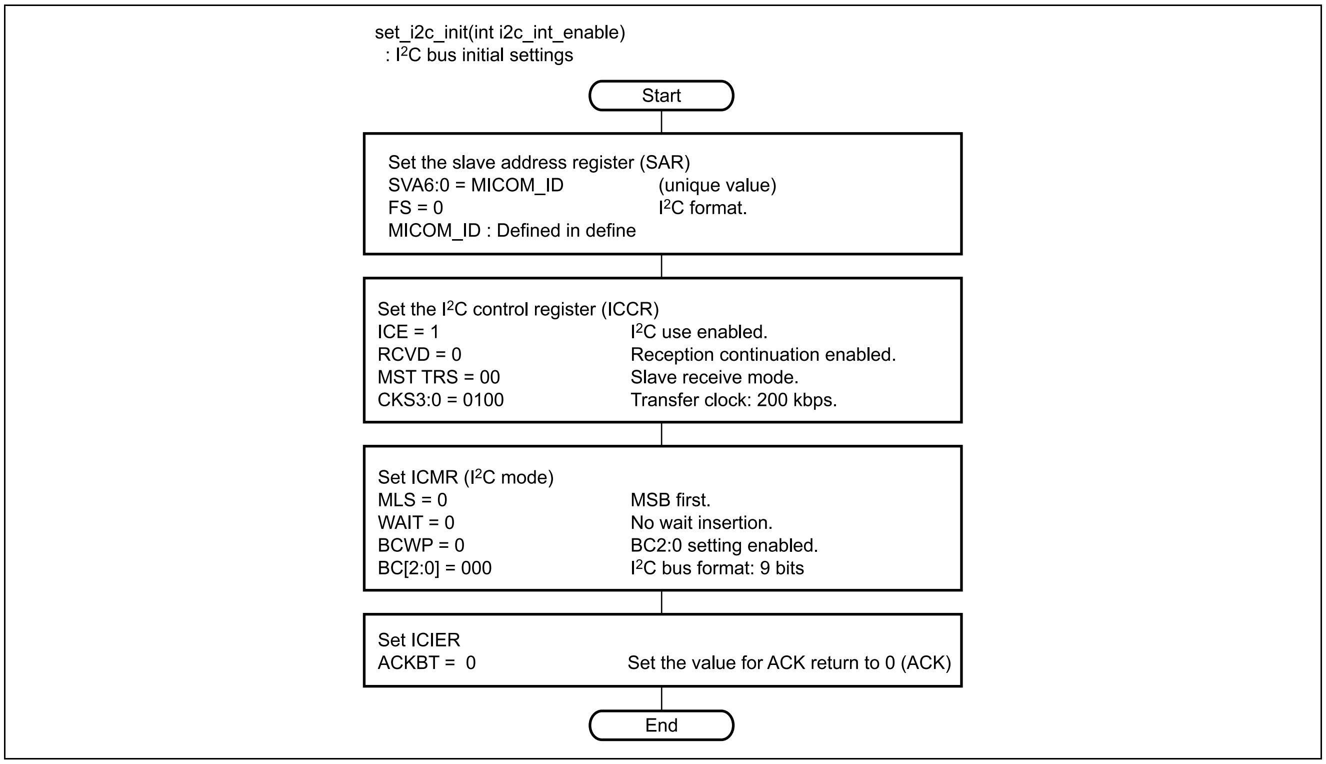


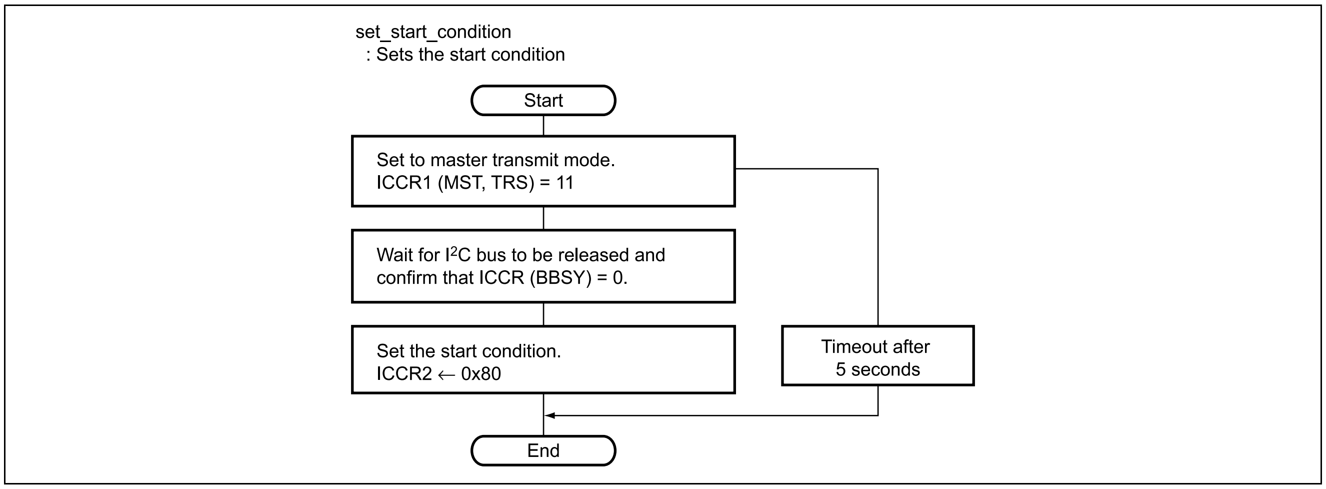

com_i2c_master_recive
Start
Set the start condition.
Set the device address word (read).
Switch to master receive mode.
Perform dummy read at the beginning of reading data.
Read receive data length.
Read data continuously.
Set to stop reception before reading the last data
Read the last byte.
Issue the stop condition.
End
Corresponding routine name
set_start_condition
set_slavesel_seq
set_master_rcv_mode
start_read_seq
get_data_seq
get_data_seq
get_read_seq
get_end_data_seq
set_end_proc

