To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

H8/3687

Access to the Serial EEPROM (SPI EEPROM) in Clock Synchronous Mode of the I^{2}C Interface

Introduction

Introduction

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

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

Target Device

H8/3687

Contents

1. Overview

The SPI EEPROM is read from or written to via the H8/3687 I^2C interface in clock synchronous mode.

2. Configuration

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

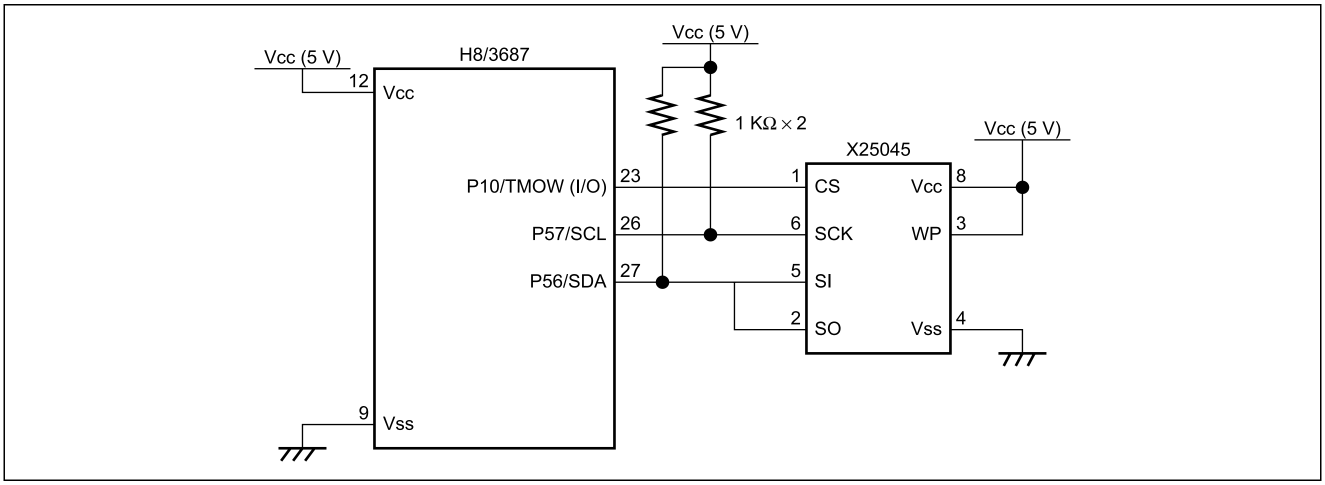


Figure 2.1 Diagram of connections between the H8/3687 and SPI EEPROM.

Note: The above bus connection is possible on the SPI EEPROM X25045 because the SO pin is usually in high impedance (the SO pin is shifted to output mode during read cycles only), however, it is not possible on a device where the SO pin is usually in output mode. If the above connection is employed with such a device, parts will be heated or destroyed. Confirm the target device pin specifications in the data sheet before usage.

Specifications

- H8/3687 operating frequency: 16 MHz
- Table 2.1 shows the SPI EEPROM X25045 pin specifications.
- SPI EEPROM specifications: 4 kbits (512 x 8 bits)

Table 2.1 SPI EEPROM Specifications

Symbol	Description
$\overline{\text{CS}}$	Chip select input
SO	Serial output
SI	Serial input
SCK	Serial clock input
$\overline{\text{WP}}$	Write protect input
Vss	Ground
Vcc	Supply voltage
$\overline{\text{RESET}}$/RESET	Reset output

3. Sample Programs

3.1 Functions

1. Data is written to the SPI EEPROM sequentially (page write).
2. Data is read from the SPI EEPROM sequentially (sequential read).

3.2 Embedding the Sample Programs

1. Sample program 4-A
   Incorporate #define directives.
2. Sample program 4-B
   Incorporate prototype declarations.
3. Sample program 4-C
   Incorporate the source program.
   — Add the initialization.
   — Add SPI EEPROM access 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/jpn/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. The sample program is designed so that timer Z is configured to start every 10 ms with timeout setting of 5 seconds in order to give timing of monitoring the state of the serial communication interface. The timer processing may be modified according to your needs, and of course can be used without modification. When using the timer processing in the sample program without modification, the following changes should be made.
   Sample program 4-D
   - Add Timer Z reset vector.
   - Add com_timer as a common variable.
   - Add Timer Z initial setting processing.
     (Change the GRA setting according to the operating frequency of the microcomputer being used, so that the Timer Z interrupt occurs in 10 ms. For setting values, refer to the H8/3687 Hardware Manual; for the location of the setting to be changed, refer to the program notes in the sample program.)
   - Add Timer Z interrupt processing.
3. The I^2C interface transfer rate ICCR1 (CKS3 to CKS0) should be set according to the target device specifications and the microcomputer operating frequency. Refer to the H8/3687 Hardware Manual for setting values, and to the program notes in the sample program for the location to be changed. In this sample program, the transfer rate is set to 100 kbps.

3.4 Method of use

1. Writing data sequentially to the SPI EEPROM

```
unsigned int com_spi_eeprom_page_write
( unsigned int rom_addr , unsigned int  rom_length, unsigned char *rom_data )
```

Argument	Description
rom_addr	Specifies the ROM address where data is written to.
rom_length	Specifies the write data length. On the device used in this sample program, up to four bytes can be specified.
*rom_data	Specifies the start address of the area where write data is stored.

Return value	Explanation
0	Normal termination
1	Abnormal termination (transfer preparation completion wait timeout)
2	Abnormal termination (transfer completion wait timeout)
3	Abnormal termination (reception completion wait timeout)
4	Abnormal termination (write wait timeout)

Example of use:

```
int ret;
unsigned char *rom_data ;
unsigned int  rom_length , rom_addr ;
ret = com_spi_eeprom_page_write (rom_addr , rom_length , *rom_data)
```

2. Reading data sequentially from the SPI EEPROM.

```
unsigned int com_spi_eeprom_seq_read
( unsigned int rom_addr , unsigned int  rom_length, unsigned char *rom_data )
```

Argument	Explanation
rom_addr	Specifies the ROM address where data is read from.
rom_length	Specifies the read data length.
*rom_data	Specifies the start address of the area where read data is stored.

Return value	Explanation
0	Normal termination
1	Abnormal termination (transfer preparation completion wait timeout)
2	Abnormal termination (transfer completion wait timeout)
3	Abnormal termination (reception completion wait timeout)

Example of use:

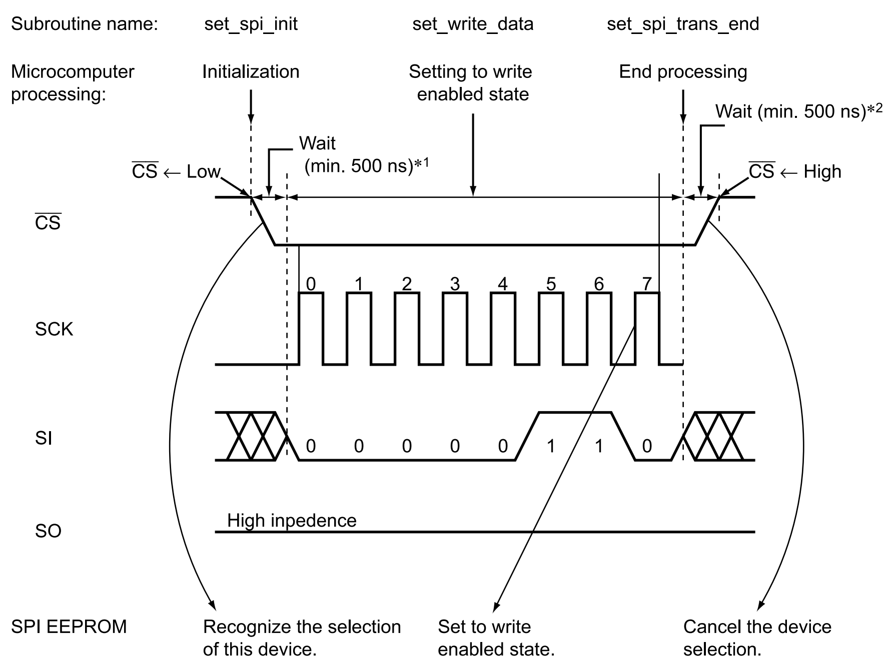
```
int ret;
unsigned char *rom_data;
unsigned int  rom_length , rom_addr;
ret = com_spi_eeprom_seq_read (rom_addr , rom_length , *rom_data)
```

3.5 Description of operation

The operation is as described below. The following figure depicts the operation of the H8 microcomputer and the EEPROM with respect to the states of the interface signals.

