To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

M16C/28, M16C/29 Group

Communication with I^2C serial EEPROM by Multi-master I^2C bus Interface

1. Abstract

The application note describes an application for M16C/28 and M16C/29 group MCU to communicate with I^2C serial EEPROM by Multi-master I^2C bus Interface.

The MCU used for current application belongs to M16C/29 group and the I^2C serial EEROM is HN58X2402SI.

The functions of the attached program are: writing data to EEPROM by byte (minimum 1 byte and maximum 8 bytes) sequentially; checking the status of internally-timed write cycle; reading data from EEPROM sequentially (minimum 1 byte and no maximum limit).

Please just take the hardware structure and parameters of the following description as a reference, and confirm or make modifications according to actual conditions in experiment or evaluation.

2. Introduction

The application example described in this document is applied to the following:

• MCU: M16C/28, M16C/29 Group

This program can be used with other microcomputers within the M16C family, which have the same SFR (special function register) as the M16C/28, and M16C/29 microcomputers. Please check the manual for any additions and modifications to functions. Careful evaluation is recommended before using this application note.

3. Specification

3.1 Multi-master I2C bus Interface

The multi-master I2C bus interface is a serial communication circuit based on Philips I2C bus data transfer format, equipped with arbitration lost detection and synchronous functions. The multi-master I2C bus interface functions are listed in table 3.1.

The multi-master I2C bus interface consists of the I2C0 Address Register (S0D0), the I2C0 Data Shift Register (S00), the I2C0 Clock Control Register (S20), the I 2C0 Control Register 0 (S1D0), the I2C0 Control Register 1 (S3D0), the I2C0 Control Register 2 (S4D0), the I2C0 Status Register (S10), the I 2C0 Start/stop Condition Control Register (S2D0) and other control circuits.

Table 3.1 Multi-master I2C bus interface functions

Item	Function
Format	Based on Philips I2C bus standard: • 7-bit addressing format • High-speed clock mode • Standard clock mode
Communication mode	Based on Philips I2C bus standard: • Master transmit • Master receive • Slave transmit • Slave receive
SCL clock frequency	16.1kHz ~ 400kHz (VIIC* = 4MHz)
I/O pin	Serial data line SDAMM(SDA) Serial clock line SDLMM (SCL)

*VIIC = I2C bus system clock

In current application, the topological structure of I2C bus is single-master and single-slave. SCL mode is standard clock mode; SCL clock frequency is 100 kHz; Communication mode is master transmit and master receive.

3.2 Operation of Renesas I2C serial EEPROM

The memory product of Renesas includes a series of I2C serial EEPROM with different package and different capacities from 2K to 1M. These products are similar in structure, function and operation. As an example, the type used in current application is HN58X2402SI, which is 2Kbits.

HN58X2402SI is two-wire serial interface EEPROM (Electrically Erasable and Programmable ROM). It realizes high speed, low power consumption and a high level of reliability by employing advanced MNOS memory technology and CMOS process and low voltage circuitry technology. It also has an 8-byte page programming function to make the write operation faster.

Please refer to figure 3.1 for the pin arrangement of HN58X2402SI, and table 3.2 for the pin description of HN58X2402SI.

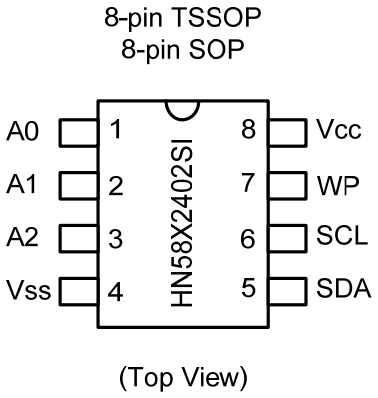


Figure 3.1 Pin Arrangement of HN58X2402SI

Table 3.2 Pin Description of HN58X2402SI

Pin name	Function
A2, A1, A0	Device address
SCL	Serial clock input
SDA	Serial data input/output
WP	Write protect
Vcc	Power supply
Vss	Ground

The device address of HN58X2402SI is 7 bits, the first 4 bits of the address is fixed to $(1010)_2$, and the last 3 bits are decided by A2, A1 and A0 pins. That is, the device address of a single EEPROM is $(1010A_2A_1A_0)_2$. Take HN58X2402SI for example, an I²C bus supports 8 chips with different device address.