2. Reception-side microcomputer processing (I^2C interrupt processing)

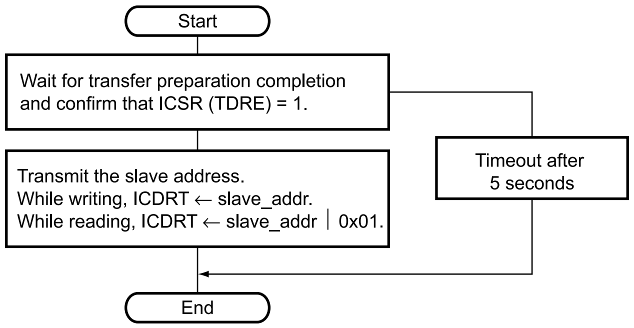

Start
Set masks to enable TimerZ interrupts to be used during I2C reception interrupt processing.
Corresponding subroutine name
com_int_ctl (0)
Reception interrupt? (ICSR (RDRF) = 1).
No
Yes
Does the address match? ICSR (AAS) = 1
No
Yes
Read the start byte (read ICDRR).
Write mode? (read data bit 0 = 0).
No
Yes
Wait for reception completion (confirm that ICSR (RDRF) = 1).
Read receive data (store ICDDR contents).
Repeat
Repeat twice. Recognize receive data length.
Wait for reception completion (confirm that ICSR (RDRF) = 1).
Read receive data and store ICDDR contents into the address specified by *read_data.
Repeat
Repeat as many times as the specified data length.
B
A
C


A
C
Wait for entering slave transmit mode (confirm that ICCR1 (TRS) = 1).
Wait for transfer preparation completion (confirm that ICSR (TDRE) = 1).
Transmit data (ICDRT ← write_data).
Repeat
Repeat twice Recognize receive data length.
Wait for transfer preparation completion (confirm that ICSR (TDRE) = 1).
Transmit data (ICDRT ← write_data).
Repeat
Repeat as many times as the specified data length.
Wait for response of the last transmit byte (confirm that ICSR (TDRE) = 1).
Wait for transfer completion (confirm that ICSR1 (TEND) = 1).
Timeout after 5 seconds
B
Reset ICSR (TEND).
Set ICCR1 (TRS) to 0 to re-enter slave receive mode.
Dummy read ICSR to release SCL.
Reset ICSR (AAS).
Disable TimerZ interrupts.
End
Corresponding subroutine name
com_int_ctl (1)

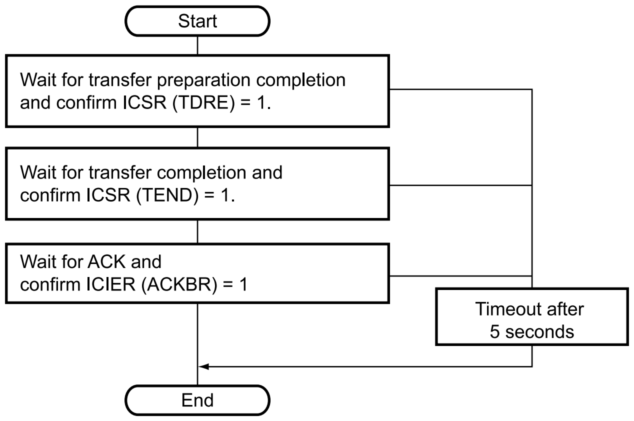

set_i2c_init(int i2c_int_enable)
: I2C bus initial settings
Start
Set the slave address register (SAR)
SVA6:0 = MICOM_ID (unique value)
FS = 0 I2C format.
MICOM_ID : Defined in define
Set the I2C control register (ICCR)
ICE = 1 I2C use enabled.
RCVD = 0 Reception continuation enabled.
MST TRS = 00 Slave receive mode.
CKS3:0 = 0100 Transfer clock: 200 kbps.
Set ICMR (I2C mode)
MLS = 0 MSB first.
WAIT = 0 No wait insertion.
BCWP = 0 BC2:0 setting enabled.
BC[2:0] = 000 I2C bus format: 9 bits
Set ICIER
ACKBT = 0 Set the value for ACK return to 0 (ACK)
End


set_start_condition
: Sets the start condition
Start
Set to master transmit mode.
ICCR1 (MST, TRS) = 11
Wait for I2C bus to be released and
confirm that ICCR (BBSY) = 0.
Set the start condition.
ICCR2 ← 0x80
Timeout after
5 seconds
End

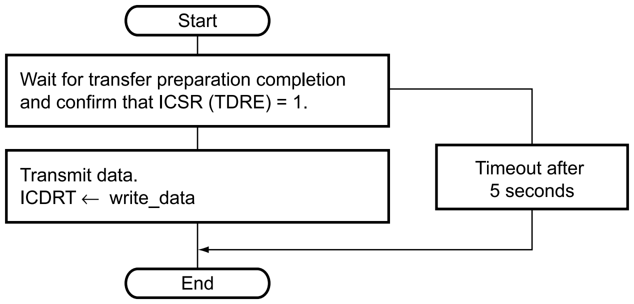
set_slavesel_seq (unsigned char mode, unsigned char slave_addr)
: Executes slave selection processing
Mode: Write or read
0: Write, 1: Read
slave_addr: EEPROM device address