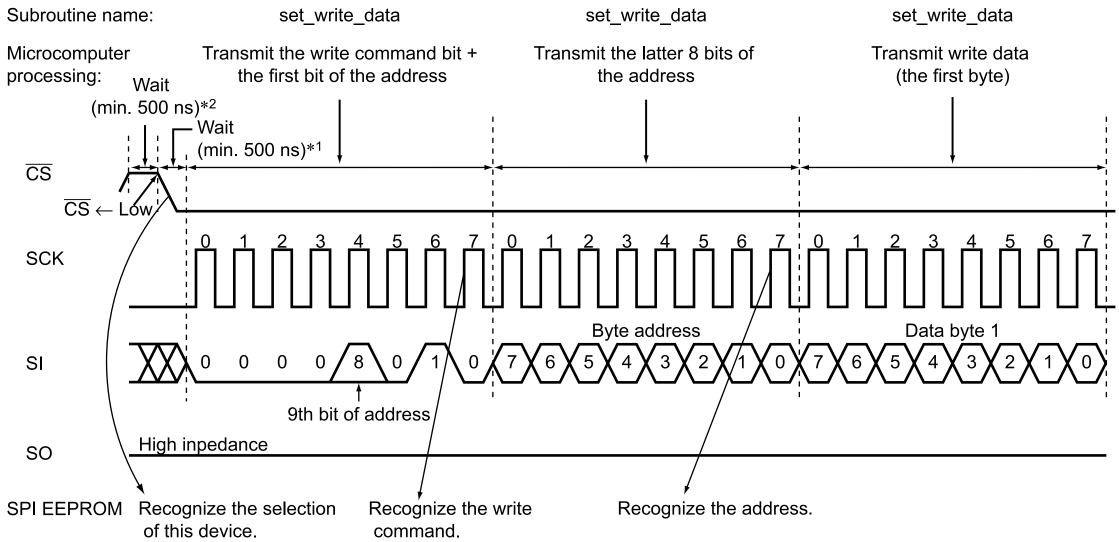
1. Writing data to the SPI EEPROM sequentially (page write)
   In this example of operation, writing of four bytes of data is shown.

   Step (a): Cancel the SPI EEPROM write disabled state.



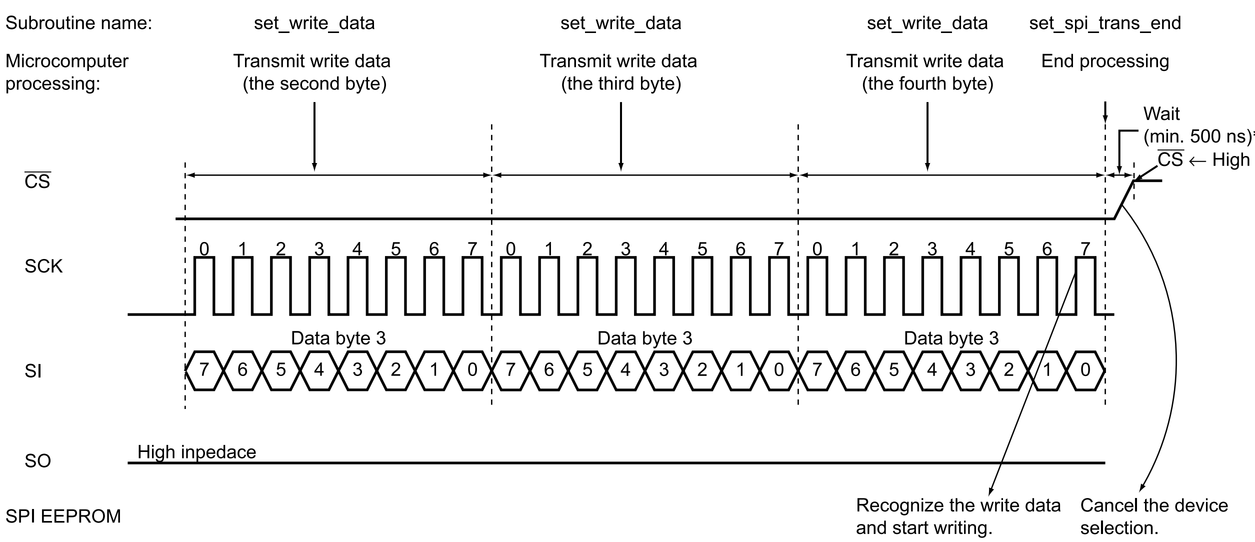
Notes
1. A wait is inserted here for the purpose of allowing at least a 500 ns interval from assertion of $\overline{CS}$ to starting the data transfer. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.
2. A wait is inserted here for the purpose of allowing at least a 500 ns interval from the end of data transfer to negation of $\overline{CS}$. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

Step (b): Write data.



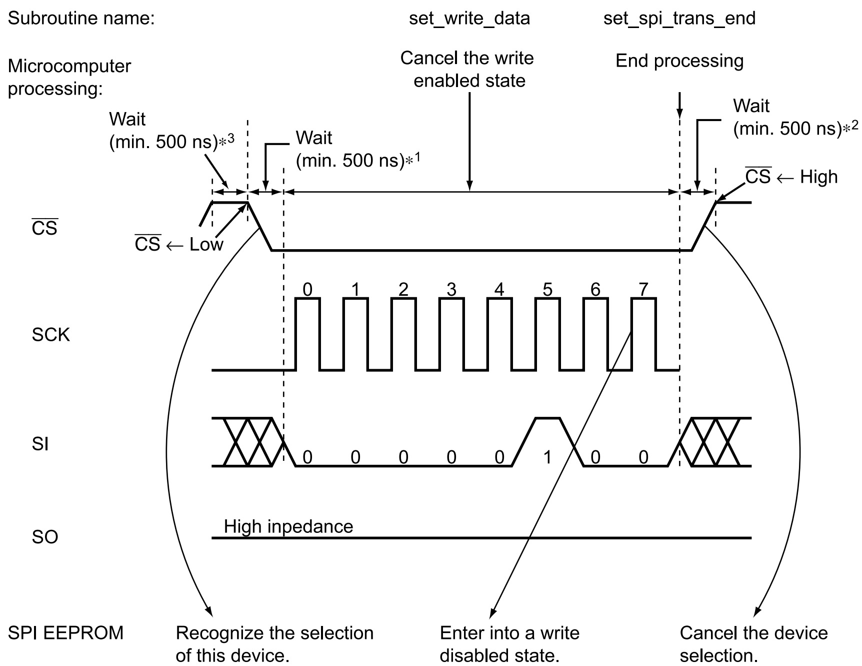
Notes: 1. A wait is inserted here for the purpose of allowing at least a 500 ns interval from assertion of CS to starting the data transfer. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

2. A wait is inserted here for the purpose of allowing at least a 500 ns interval after negation of CS until it is asserted again. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.



Note: * A wait is inserted here for the purpose of allowing at least a 500 ns interval from the end of data transfer until negation of CS. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

Step (c): Return to the SPI EEPROM write disabled state.

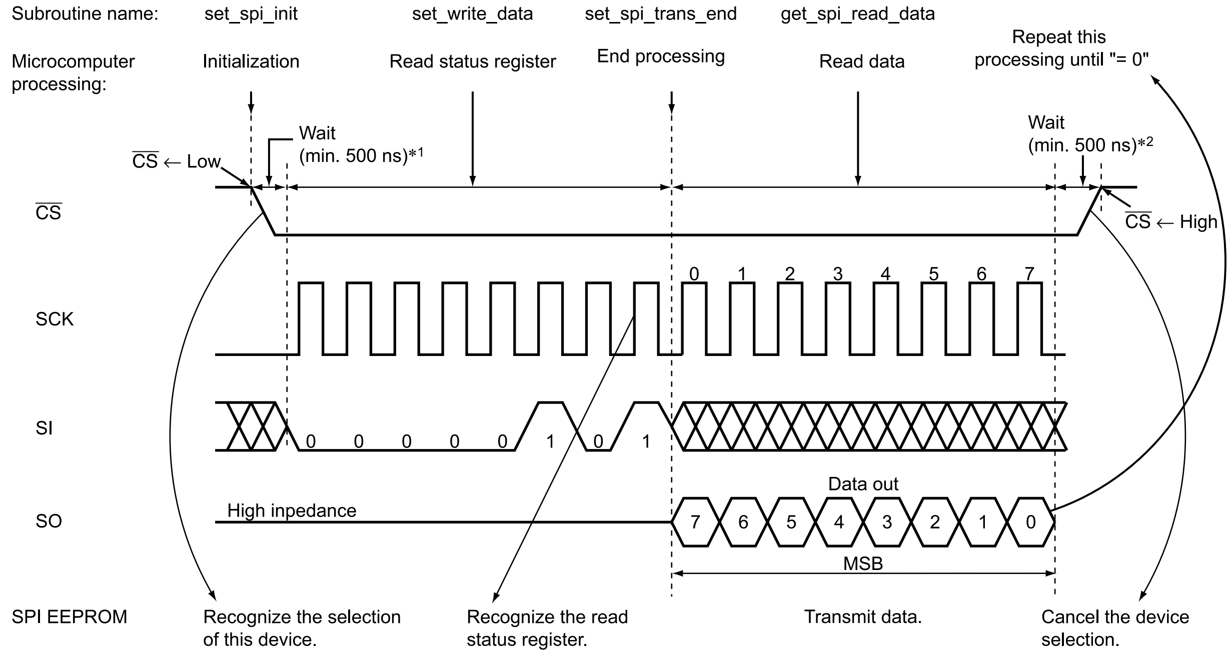


Notes: 1. A wait is inserted here for the purpose of allowing at least a 500 ns interval from assertion of CS to starting the data transfer. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