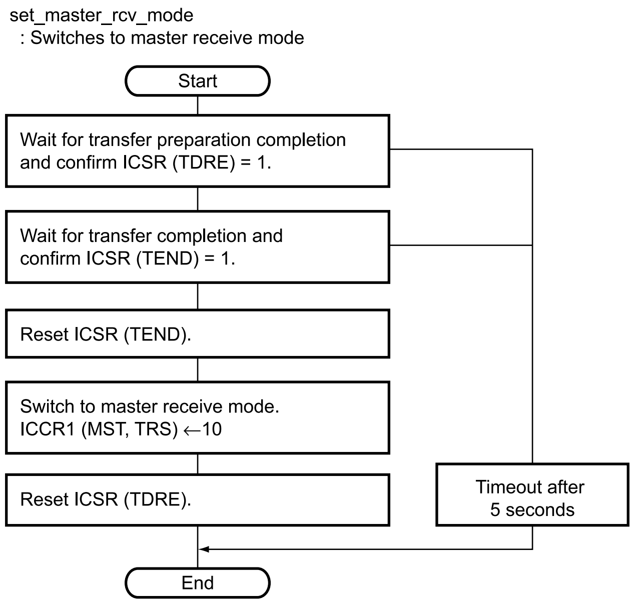

Start
Wait for transfer preparation completion and confirm that ICSR (TDRE) = 1.
Transmit the slave address.
While writing, ICDRT ← slave_addr.
While reading, ICDRT ← slave_addr | 0x01.
Timeout after 5 seconds
End

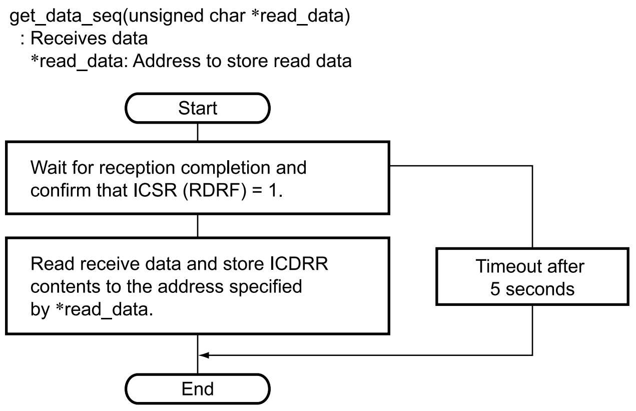
Wait_ack
: Waits for an acknowledgement

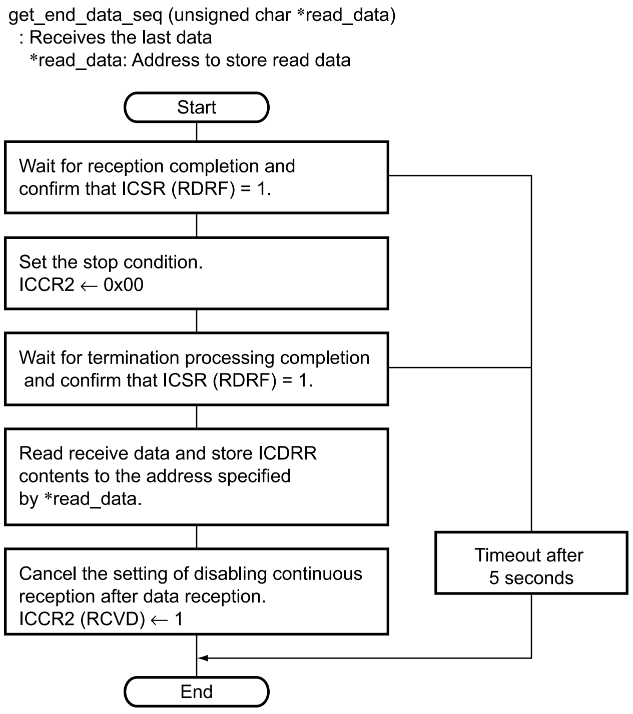

Start
Wait for transfer preparation completion and confirm ICSR (TDRE) = 1.
Wait for transfer completion and confirm ICSR (TEND) = 1.
Wait for ACK and confirm ICIER (ACKBR) = 1
Timeout after 5 seconds
End

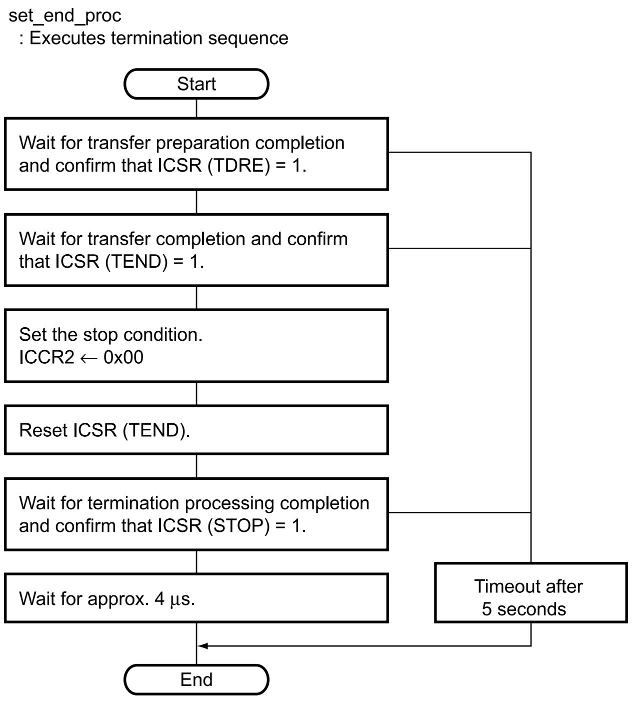
set_data_seq(unsigned char write_data)
: Sets data
Write_data: Data to be transmitted


Start
Wait for transfer preparation completion and confirm that ICSR (TDRE) = 1.
Transmit data.
ICDRT ← write_data
Timeout after 5 seconds
End

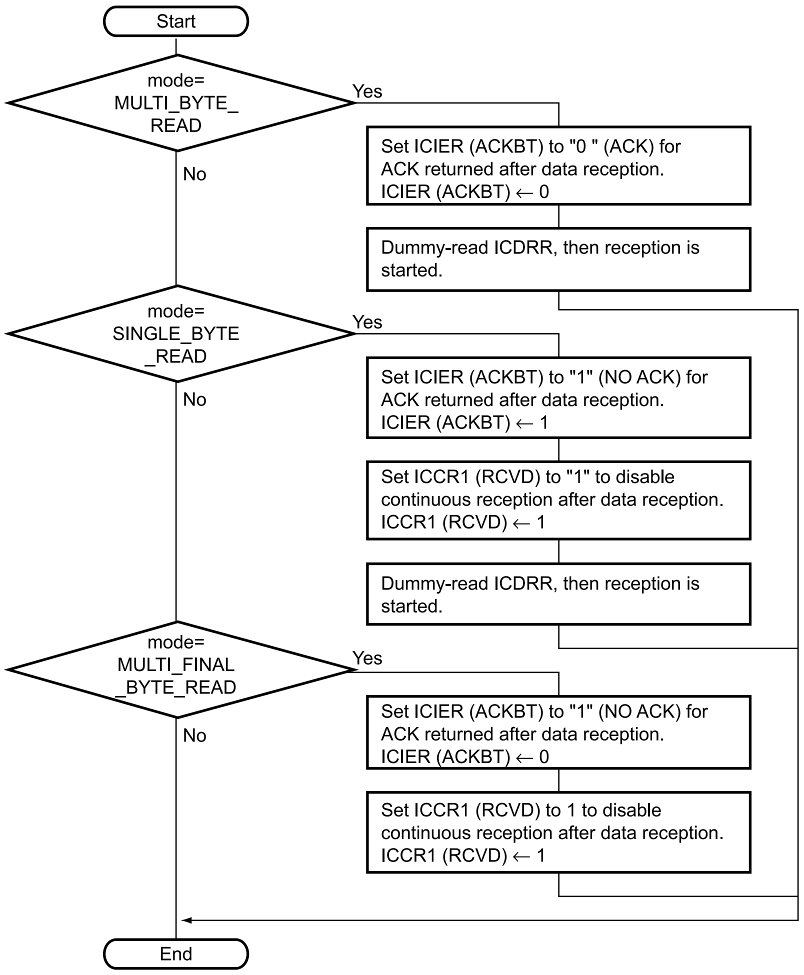

set_master_rcv_mode
: Switches to master receive mode
Start
Wait for transfer preparation completion and confirm ICSR (TDRE) = 1.
Wait for transfer completion and confirm ICSR (TEND) = 1.
Reset ICSR (TEND).
Switch to master receive mode. ICCR1 (MST, TRS) ←10
Reset ICSR (TDRE).
Timeout after 5 seconds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


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


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


set_end_proc
: Executes termination sequence
Start
Wait for transfer preparation completion and confirm that ICSR (TDRE) = 1.
Wait for transfer completion and confirm that ICSR (TEND) = 1.
Set the stop condition. ICCR2 ← 0x00
Reset ICSR (TEND).
Wait for termination processing completion and confirm that ICSR (STOP) = 1.
Wait for approx. 4 µs.
Timeout after 5 seconds
End

start_read_seq
: Execute preprocessing for data reception


Start
mode= MULTI_BYTE_ READ
Yes
No
Set ICIER (ACKBT) to "0 " (ACK) for ACK returned after data reception. ICIER (ACKBT) ← 0
Dummy-read ICDRR, then reception is started.
mode= SINGLE_BYTE _READ
Yes
No
Set ICIER (ACKBT) to "1" (NO ACK) for ACK returned after data reception. ICIER (ACKBT) ← 1
Set ICCR1 (RCVD) to "1" to disable continuous reception after data reception. ICCR1 (RCVD) ← 1
Dummy-read ICDRR, then reception is started.
mode= MULTI_FINAL _BYTE_READ
Yes
No
Set ICIER (ACKBT) to "1" (NO ACK) for ACK returned after data reception. ICIER (ACKBT) ← 0
Set ICCR1 (RCVD) to 1 to disable continuous reception after data reception. ICCR1 (RCVD) ← 1
End