2. A wait is inserted here for the purpose of allowing at least a 500 ns interval from the end of data transfer to negation of CS. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

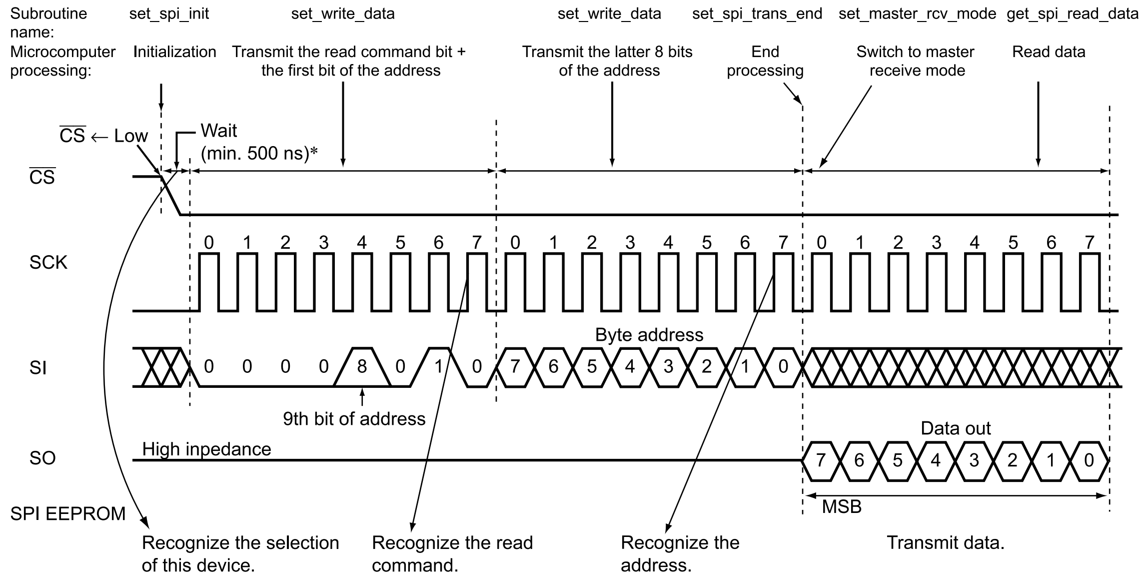
3. A wait is inserted here for the purpose of allowing at least a 500 ns interval after negation of CS until it is asserted again. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting

Step (d): Check for the end of write processing.

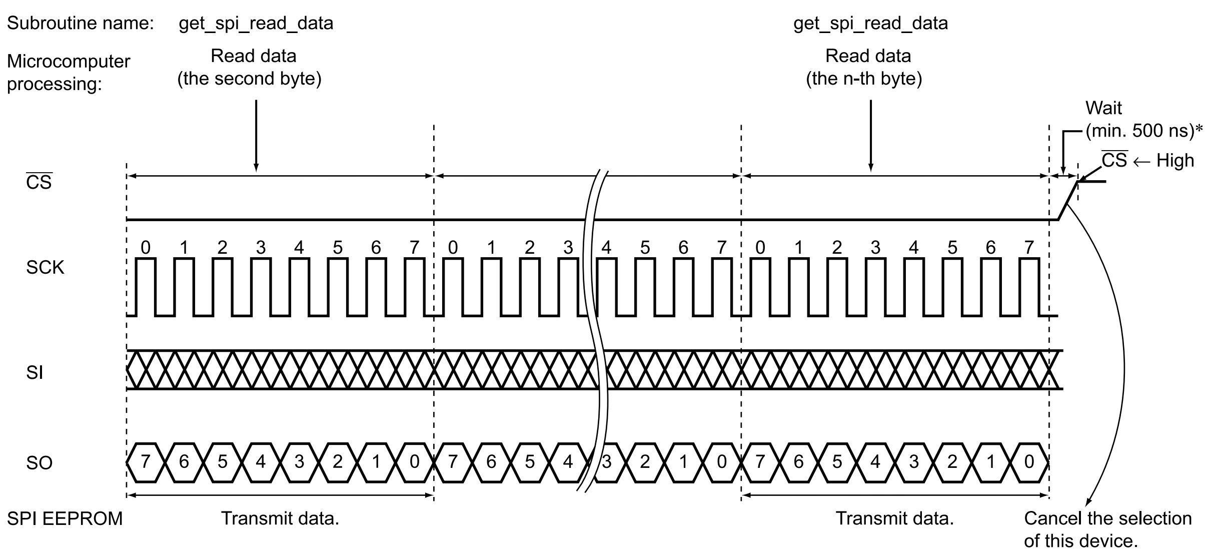


Notes: 1. A wait is inserted here for the purpose of allowing at least a at least 500 ns interval from assertion of $\overline{CS}$ to starting the data transfer. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.
2. A wait is inserted here for the purpose of allowing at least a 500 ns interval from the end of data transfer to negation of $\overline{CS}$. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

2. Reading from the SPI EEPROM sequentially (sequential read).



Note: * A wait is inserted here for the purpose of allowing at least a 500 ns interval from assertion of $\overline{CS}$ to starting the data transfer. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.



Note: * A wait is inserted here for the purpose of allowing at least a 500 ns interval from the end of data transfer to negation of $\overline{CS}$. Since the waiting time is different on each device, refer to the data sheet to set the appropriate waiting time.

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

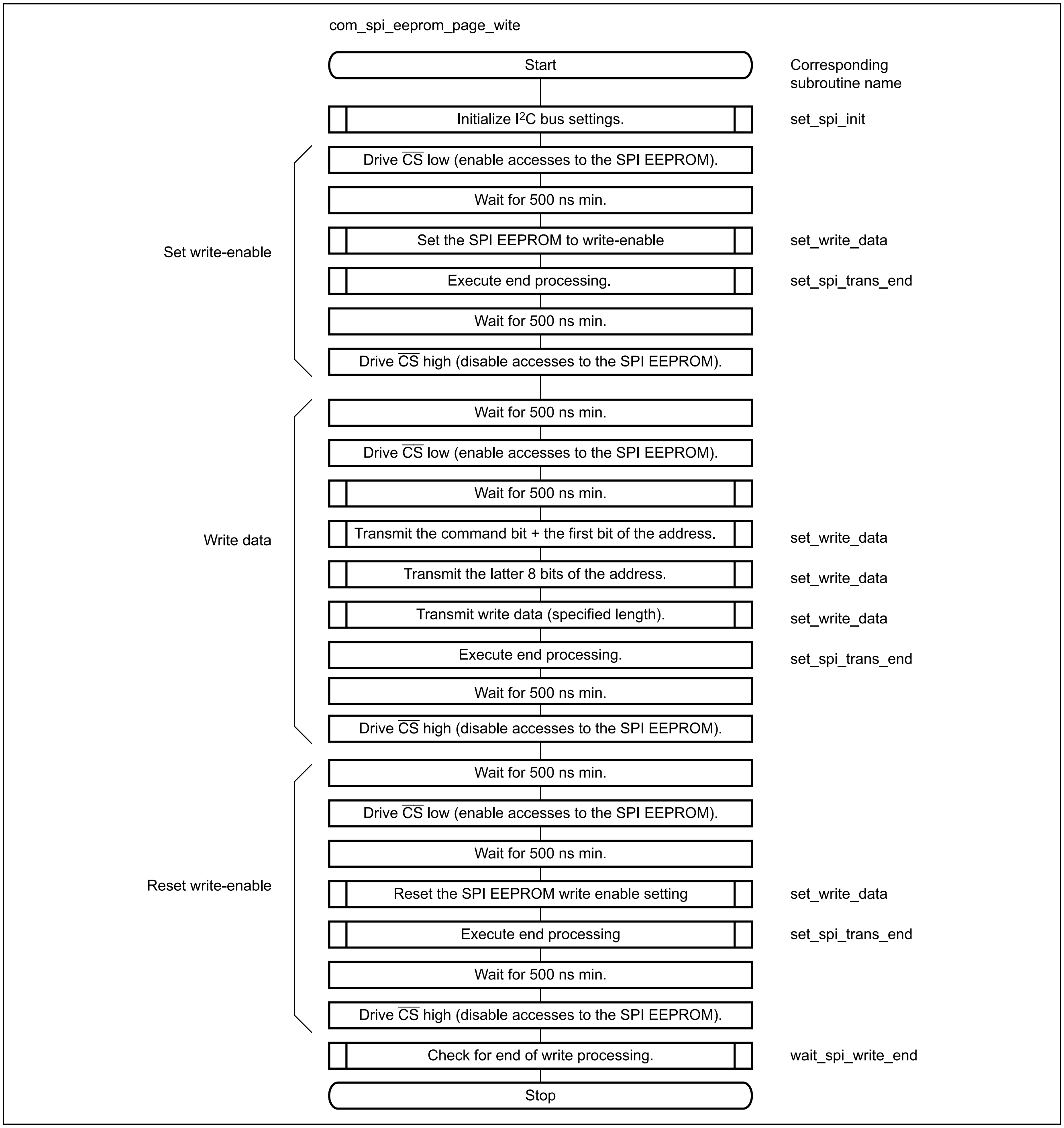
Name	Summary
I^2C bus control register 1 (ICCR1)	Starts or stops operation of the I^2C bus interface 2, controls transmission/reception, and selects master/slave mode, transmission/reception, and master mode transfer clock frequency.
I^2C bus control register 2 (ICCR2)	Issues start/stop conditions, operates 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 individual interrupts, validates/invalidates acknowledge, sets transmit acknowledge, and checks receive acknowledge.
I^2C bus status register (ICSR)	Used for checking of interrupt request flags and the statuses.
Slave address register (SAR)	Sets the slave address and transfer format.
I^2C bus transmit data register (ICDRT)	8-bit readable/writable register which stores data for transmission.
I^2C bus receive data register (ICDRR)	8-bit register which stores received data.