3.8 Program Listing

```
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* 1. Sample Program 12-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 code (anything 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

/********************************************************************************************************************************/
/*   Device number                                                                                                              */
/*   Specify unique values for communication among microcomputers.                                                              */
/********************************************************************************************************************************/

#define MICOM_ID         0x80
//#define MICOM_ID       0x40
          /* ##(program note)################################################################################################## */
          /* ## Specifies unique addresses on microcomputers for bi-directional communication via the I2C interface.         ## */
          /* ## Bit 0 cannot be used.                                                                                        ## */
          /* ################################################################################################################## */

/* ---------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* 2. Sample program 12-B  Variable definition -------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
     /***************************************************************************************************************************/
     /*   I2C communication buffer                                                                                              */
     /***************************************************************************************************************************/
     #ifndef        _DEFINE_COMMON_TABLE
          extern
     #endif                                                            /* _DEFINE_COMMON_TABLE                                  */
          unsigned char        com_i2c_packet[256]  ;
```

```
/* -------------------------------------------------------------------------------------------------------------------------- */
/* -------------------------------------------------------------------------------------------------------------------------- */
/*. Sample program 12-C  Prototype declaration ------------------------------------------------------------------------------  */
/* -------------------------------------------------------------------------------------------------------------------------- */
/* -------------------------------------------------------------------------------------------------------------------------- */
/****************************************************************************************************************************/
/****************************************************************************************************************************/
/*   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 char *recive_data ) ;
```

```
/* ------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/* 4. Sample program 12-D  Source codes ---------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------ */

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

/* ------------------------------------------------------------------------------------------------------------------------------ */
/* 4.2 i2C initial settings ---------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/*   Add to the initial settings for registers.                                                                                  */
     /* ######################################################################################################################## */
     /* ######################################################################################################################## */
     /*                                                                                                                           */
     /*        Sets I2C bus                                                                                                       */
     /*                                                                                                                           */
     /* ######################################################################################################################## */
     /* ######################################################################################################################## */

     /**************************************************************************************************************************/
     /*   SAR            Sets the slave address register                                                                         */
     /*   ICCR1          Sets the I2C control register                                                                           */
     /*   ICMR           Sets I2C mode                                                                                           */
     /**************************************************************************************************************************/
     set_i2c_init () ;

     /**************************************************************************************************************************/
     /*   ICIER          Sets I2C interrupts                                                                                     */
     /*        TIE       = 0  Transmit interrupts enabled                                                                        */
     /*        TEIE      = 0  Transmit end interrupts enabled                                                                    */
     /*        RIE       = 1  Receive interrupts enabled                                                                         */
     /*        NAKIE     = 0  NACK receive interrupts enabled                                                                    */
     /*        STIE      = 0  Stop condition detection interrupts enabled                                                        */
     /*        ACKE      = 0  Selection for acknowledgement judgments                                                            */
     /*        ACKBR     = 0  Reception acknowledgement                                                                          */
     /*        ACKBT     = 0  Transmission acknowledgement (ACK = 0)                                                             */
     /**************************************************************************************************************************/
     IIC2.ICIER.BYTE= 0x20 ;                                       /* Makes receive interrupts valid                         */
```

```
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* 4.3 Common subroutines ------------------------------------------------------------------------------------------------------ */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/******************************************************************************************************************************/
/******************************************************************************************************************************/
/******************************************************************************************************************************/
/*                                                                                                                            */
/*                                I2C EEPROM control                                                                          */
/*                                                                                                                            */
/******************************************************************************************************************************/
/******************************************************************************************************************************/
/******************************************************************************************************************************/

/******************************************************************************************************************************/
/*   1. Module name : set_i2c_init                                                                                            */
/*   2. Function overview : Sets initial settings prior to I2 access                                                          */
/******************************************************************************************************************************/
void set_i2c_init( )
{
     /*************************************************************************************************************************/
     /*   SAR            Sets the slave address register                                                                      */
     /*        SVA6:0    = MICOM_ID (unique value)                                                                            */
     /*        FS        = 0  I2C format                                                                                      */
     /*************************************************************************************************************************/
          /* ##(program note)################################################################################################## */
          /* ## Specifies unique addresses on microcomputers for bi-directional communication via the I2C interface.         ## */
          /* ## Bit 0 cannot be used.                                                                                         ## */
          /* ################################################################################################################## */

     IIC2.SAR.BYTE= MICOM_ID & 0xFE ;

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

     IIC2.ICCR1.BYTE     = 0x84 ;
          /* ##(program note)################################################################################################## */
          /* ## CK The value set for CKS3:0 should be changed based on the necessary 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 ;

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

}
```

```
/*******************************************************************************************************************************/
/*   1. Module name : set_start_condition                                                                                     */
/*   2. Function overview : Sets the I2C start condition.                                                                     */
/*******************************************************************************************************************************/
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){                                          /* Timeouts after 5 seconds.                           */
               ret = I2C_BBSY_TOUT;                                    /* Ended abnormally (timeout)                          */
               goto exit ;
          }

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

     }

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

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

          /* ##(program note)################################################################################################## */
          /* ## The settings for bits 7 and 6, which set the start condition, have to be set simultaneously,                ## */
          /* ## Theso they must be written in byte units                                                                    ## */
          /* ## Please 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.                                             */
/*****************************************************************************************************************/
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){                                     /* Timeouts after 5 seconds.               */
               ret = I2C_TDRE_TOUT;                               /* Ended abnormally (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.                                                               */
/*****************************************************************************************************************/
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){                                     /* Timeouts after 5 seconds.               */
               ret = I2C_TDRE_TOUT;                               /* Ended abnormally (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){                                          /* Timeouts after 5 seconds.                           */
               ret = I2C_TEND_TOUT;                                    /* Ended abnormally (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){                                          /* Timeouts after 5 seconds.                           */
               ret = I2C_ACKBR_TOUT;                                   /* Ended abnormally (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                                                               */
/*******************************************************************************************************************************/
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){                                          /* Timeouts after 5 seconds.                           */
               ret = I2C_TDRE_TOUT;                                    /* Ended abnormally (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 the master receive mode                                                               */
/******************************************************************************************************************************/
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){                                          /* Timeouts after 5 seconds.                           */
               ret = I2C_TDRE_TOUT;                                    /* Ended abnormally (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){                                          /* Timeouts after 5 seconds.                           */
               ret = I2C_TEND_TOUT;                                    /* Ended abnormally (timeout)                          */
               goto exit ;
          }

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

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

     /*********************************************************************************************************************/
     /*   Switches to the 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                                      */
/**************************************************************************************************************************/
void start_read_seq (unsigned char mode)
{
     int ret , Timer_wk;
     unsigned char dummy_data ;

     ret = NORMAL_END ;

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

          /****************************************************************************************************************/
          /*   Initiates reception when a dummy read is carried out                                                       */
          /****************************************************************************************************************/
          dummy_data = IIC2.ICDRR ;
```

```
                /* ##(program note)############################################################################################ */
                /* ## Reception begins when a dummy read is carried out, and data is sent                                  ## */
                /* ## from the device synchronized to the SCL.                                                            ## */
                /* ## A low level signal is sent to the device synchronized to the ninth SCL,                             ## */