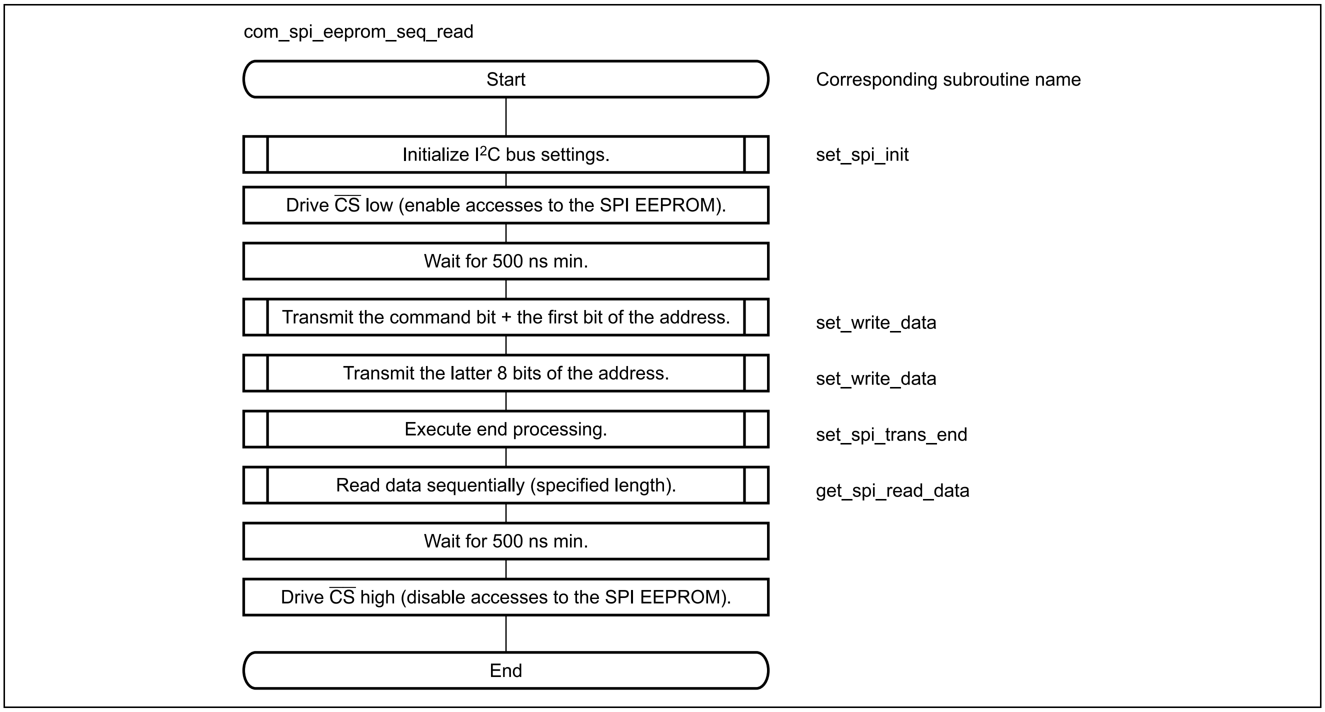
2. Timer Z-related registers
   Timer Z has various functions, but in the sample program it uses the compare-match function with the GRA register to generate an interrupt every 10 ms.

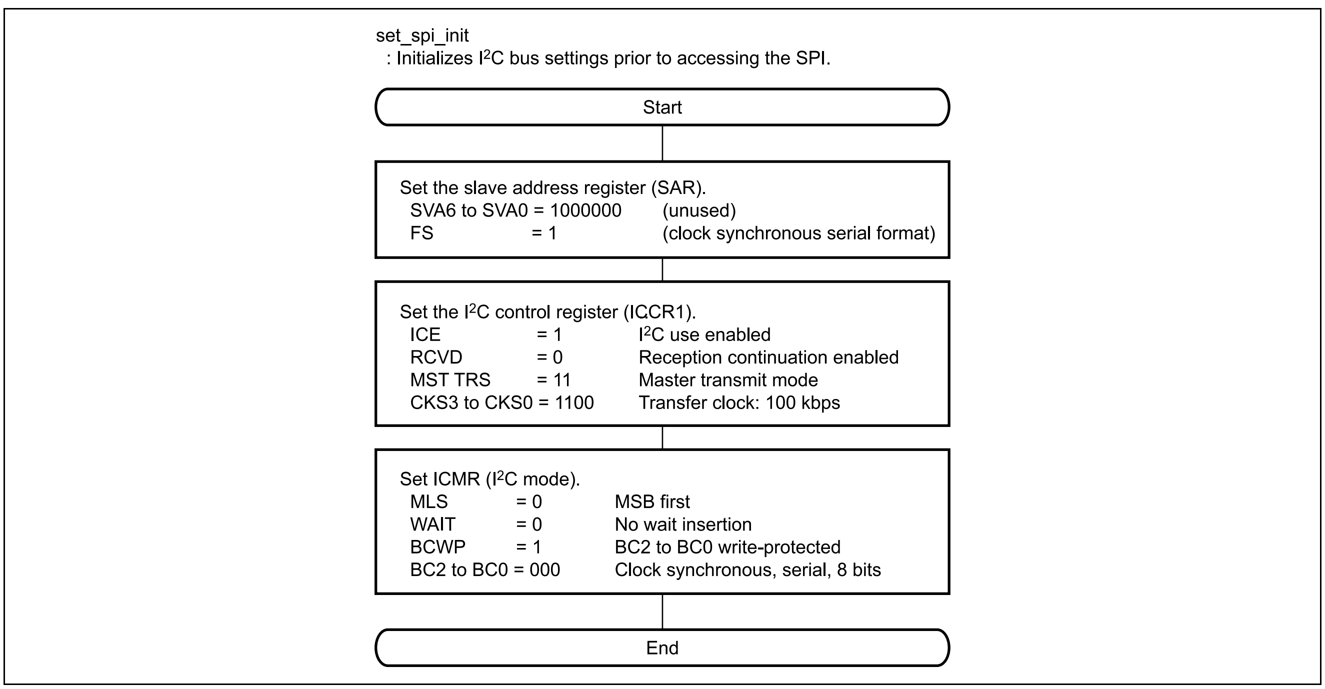
Name	Summary
Timer start register (TSTR)	Starts or stops TCNT operation.
Timer mode register W (TMDR)	Sets buffer operation and selects synchronous operation.
Timer PWM mode register (TPMR)	Sets pins for PWM mode. Not used in this sample program.
Timer function control register (TFCR)	Selects operation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

3.7 Flowcharts

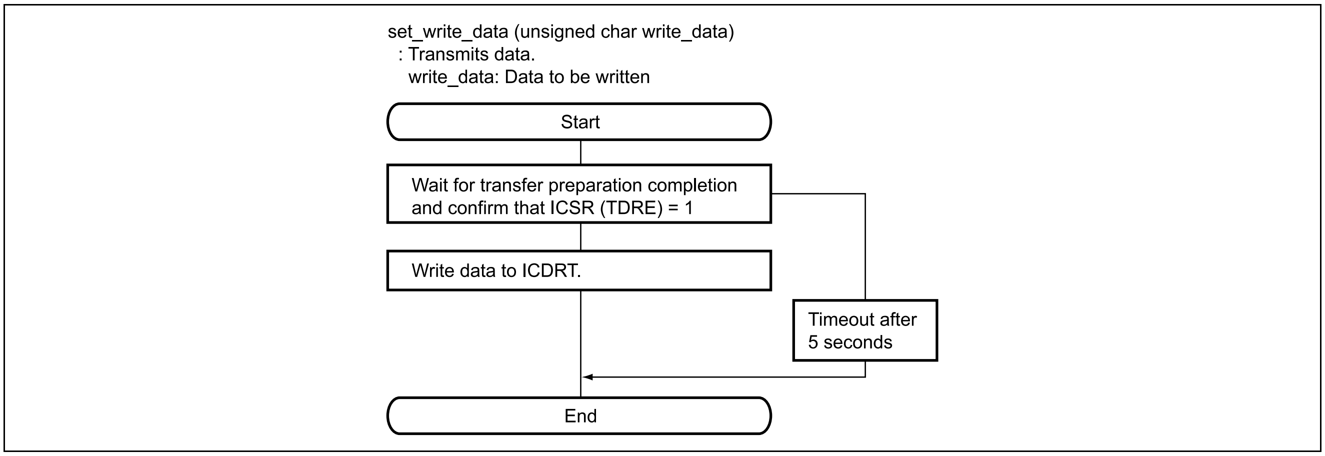
The program processing flow is shown below.

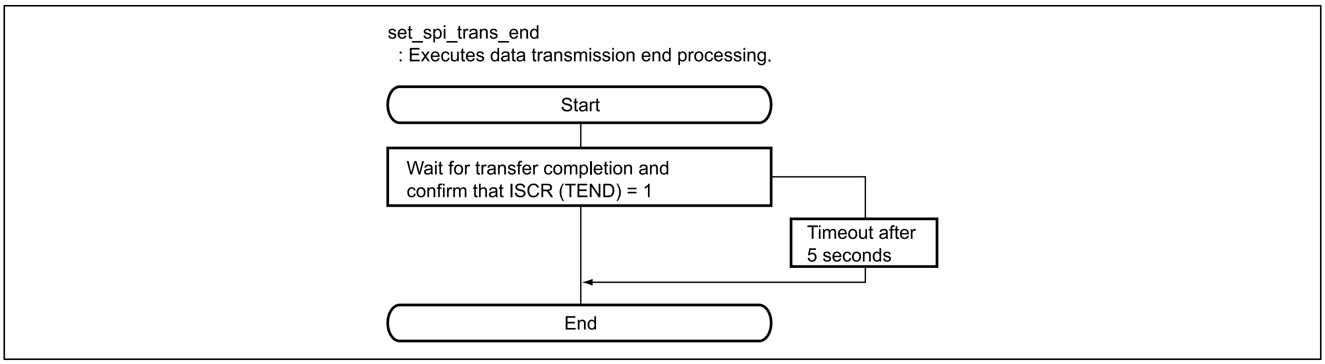


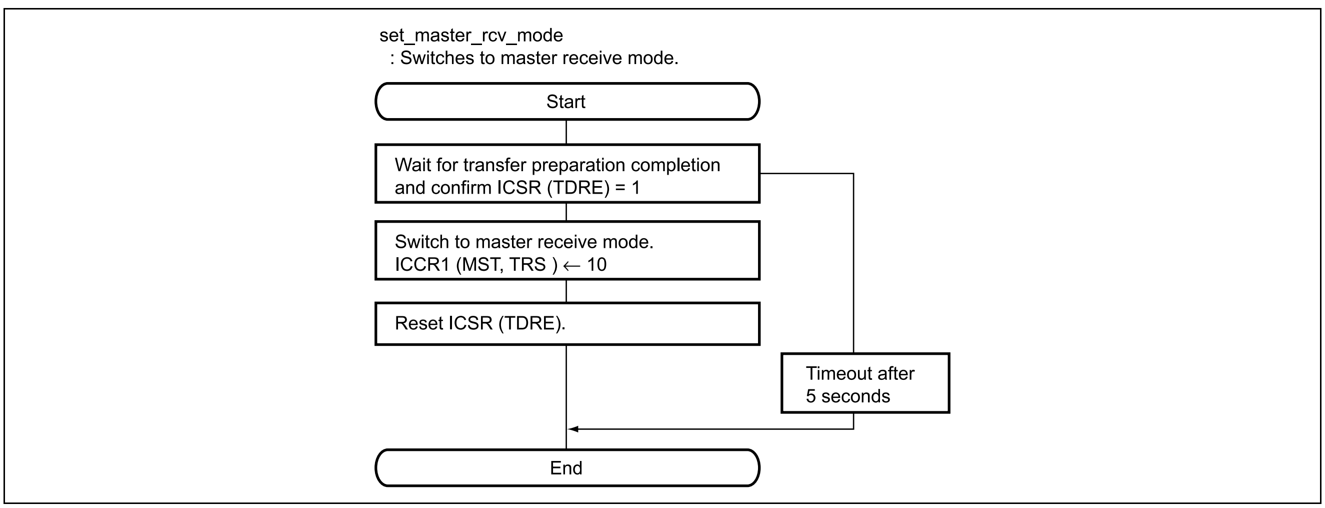




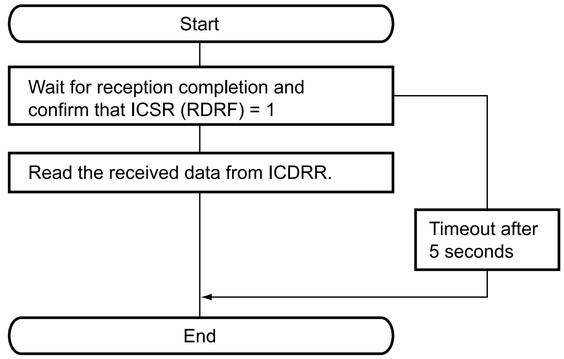
com_delay (int delaytime)
: Produces a delay of desired length.
delaytime: 1 for approx. 0.5 µs


set_write_data (unsigned char write_data)
: Transmits data.
write_data: Data to be written
Start
Wait for transfer preparation completion and confirm that ICSR (TDRE) = 1
Write data to ICDRT.
Timeout after 5 seconds
End

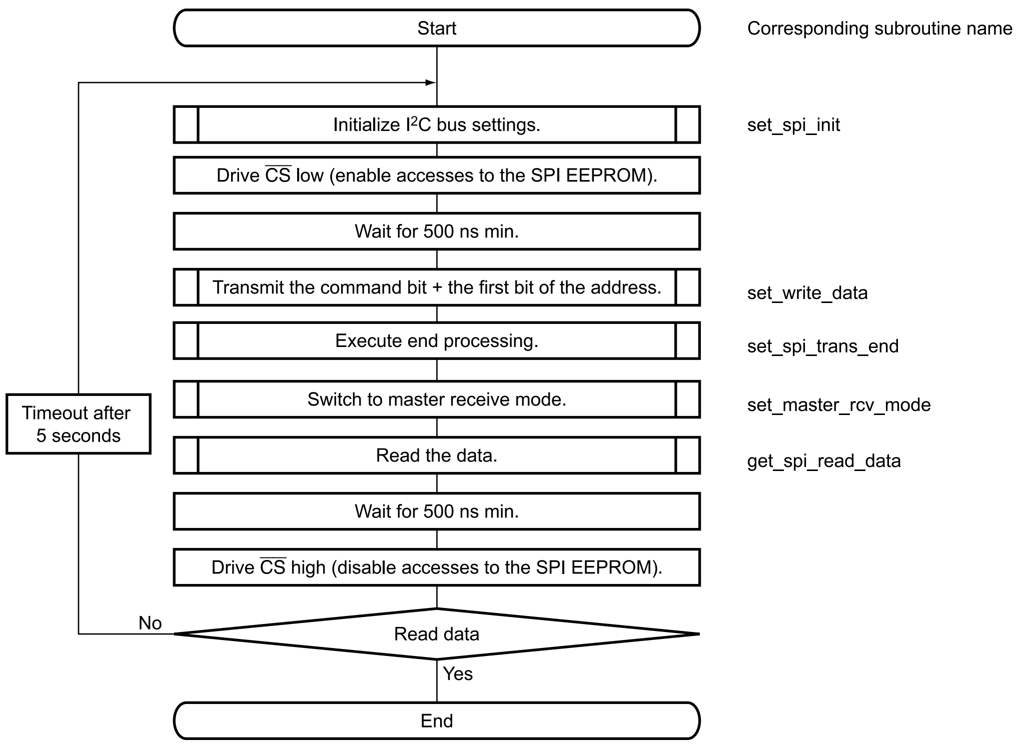

set_spi_trans_end
: Executes data transmission end processing.
Start
Wait for transfer completion and confirm that ISCR (TEND) = 1
Timeout after 5 seconds
End


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

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



wait_spi_write_end
: Checks for the end of SPI write processing.