                /* ## in response to the ICSR (ACKB) 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 continuous reception after data reception                                                               */
            /**********************************************************************************************************************/
            IIC2.ICCR1.BIT.RCVD = 1  ;

            /**********************************************************************************************************************/
            /*   Initiates reception when a dummy read is carried out                                                             */
            /**********************************************************************************************************************/
            dummy_data = IIC2.ICDRR ;
                /* ##(program note)############################################################################################ */
                /* ## Reception begins when a dummy read is carried out, and data is sent                                  ## */
                /* ## from the device synchronized to the SCL.                                                            ## */
                /* ## A high level signal is sent to the device synchronized to the ninth SCL,                            ## */
                /* ## 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 continuous reception after data reception                                                               */
            /**********************************************************************************************************************/
            IIC2.ICCR1.BIT.RCVD = 1  ;
        }
    }
```

```
/**********************************************************************************************************************/
/*   1. Module name : get_data_seq                                                                                    */
/*   2. Function overview : Reads data from the I2C target device                                                     */
/**********************************************************************************************************************/
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){                                           /* Timeouts after 5 seconds.                           */
               ret = I2C_RDRF_TOUT;                                     /* Ended abnormally (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                                                     */
/**********************************************************************************************************************/
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){                                           /* Timeouts after 5 seconds.                           */
               ret = I2C_RDRF_TOUT;                                     /* Ended abnormally (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){                                            /* Timeouts after 5 seconds.                            */
              ret = I2C_STOP_TOUT;                                      /* Ended abnormally (timeout)                           */
              goto exit ;
         }

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

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

    /******************************************************************************************************************************/
    /*   Cancels the disable setting of continuous reception after 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 exit sequence                                                                       */
/*******************************************************************************************************************************/
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 een completed.                     */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                           /* Timeouts after 5 seconds.                           */
               ret = I2C_TDRE_TOUT;                                     /* Ended abnormally (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){                                           /* Timeouts after 5 seconds.                           */
               ret = I2C_TEND_TOUT;                                     /* Ended abnormally (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){                                        /* Timeouts after 5 seconds.                           */
               ret = I2C_STOP_TOUT;                                  /* Ended abnormally (timeout)                          */
               goto exit ;
          }

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

exit :
     return (ret);

}
/**************************************************************************************************************************/
/*   1. Module name : wait_write_end                                                                                      */
/*   2. Function overview : Checks the end of I2C write                                                                   */
/**************************************************************************************************************************/
unsigned int wait_write_end (unsigned char slave_addr )
{
     int ret , Timer_wk;

     ret = NORMAL_END ;

     com_timer.wait_100ms = 50 ;

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

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


          /****************************************************************************************************************/
          /*   Confirms that ICSR (TDRE)=1.                                                                              */
          /****************************************************************************************************************/
          com_timer.wait_100ms_scan = 50 ;
          while (IIC2.ICSR.BIT.TDRE == 0){                           /* Waits until the preparation                         */
                                                                     /* for transfer has been completed.                    */
               Timer_wk = com_timer.wait_100ms_scan ;
               if (Timer_wk == 0){                                   /* Timeouts after 5 seconds.                           */
                    ret = I2C_TDRE_TOUT;                             /* Ended abnormally (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){                                    /* Timeouts after 5 seconds.                           */
                    ret = I2C_TEND_TOUT;                              /* Ended abnormally (timeout)                          */
                    goto exit ;
               }

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

          }

          /*****************************************************************************************************************/
          /*   I Checks ICIER (ACKBR): ACK = 0  The write has completed. ACK = 1  The write is being performed.            */
          /*****************************************************************************************************************/
          if (com_timer.wait_100ms == 0){                             /* If this remains 1 for 5 seconds,                    */
                                                                      /* it can be escaped by an error being generated.      */
               ret = I2C_ACKBR_TOUT;                                  /* Ended abnormally (timeout)                          */
               goto exit ;
          }

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

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

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

exit :
     return (ret);

}
```

```
/*****************************************************************************************************************************/
/*   1. Module name : com_i2c_master_send                                                                                  */
/*   2. Function overview : Transfers the specified length of data from the master to a slave device                       */
/*****************************************************************************************************************************/
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 ;}

     /************************************************************************************************************************/
     /*   Transmits the data length                                                                                          */
     /************************************************************************************************************************/
     buf.d_int = data_length  ;
     ret = set_data_seq ( buf.d_byte[0] ) ;
     if (ret !=0) { goto exit ;}

    ret = set_data_seq ( buf.d_byte[1] ) ;
     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 :                                                                /* Error processing                               */
     /*******************************************************************************************************************/
     /*   Resets the I2C and issues the stop condition if an error occurs                                               */
     /*******************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x02 ;                                     /* Resets the I2C control                         */
     IIC2.ICCR2.BYTE     = 0x00 ;                                     /* Sets the stop condition                        */

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

     return (ret);

}

/************************************************************************************************************************/
/*   1. Module name : com_i2c_master_recive                                                                             */
/*   2. Function overview : Receives the specified length of data from the slave device                                 */
/************************************************************************************************************************/
unsigned int com_i2c_master_recive ( unsigned char slave_addr ,unsigned char *recive_data )
{
     int ret , i , length  ;
     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 the master receive mode                                                                           */
     /*******************************************************************************************************************/
     ret = set_master_rcv_mode () ;
     if (ret !=0) { goto exit ;}
     /*******************************************************************************************************************/
     /*   Carries out a dummy read at the start of data reading                                                         */
     /*******************************************************************************************************************/
     start_read_seq ( MULTI_BYTE_READ ) ;
```

```
    /**************************************************************************************************************************/
    /*   Recognizes the first two bytes of read data.                                                                         */
    /**************************************************************************************************************************/
    ret = get_data_seq ( &buf.d_byte[0] ) ;
    if (ret !=0) { goto exit ;}
    ret = get_data_seq ( &buf.d_byte[1] ) ;
    if (ret !=0) { goto exit ;}

    *recive_data = buf.d_byte[0] ;
    *recive_data ++ ;
    *recive_data = buf.d_byte[1] ;
    *recive_data ++ ;

    length = buf.d_int ;

    /* ##(program note)####################################################################################################### */
    /* ## Check the data length not to exceed the data buffer specified by recive_data.                                    ## */
    /* ###################################################################################################################### */

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

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

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

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

    *recive_data = buf.d_byte[0] ;

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

    return (ret);

exit :                                                                     /* Error processing                                      */
```

```
     /**********************************************************************************************************************/
     /*   Resets the I2C and issues the stop condition if an error occurs                                                  */
     /**********************************************************************************************************************/
     IIC2.ICCR2.BYTE     = 0x02 ;                                /* Resets I2C control                                    */
     IIC2.ICCR2.BYTE     = 0x00 ;                                /* Sets the stop condition                               */

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

     return (ret);

}

/***************************************************************************************************************************/
/*   1. Module name : com_int_ctl                                                                                          */
/*   2. Function overview : Clears set_imask_ccr to 0, to make only the TimerZ interrupt valid.                            */
/***************************************************************************************************************************/
void com_int_ctl (unsigned char kind)
{

     if (kind == 0){
          /*****************************************************************************************************************/
          /*   Disables SCI3 receive interrupts                                                                            */
          /*****************************************************************************************************************/
          SCI3.SCR3.BYTE      = 0x10;                            /* RCV int disable                                       */

          /*****************************************************************************************************************/
          /*   Disables I2C receive interrupts                                                                             */
          /*****************************************************************************************************************/
          IIC2.ICIER.BIT.RIE  = 0 ;                              /* Disables I2C receive interrupts                       */

          /*****************************************************************************************************************/
          /*   Makes TimerZ interrupt valid                                                                                */
          /*****************************************************************************************************************/
          TZ0.TIER.BIT.IMIEA  = 1 ;                              /* timerZ IMFA  enable                                   */

          /*****************************************************************************************************************/
          /*   Cancels interrupt disable                                                                                   */
          /*****************************************************************************************************************/
          set_imask_ccr(0);                                      /* Enables interrupts                                    */
     }
     else{
          /*****************************************************************************************************************/
          /*   Sets interrupt disable (reason: to prevent other interrupts from coming in while an interrupt is being processed)*/
          /*****************************************************************************************************************/