3.8 Program Listing

```
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* 1. Sample Program 4-A  #define directives --------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/********************************************************************************************************************************/
/*   For SPIEEPROM access                                                                                                        */
/********************************************************************************************************************************/
#define   SET_WRITE_MODE      0x02
#define   SET_READ_MODE       0x03
#define   RESET_WRITE_ENABLE  0x04
#define   READ_STATUS         0x05
#define   SET_WRITE_ENABLE    0x06

/********************************************************************************************************************************/
/*   SPIEEPROM access error code (codes other than 0)                                                                            */
/********************************************************************************************************************************/
#define   I2C_TDRE_TOUT       1
#define   I2C_TEND_TOUT       2
#define   I2C_RDRF_TOUT       3
#define   WRITE_TOUT          4

/* ---------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* 2. Sample program 4-B  Prototype declarations ----------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */

/********************************************************************************************************************************/
/********************************************************************************************************************************/
/*   SPI BUS access processing                                                                                                   */
/********************************************************************************************************************************/
/********************************************************************************************************************************/
void com_delay( int delaytime ) ;
void set_spi_init ( ) ;
unsigned int set_master_rcv_mode () ;
unsigned int set_write_data (unsigned char write_data);
unsigned int set_spi_trans_end ();
unsigned int wait_spi_write_end () ;
unsigned int get_spi_read_data (unsigned char *read_data);
unsigned int com_spi_eeprom_seq_read
               ( unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data ) ;
unsigned int com_spi_eeprom_page_write
               ( unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data ) ;
```

```
/* ------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/* 3. Sample program 4-C  Source code---------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------------------------------------------ */

/* ------------------------------------------------------------------------------------------------------------------------------ */
/*   3.1 Initialization ----------------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------------ */
/*   Add the followings to the initial settings in the H8 start-up processing                                                    */

     /*****************************************************************************************************************************/
     /*   PCR1      Defines input/output for the IO port 1.                                                                       */
     /*        PCR17     = 1  Not used (defined as an output pin)                                                                 */
     /*        PCR16     = 1  Not used (defined as an output pin)                                                                 */
     /*        PCR15     = 1  Not used (defined as an output pin)                                                                 */
     /*        PCR14     = 1  Not used (defined as an output pin)                                                                 */
     /*        PCR12     = 1  Not used (defined as an output pin)                                                                 */
     /*        PCR11     = 1  Not used (defined as an output pin)                                                                 */
     /*        PCR10     = 1  Used as the CS pin of SPIEEPROM                                                                     */
     /*****************************************************************************************************************************/
     IO.PCR1             = 0xFF ;

     /*****************************************************************************************************************************/
     /*   PDR1      Specifies the output data of the IO port 1.                                                                   */
     /*        PDR17     = 0  Not used (defined as an output pin)                                                                 */
     /*        PDR16     = 0  Not used (defined as an output pin)                                                                 */
     /*        PDR15     = 0  Not used (defined as an output pin)                                                                 */
     /*        PDR14     = 0  Not used (defined as an output pin)                                                                 */
     /*        PDR12     = 0  Not used (defined as an output pin)                                                                 */
     /*        PDR11     = 0  Not used (defined as an output pin)                                                                 */
     /*        PDR10     = 1  Sets CS# = high (SPIEEPROM not active).                                                             */
     /*****************************************************************************************************************************/
     IO.PDR1.BYTE             = 0x01 ;
          /* ##(program note)################################################################################################# */
          /* ## Since the signal on the CS pin of SPIEEPROMk is active-low, the CS pin should be initialized to high        ## */
          /* ## (not active).                                                                                               ## */
          /* ## Though the CS level is not guaranteed before this setting, SPIEEPROM cannot be written illegally            ## */
          /* ## because SPIEEPROM is write protected.                                                                       ## */
          /* ################################################################################################################## */
```

```
/* ---------------------------------------------------------------------------------------------------------------------------- */
/*   3.2 SPIEEPROM access processing --------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------------------------------------- */

/******************************************************************************************************************************/
/******************************************************************************************************************************/
/******************************************************************************************************************************/
/*                                                                                                                            */
/*                       SPI EEPROM control with I2C clock synchronization                                                    */
/*                                                                                                                            */
/******************************************************************************************************************************/
/******************************************************************************************************************************/
/******************************************************************************************************************************/

/******************************************************************************************************************************/
/*   1. Module name: com_delay                                                                                                */
/*   2. Function overview: Produces a delay of any desired length.                                                            */
/******************************************************************************************************************************/
void com_delay( int delaytime )
{
     register int i,a;

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

/******************************************************************************************************************************/
/*   1. Module name: set_spi_init                                                                                             */
/*   2. Function overview: Initializes settings prior to SPI access.                                                          */
/******************************************************************************************************************************/
void set_spi_init( )
{
     /*************************************************************************************************************************/
     /*   SAR            Sets the slave address register                                                                      */
     /*        SVA6:0    = 1000000 (unique value)                                                                             */
     /*        FS        = 1 Clock synchronous serial format                                                                  */
     /*************************************************************************************************************************/
          /* ##(program note)################################################################################################ */
          /* ## SVA6:0 are used in the slave mode. They should be set to a unique address that is different              ## */
          /* ## from the addresses used for other slave devices connected to the I2C bus                                 ## */
          /* ################################################################################################################ */

     IIC2.SAR.BYTE= 0x81 ;

     /*************************************************************************************************************************/
     /*   ICCR1          Sets the I2C control register.                                                                       */
     /*        ICE       = 1  I2C use enabled                                                                                 */
     /*        RCVD      = 0  Reception disabled.                                                                             */
     /*        MST,TRS   = 11      Master transmit mode                                                                       */
     /*        CKS3:0    = 1100  Transfer clock frequency (φ/160, transfer rate: 100kbps)                                     */
     /*************************************************************************************************************************/

     IIC2.ICCR1.BYTE     = 0xBC ;
          /* ##(program note)################################################################################################ */
          /* ## Setting of CKS3:0 should be changed according to the required transfer rate.                             ## */
          /* ## For details, please refer to the H8/3687 Hardware Manual.                                                ## */
          /* ################################################################################################################ */
```

```
    /****************************************************************************************************************/
    /*   ICMR      Sets I2C mode                                                                                   */
    /*        MLS       = 0  MSB first                                                                             */
    /*        WAIT      = 0  No wait inserted                                                                      */
    /*        BCWP      = 1  BC2:0 write protect                                                                   */
    /*        BC[2:0]   = 000 Clock synchronous, serial, 8 bits                                                    */
    /****************************************************************************************************************/
    IIC2.ICMR.BYTE= 0x08 ;

}

/*********************************************************************************************************************/
/*   1. Module name: set_write_data                                                                                 */
/*   2. Function overview: Transmits data.                                                                          */
/*********************************************************************************************************************/
unsigned int set_write_data (unsigned char write_data)
{
    int ret , Timer_wk , i ;
    unsigned char buf ;

    ret = NORMAL_END ;

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

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

    }

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

exit :
    return (ret);

}
```

```
/******************************************************************************************************************************/
/*   1. Module name: set_spi_trans_end                                                                                        */
/*   2. Function overview: Executes data transmit end processing.                                                             */
/******************************************************************************************************************************/
unsigned int set_spi_trans_end ()
{
     int ret , Timer_wk;

     ret = NORMAL_END ;
     /*************************************************************************************************************************/
     /*   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){                                                      /* Timeout after 5 seconds                       */
               ret = I2C_TEND_TOUT;                                                /* Abnormal end (timeout).                       */
               goto exit ;
          }

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

     }

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 preparation                       */
                                                                                   /* for transfer has been completed.              */
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                                      /* Timeout after 5 seconds                       */
               ret = I2C_TDRE_TOUT;                                                /* Abnormal end (timeout).                       */
               goto exit ;
          }

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

     }
```

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


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

exit :
     return (ret);
}

/******************************************************************************************************************/
/*   1. Module name: get_spi_read_data                                                                              */
/*   2. Function overview: Receives data.                                                                           */
/******************************************************************************************************************/
unsigned int get_spi_read_data (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 reception has been completed.*/
          Timer_wk = com_timer.wait_100ms_scan ;
          if (Timer_wk == 0){                                                   /* Timeout after 5 seconds                   */
               ret = I2C_RDRF_TOUT;                                             /* Abnormal end (timeout).                   */
               goto exit ;
          }

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

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


exit :
     return (ret);
}
```

```
/*************************************************************************************************************************/
/*   1. Module name: wait_spi_write_end                                                                                  */
/*   2. Function overview: Checks for the end of SPI write.                                                              */
/*************************************************************************************************************************/
unsigned int wait_spi_write_end ()
{

     int ret , i;
      unsigned char      status;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;
     unsigned char dummy_data ;

     ret = NORMAL_END ;

     com_timer.wait_100ms = 50 ;

     do{
          /******************************************************************************************************************/
          /*   Initializes the I2C                                                                                          */
          /******************************************************************************************************************/
          set_spi_init( ) ;

          /******************************************************************************************************************/
          /*   Drives CS# low (enables SPIEEPROM access).                                                                   */
          /******************************************************************************************************************/
          IO.PDR1.BYTE = 0x00 ;
          com_delay(10) ;
               /* ##(program note)############################################################################################ */
               /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from assertion of CS        ## */
               /* ## to starting the data transfer.                                                                         ## */
               /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet       ## */
               /* ## to set the appropriate waiting time.                                                                   ## */
               /* ############################################################################################################ */

          /******************************************************************************************************************/
          /******************************************************************************************************************/
          /*   Reads data.                                                                                                  */
          /******************************************************************************************************************/
          /******************************************************************************************************************/
          /******************************************************************************************************************/
          /*   Transmits command data (Read Status).                                                                        */
          /******************************************************************************************************************/
          ret = set_write_data (READ_STATUS) ;
          if (ret !=0) { goto exit ;}

          /******************************************************************************************************************/
          /*   Executes the end processing.                                                                                 */
          /******************************************************************************************************************/
          ret = set_spi_trans_end () ;
          if (ret !=0) { goto exit ;}
```

```
        /**************************************************************************************************************/
        /*   Switches to the master receive mode.                                                                      */
        /**************************************************************************************************************/
        ret = set_master_rcv_mode () ;
             if (ret !=0) { goto exit ;}
        /**************************************************************************************************************/
        /*   Disables the next receive operation following the data reception.                                        */
        /**************************************************************************************************************/
        com_delay(10) ;
             /* ##(program note)############################################################################################ */
             /* ## In the case of 1 byte read, the receive clock will not be output if ICCR1 (RCVD) = 1 immediately       ## */
             /* ## after switching to the master                                                                          ## */
             /* ## Wait for a few µs.                                                                                     ## */
             /* ############################################################################################################ */
        IIC2.ICCR1.BIT.RCVD = 1  ;

        /**************************************************************************************************************/
        /*   Reads data.                                                                                               */
        /**************************************************************************************************************/
        ret = get_spi_read_data (&status) ;
        if (ret !=0) { goto exit ;}

        /**************************************************************************************************************/
        /*   Drives CS# high (disables SPIEEPROM access)                                                               */
        /**************************************************************************************************************/
        com_delay(10) ;
             /* ##(program note)############################################################################################ */
             /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval                             ## */
             /* ## from the end of data transfer to negation of CS#.                                                      ## */
             /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet       ## */
             /* ## to set the appropriate waiting time.                                                                   ## */
             /* ############################################################################################################ */
        IO.PDR1.BYTE = 0x01 ;

        #ifdef UT
             status = 0x01 ;
        #endif
        if (com_timer.wait_100ms == 0){                                    /* Timeout after 5 seconds               */
             ret = WRITE_TOUT;                                             /* Abnormal end (timeout)                */
             goto exit ;
        }

    } while ((status & 0x01) == 1) ;                                       /* While write is being performed        */

exit :                                                                     /* Error processing                      */
    /**************************************************************************************************************/
    /*   Drives CS# high (disables SPIEEPROM access).                                                              */
    /**************************************************************************************************************/
    com_delay(10) ;
        /* ##(program note)############################################################################################ */
        /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from the end of data transfer ## */
        /* ## to negation of CS#.                                                                                      ## */
        /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet         ## */
        /* ## to set the appropriate waiting time.                                                                     ## */
        /* ############################################################################################################ */
    IO.PDR1.BYTE = 0x01 ;

    return (ret) ;

}
```

```
/*******************************************************************************************************************/
/*   1. Module name: com_spi_eeprom_seq_read                                                                       */
/*   2. Function overview: Reads the specified length of data from SPIEEPROM.                                      */
/*******************************************************************************************************************/
unsigned int com_spi_eeprom_seq_read ( unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data )
{
     int ret , i;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;


     ret = NORMAL_END ;

     /**************************************************************************************************************/
     /*   Initializes the I2C.                                                                                     */
     /**************************************************************************************************************/
     set_spi_init( ) ;

     /**************************************************************************************************************/
     /*   Drives CS# low (enables SPIEEPROM access).                                                               */
     /**************************************************************************************************************/
     IO.PDR1.BYTE = 0x00 ;
     com_delay(10) ;
          /* ##(program note)################################################################################### */
          /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from assertion of CS# to starting ## */
          /* ## the data transfer.                                                                              ## */
          /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet ## */
          /* ## to set the appropriate waiting time.                                                            ## */
          /* ################################################################################################### */

     /**************************************************************************************************************/
     /**************************************************************************************************************/
     /*   Reads data.                                                                                              */
     /**************************************************************************************************************/
     /**************************************************************************************************************/

     /**************************************************************************************************************/
     /*   Transmits the command data and address start bit.                                                        */
     /**************************************************************************************************************/
     buf.d_int = rom_addr ;
     buf.d_byte[0] = (buf.d_byte[0] && 0x01) << 3 ;
     buf.d_byte[0] |= SET_READ_MODE ;

     ret = set_write_data (buf.d_byte[0]) ;
     if (ret !=0) { goto exit ;}

     /**************************************************************************************************************/
     /*   Transmits the latter 8 bits of the address.                                                              */
     /**************************************************************************************************************/
     ret = set_write_data (buf.d_byte[1]) ;
     if (ret !=0) { goto exit ;}