          set_imask_ccr(1);                                      /* Disables interrupts                                   */
          /*****************************************************************************************************************/
          /*   Enables IREQ0-3 and SCI3 receive interrupts                                                                 */
          /*****************************************************************************************************************/
          SCI3.SCR3.BYTE      = 0x50;                            /* Enables REV int only                                  */
          /*********************************************************************************************************************/
          /*   Enables I2C receive interrupts                                                                                  */
          /*********************************************************************************************************************/
          IIC2.ICIER.BIT.RIE  = 1 ;                               /* Enables I2C receive interrupts                         */

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

     }
}


/* ------------------------------------------------------------------------------------------------------------------------ */
/* 4.4 i2C interrupt processing ------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------ */
/******************************************************************************************************************************/
/*   1. Module name : h8_i2c                                                                                                  */
/*   2. Function overview : Interrupts from the I2C bus                                                                       */
/******************************************************************************************************************************/
#pragma interrupt( h8_i2c )
void h8_i2c ( void )
{
     int i , j , timer_wk;
     unsigned int ret ;
     unsigned char        slave_addr , dummy_data ;

     union {
      unsigned int       d_int ;
      unsigned char      d_byte[2];
     } length;



     ret = NORMAL_END ;

     memset( com_i2c_packet,0,256 );
     /************************************************************************************************************************/
     /* Clears set_imask_ccr to 0 to close the IREQ0-3 and SCI rcvint masks.                                                 */
     /* Makes TimerZ interrupts valid                                                                                        */
     /************************************************************************************************************************/
     com_int_ctl(0) ;                                             /* Clears ccr to 0 to make only timerZ interrupts valid */

     /************************************************************************************************************************/
     /*   Checks received interrupts                                                                                         */
     /************************************************************************************************************************/
     if (IIC2.ICSR.BIT.RDRF == 1){                                /* Reception                                              */
          if (IIC2.ICSR.BIT.AAS == 1){                            /* Slave address matching                                 */
               /*******************************************************************************************************************/
               /*   Receives slave_addr and r/w                                                                                   */
               /*******************************************************************************************************************/
               slave_addr = IIC2.ICDRR ;
```

```
/*****************************************************************************************************/
/*****************************************************************************************************/
/*****************************************************************************************************/
/*   Receives the packet in the following format.                                                    */
/*        Packet length: 2 bytes                                                                     */
/*        data       : Max 256 byte                                                                  */
/*        A packet of the longer packet length can be received by increasing the packet size         */
/*        by common variable com_i2c_packet.                                                         */
/*****************************************************************************************************/
/*****************************************************************************************************/
/*****************************************************************************************************/

if ((slave_addr & 0x01) == 0){                          /* write                                     */
     /************************************************************************************************/
     /*   Receives the data length (2 bytes)                                                         */
     /************************************************************************************************/
     for (i=0; i< 2 ; 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,          */
                                                        /* it can be escaped by an error being generated.      */
                    ret = I2C_RDRF_TOUT;                /* Ended abnormally (timeout)                */
                    goto exit ;
               }

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

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

     /************************************************************************************************/
     /*   Receives the specified length of data                                                      */
     /************************************************************************************************/
     for (i=0; i< length.d_int ; 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,          */
                                                        /* it can be escaped by an error being generated.      */
                    ret = I2C_RDRF_TOUT;                /* Ended abnormally (timeout)                */
                    goto exit ;
               }

               #ifdef UT
                    IIC2.ICSR.BIT.RDRF = 1 ;
               #endif
          }
          /*******************************************************************************************/
          /*   Reads received data                                                                   */
          /*******************************************************************************************/
          com_i2c_packet[i] = IIC2.ICDRR ;              /* data read                                 */
     }
```

```
/*********************************************************************************************************/
/*********************************************************************************************************/
/*   Performs the processing for the received command.                                                   */
/*********************************************************************************************************/
/*********************************************************************************************************/
     /* ##(program note)################################################################################## */
     /* ## Performs the processing for the received data. After data processing,                      ## */
     /* ## the data processing result is returned in the following data transmit processing.          ## */
     /* ################################################################################################## */

com_cnsl_msg("\n") ;
com_cnsl_msg("length =%02X\n",length.d_int) ;
com_cnsl_msg("data   = ") ;

for (i=0; i< 16 ; i++){
     com_cnsl_msg("%02X%02X%02X%02X %02X%02X%02X%02X %02X%02X%02X%02X %02X%02X%02X%02X \n"
          ,com_i2c_packet[i*16+ 0],com_i2c_packet[i*16+ 1],com_i2c_packet[i*16+ 2],com_i2c_packet[i*16+ 3]
          ,com_i2c_packet[i*16+ 4],com_i2c_packet[i*16+ 5],com_i2c_packet[i*16+ 6],com_i2c_packet[i*16+ 7]
          ,com_i2c_packet[i*16+ 8],com_i2c_packet[i*16+ 9],com_i2c_packet[i*16+10],com_i2c_packet[i*16+11]
          ,com_i2c_packet[i*16+12],com_i2c_packet[i*16+13],com_i2c_packet[i*16+14],com_i2c_packet[i*16+15]) ;
}

/*********************************************************************************************************/
/*   Confirms that ICCR1 (TRS) = 1                                                                       */
/*********************************************************************************************************/
com_timer.wait_100ms_scan = 50 ;
while (IIC2.ICCR1.BIT.TRS == 0){                    /* Waits until the system enters                     */
                                                    /* the slavec transmit mode                          */
     timer_wk = com_timer.wait_100ms_scan  ;
     if (timer_wk == 0){                            /* If this remains 1 for 5 seconds,                  */
                                                    /* it can be escaped by an error being generated.    */
          ret = I2C_TRS_TOUT;                       /* Ended abnormally (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.       ## */
     /* ################################################################################################## */
```

```
/*************************************************************************************************************/
/*************************************************************************************************************/
/*   Returns the received data directly as a response for the data reception.                               */
/*        Packet length: 2 bytes                                                                             */
/*        Data: 256 bytes max.                                                                               */
/*        A packet of the longer packet length can be received by increasing the packet size                */
/*        by common variable com_i2c_packet.                                                                 */
/*************************************************************************************************************/
/*************************************************************************************************************/

/*************************************************************************************************************/
/*   Transmit the data length (2 bytes)                                                                      */
/*************************************************************************************************************/
for (i=0; i< 2 ; i++){
     /********************************************************************************************************/
     /*   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,                    */
                                                     /* it can be escaped by an error being generated.      */
               ret = I2C_TDRE_TOUT;                  /* Ended abnormally (timeout)                          */
               goto exit ;
          }

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

     }

     /********************************************************************************************************/
     /*   Sets data                                                                                          */
     /********************************************************************************************************/
     IIC2.ICDRT = length.d_byte[i] ;
}

/*************************************************************************************************************/
/*   Transmits the specified length of data                                                                  */
/*************************************************************************************************************/
for (i=0; i< length.d_int ; i++){
     /********************************************************************************************************/
     /*   Confirms that ICSR (TDRE) = 1.                                                                     */
     /********************************************************************************************************/
     com_timer.wait_100ms_scan = 50 ;
     while (IIC2.ICSR.BIT.TDRE == 0){                /* Waits until the                                     */
                                                     /* for transfer has been completed.                    */
          timer_wk = com_timer.wait_100ms_scan  ;
          if (timer_wk == 0){                        /* If this remains 1 for 5 seconds,                    */
                                                     /* it can be escaped by an error being generated.      */
               ret = I2C_TDRE_TOUT;                  /* Ended abnormally (timeout)                          */
               goto exit ;
          }

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

     }
```

```
                        /****************************************************************************************************/
                        /*    Sets data                                                                                     */
                        /****************************************************************************************************/
                        IIC2.ICDRT = com_i2c_packet[i] ;
                             /* ##(program note)############################################################################ */
                             /* ## In this sample program example, the received data is directly returned, however,      ## */
                             /* ## if necessary, the transmit data should be appropriately changed for your program.     ## */
                             /* ############################################################################################ */

                   }

                   /*********************************************************************************************************/
                   /*    I 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,             */
                                                                        /* it can be escaped by an error being generated. */
                             ret = I2C_TDRE_TOUT;                       /* Ended abnormally (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,             */
                                                                        /* it can be escaped by an error being generated. */
                             ret = I2C_TEND_TOUT;                       /* Ended abnormally (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 a console.                                                                               */
     /**************************************************************************************************************************/
     if (ret != NORMAL_END){                                           /* Ended abnormally (timeout)                        */
          com_cnsl_msg("  i2c_int_exec err(%02X)\n",ret);
     }

     /**************************************************************************************************************************/
     /*   Opens the IREQ0-3 and SCI rcvint masks                                                                               */
     /**************************************************************************************************************************/
     com_int_ctl(1) ;

}
```