     /**************************************************************************************************************/
     /*   Executes the end processing.                                                                             */
     /**************************************************************************************************************/
     ret = set_spi_trans_end () ;
     if (ret !=0) { goto exit ;}
```

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

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

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

    /**************************************************************************************************************************/
    /*   Reads the last byte.                                                                                                 */
    /**************************************************************************************************************************/
    /**************************************************************************************************************************/
    /*   Disables the next receive operation following the data reception.                                                   */
    /**************************************************************************************************************************/
    com_delay(10) ;
         /* ##(program note)################################################################################################ */
         /* ## In the case of 1 byte read, the receive clock will not be output if ICCR1 (RCVD) = 1 immediately            ## */
         /* ## after switching to the master receive mode.                                                                  ## */
         /* ## Wait for several µs.                                                                                         ## */
         /* ################################################################################################################ */
    IIC2.ICCR1.BIT.RCVD = 1  ;

    /**************************************************************************************************************************/
    /*   Reads data.                                                                                                          */
    /**************************************************************************************************************************/
    ret = get_spi_read_data (&buf.d_byte[0]) ;
    if (ret !=0) { goto exit ;}

    *rom_data = buf.d_byte[0] ;

exit :
    /**************************************************************************************************************************/
    /*   Drives CS# high (disables SPIEEPROM access).                                                                         */
    /**************************************************************************************************************************/
    com_delay(10) ;
         /* ##(program note)################################################################################################ */
         /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from the end of data transfer     ## */
         /* ## to negation of CS#.                                                                                          ## */
         /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet            ## */
         /* ## to set the appropriate waiting time.                                                                         ## */
         /* ################################################################################################################ */

    IO.PDR1.BYTE = 0x01 ;

    /**************************************************************************************************************************/
    /*   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.                       */
    return (ret);

}
```

```
/*******************************************************************************************************************************/
/*   1. Module name: com_spi_eeprom_page_write                                                                                 */
/*   2. Function overview: Writes the specified length of data to SPIEEPROM.                                                   */
/*******************************************************************************************************************************/
unsigned int com_spi_eeprom_page_write ( unsigned int rom_addr , unsigned int rom_length , unsigned char *rom_data )
{
     int ret , i ;
     union {
           unsigned int       d_int ;
           unsigned char      d_byte[2];
     } buf;


     ret = NORMAL_END ;

     /**************************************************************************************************************************/
     /*   Initializes the I2C.                                                                                                 */
     /**************************************************************************************************************************/
     set_spi_init( ) ;

     /**************************************************************************************************************************/
     /**************************************************************************************************************************/
     /*   Cancels the SPI EEPROM write enable.                                                                                 */
     /**************************************************************************************************************************/
     /**************************************************************************************************************************/
     /**************************************************************************************************************************/
     /*   Drives CS# low (enables SPIEEPROM access).                                                                           */
     /**************************************************************************************************************************/
     IO.PDR1.BYTE = 0x00 ;
     com_delay(10) ;
          /* ##(program note)################################################################################################### */
          /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from assertion of CS#            ## */
          /* ## to starting the data transfer.                                                                              ## */
          /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet            ## */
          /* ## to set the appropriate waiting time.                                                                        ## */
          /* ################################################################################################################### */

     /**************************************************************************************************************************/
     /*   Cancels the SPI EEPROM write enable.                                                                                 */
     /**************************************************************************************************************************/
     ret = set_write_data (SET_WRITE_ENABLE) ;
     if (ret !=0) { goto exit ;}

     /**************************************************************************************************************************/
     /*   Executes the end processing.                                                                                         */
     /**************************************************************************************************************************/
     ret = set_spi_trans_end () ;
     if (ret !=0) { goto exit ;}

     /**************************************************************************************************************************/
     /*   Drives CS# high (disables SPIEEPROM access).                                                                         */
     /**************************************************************************************************************************/
     com_delay(10) ;
          /* ##(program note)################################################################################################### */
          /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from the end of data transfer    ## */
          /* ## to negation of CS#.                                                                                         ## */
          /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet            ## */
          /* ## to set the appropriate waiting time.                                                                        ## */
          /* ################################################################################################################### */

     IO.PDR1.BYTE = 0x01 ;
```

```
/*********************************************************************************************************************************/
/*********************************************************************************************************************************/
/*   Writes data.                                                                                                                */
/*********************************************************************************************************************************/
/*********************************************************************************************************************************/
/*********************************************************************************************************************************/
/*   Drives CS# low (enables SPIEEPROM access).                                                                                  */
/*********************************************************************************************************************************/
com_delay(10) ;
     /* ##(program note)################################################################################################## */
     /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval after negation of CS# until       ## */
     /* ## it is asserted again.                                                                                        ## */
     /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
     /* ## to set the appropriate waiting time.                                                                          ## */
     /* ################################################################################################################### */
IO.PDR1.BYTE = 0x00 ;
com_delay(10) ;
     /* ##(program note)################################################################################################## */
     /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from assertion of CS#             ## */
     /* ## to starting the data transfer.                                                                               ## */
     /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
     /* ## to set the appropriate waiting time.                                                                          ## */
     /* ################################################################################################################### */


/*********************************************************************************************************************************/
/*   Transmits the command data and the first bit of the address.                                                                */
/*********************************************************************************************************************************/
buf.d_int = rom_addr ;
buf.d_byte[0] = (buf.d_byte[0] && 0x01) << 3 ;
buf.d_byte[0] |= SET_WRITE_MODE ;

ret = set_write_data (buf.d_byte[0]) ;
if (ret !=0) { goto exit ;}

/*********************************************************************************************************************************/
/*   Transmits the latter 8 bits of the address.                                                                                 */
/*********************************************************************************************************************************/
ret = set_write_data (buf.d_byte[1]) ;
if (ret !=0) { goto exit ;}

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

```
/*******************************************************************************************************************/
/*   Executes the end processing.                                                                                  */
/*******************************************************************************************************************/
ret = set_spi_trans_end () ;
if (ret !=0) { goto exit ;}

/*******************************************************************************************************************/
/*   Drives CS# high (disables SPIEEPROM access).                                                                  */
/*******************************************************************************************************************/
com_delay(10) ;
     /* ##(program note)################################################################################################ */
     /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from the end of data transfer     ## */
     /* ## to negation of CS#.                                                                                          ## */
     /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
     /* ## to set the appropriate waiting time.                                                                         ## */
     /* ################################################################################################################ */

IO.PDR1.BYTE = 0x01 ;

/*******************************************************************************************************************/
/*******************************************************************************************************************/
/*   Specifies the SPI EEPROM write enable.                                                                        */
/*******************************************************************************************************************/
/*******************************************************************************************************************/
/*******************************************************************************************************************/
/*   Drives CS# low (enables SPIEEPROM access).                                                                    */
/*******************************************************************************************************************/
com_delay(10) ;
     /* ##(program note)################################################################################################ */
     /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval after negation of CS#             ## */
     /* ## until it is asserted again.                                                                                  ## */
     /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
     /* ## to set the appropriate waiting time.                                                                         ## */
     /* ################################################################################################################ */
IO.PDR1.BYTE = 0x00 ;
com_delay(10) ;
     /* ##(program note)################################################################################################ */
     /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from assertion of CS#             ## */
     /* ## to starting the data transfer.                                                                               ## */
     /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
     /* ## to set the appropriate waiting time.                                                                         ## */
     /* ################################################################################################################ */

/*******************************************************************************************************************/
/*   Cancels the SPI EEPROM write enable.                                                                          */
/*******************************************************************************************************************/
ret = set_write_data (RESET_WRITE_ENABLE) ;
if (ret !=0) { goto exit ;}

/*******************************************************************************************************************/
/*   Executes the end processing.                                                                                  */
/*******************************************************************************************************************/
ret = set_spi_trans_end () ;
if (ret !=0) { goto exit ;}

/*******************************************************************************************************************/
/*   Drives CS# high (disables SPIEEPROM access).                                                                  */
/*******************************************************************************************************************/
com_delay(10) ;
```

```
        /* ##(program note)############################################################################################ */
        /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from the end of data transfer     ## */
        /* ## to negation of CS#.                                                                                          ## */
        /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
        /* ## to set the appropriate waiting time.                                                                         ## */
        /* ############################################################################################################## */

    IO.PDR1.BYTE = 0x01 ;

    /*****************************************************************************************************************/
    /*   Checks for the end of write.                                                                                */
    /*****************************************************************************************************************/
   ret = wait_spi_write_end () ;
    if (ret !=0) { goto exit ;}
        /* ##(program note)############################################################################################ */
        /* ## SPIEEPROM starts write operation by CS = high.  The end of write is detected                                 ## */
        /* ## by checking the SPIEEPROM internal status register since the write operation takes some time.                ## */
        /* ############################################################################################################## */

    return (ret);

exit :                                                                              /* Error processing                */
    /*****************************************************************************************************************/
    /*   Drives CS# high (disables SPIEEPROM access).                                                                */
    /*****************************************************************************************************************/
    com_delay(10) ;
        /* ##(program note)############################################################################################ */
        /* ## Inserts a wait here for the purpose of allowing at least a 500 ns interval from the end of data transfer     ## */
        /* ## to negation of CS#.                                                                                          ## */
        /* ## Since the waiting time differs according to the device to be controlled, refer to the data sheet             ## */
        /* ## to set the appropriate waiting time.                                                                         ## */
        /* ############################################################################################################## */

    IO.PDR1.BYTE = 0x01 ;

    /*****************************************************************************************************************/
    /*   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.         */
    return (ret);

}
```

```
/* ------------------------------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------- */
/* 4.  Sample program 4-D  TimerZ processing ------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------------------------------------------------------- */

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

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

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

     TZ0.TCR.BYTE   = 0x23 ;
          /* CCLR[2:0] = 001       Clears the counter on a GRA compare-match.                                             */
          /* CKEG[1:0] = 00        Counts at the rising edge.                                                             */
          /* TPSC[2:0] = 011       Counts using internal clock ϕ/8.                                                       */
     TZ0.TIORA.BYTE = 0x00 ;
          /* IOA[2:0] = 000        GRA is used as an output compare reregister.                                           */
     TZ0.TIER.BYTE  = 0x01 ;
          /* IMIEA  = 1            Enables IMFA.                                                                          */

     TZ0.GRA        = 20000 ;                                            /* 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.                                                                       */

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

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

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

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

    /*****************************************************************************************************************/
    /*   Increments the counter.                                                                                     */
    /*****************************************************************************************************************/
    com_timer.counter++;
    if( com_timer.counter >= 10 ){
         /************************************************************************************************************/
         /*   Decrement by 1 every 100 msec.                                                                         */
         /************************************************************************************************************/
         if( com_timer.wait_100ms>0 )
              com_timer.wait_100ms --;
         if( com_timer.wait_100ms_scan>0 )
              com_timer.wait_100ms_scan --;

         com_timer.counter = 0;
    }

}
```

4. Reference Documents

- H8/3687 Group Hardware Manual (published by Renesas Technology Corp.)
- X25043/45 APPLICATION NOTES (Xicor, Inc.)

Revision Record

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

Keep safety first in your circuit designs!

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

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.