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



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

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

/* ------------------------------------------------------------------------------------------------------------------------------ */
/* 5.3 TimerZ initial settings -------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
     /* ######################################################################################################################### */
     /* ######################################################################################################################### */
     /*                                                                                                                           */
     /*         Sets  TimerZ                                                                                                      */
     /*                                                                                                                           */
     /* ######################################################################################################################### */
     /* ######################################################################################################################### */
     /****************************************************************************************************************************/
     /*   ts 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 GRA Clears the counter                 */
                                                                    /* when a GRA compare match occurs.                       */
                                                                    /* CKEG[1:0] = 00 Counts up at the rising edges           */
                                                                    /* TPSC[2:0] = 011 Counts using internal clock φ/8        */
     TZ0.TIORA.BYTE = 0x00 ;
                                                                    /* IOA[2:0] = 000                                         */
                                                                    /* GRA functions as the output compare register           */
     TZ0.TIER.BYTE  = 0x01 ;
                                                                    /* IMIEA = 1 Enables IMFA                                 */

     TZ0.GRA        = 20000 ;                                       /* 1 Issues 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 Start counting by TCNT_0                  */


/* ---------------------------------------------------------------------------------------------------------------------- */
/* 5.4 TimerZ interrupt processing ------------------------------------------------------------------------------------ */
/* ---------------------------------------------------------------------------------------------------------------------- */
/*************************************************************************************************************************/
/*   1. Module name : h8_TimerZ                                                                                          */
/*   2. Function overview: Interval timer processing every 10 msec                                                       */
/*************************************************************************************************************************/
#pragma interrupt( h8_timerz )
void h8_timerz( void )
{

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

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

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

     /**************************************************************************************************************************/
     /*   Increments the counter                                                                                               */
     /**************************************************************************************************************************/
     com_timer.counter++;
     if( com_timer.counter >= 10 ){
          /*********************************************************************************************************************/
          /*   -1 in units of 100 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.)
- I^2C Bus Usage (published by Phillips)

Revision Record

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