In current application, the device address pins A2, A1 and A0 are set to “0”, “1”, and “1” separately. As write protect function is not used in application, WP pin is connected to ground (disable write protect function).

3.2.1 Byte Write

A write operation requires an 8-bit device address word with R/W bit which is set to “0” (write). Then the EEPROM sends acknowledgment “0” (ACK) at the 9th clock cycle (ACK clock). Then, EEPROM receives 8-bit memory address word. Upon receipt of this memory address, the EEPROM outputs acknowledgment “0” (ACK) and receives a following 8-bit data. After receipt of data, the EEPROM outputs acknowledgment “0” (ACK). If the EEPROM receives a stop condition, the EEPROM enters an internally-timed write cycle and terminates receipt of SCL, SDA inputs until completion of the internally-timed write cycle.

The time sequence chart of Byte Write operation is shown in figure 3.2.

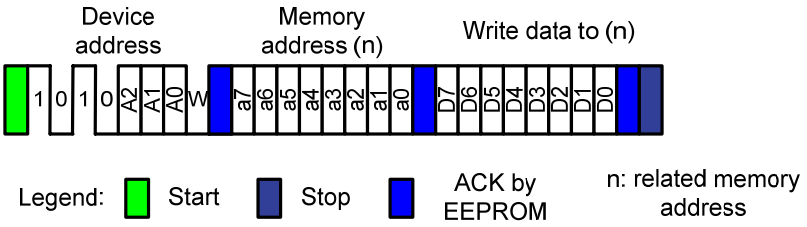


Figure 3.2 Time Sequence Chart of Byte Write Operation

3.2.2 Page Write

The EEPROM is capable of the page write operation which allows up to 8 bytes to be written in a single write cycle. The page write is the same sequence as the Byte Write except for inputting the more data.

The page write is initiated by a start condition, device address word, memory address and data with every 9th bit acknowledgment. The EEPROM enters the page write operation if the EEPROM receives more data instead of receiving a stop condition. The a0 to a2 address bits are automatically incremented upon receiving data. The EEPROM can continue to receive data up to 8 bytes. If the a0 to a2 address bits reaches the last address of the page, the a0 to a2 address bits will roll over to the first address of the same page and previous data will be overwritten. Upon receiving a stop condition, the EEPROM enters internally-timed write cycle.

The time sequence chart of Page Write operation is shown in figure 3.3.

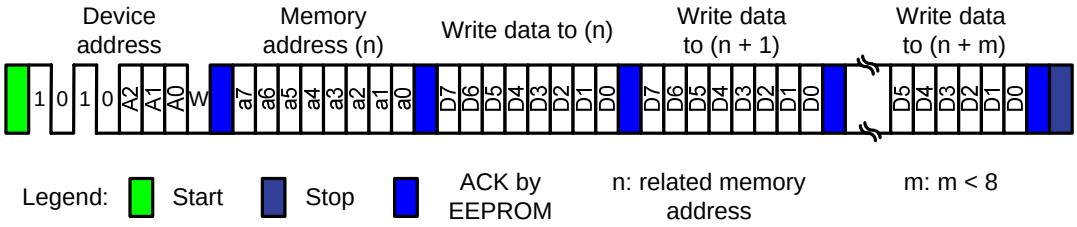


Figure 3.3 Time Sequence Chart of Page Write Operation

3.2.3 Acknowledge Polling

Whether on Byte Write operation or Page Write operation, the EEPROM enters internally-timed write cycle after receiving a stop condition. In internally-timed write cycle, EEPROM does not response any signal on I2C bus. Therefore, in order to take next Read or Write operation, it must be assured that the EEPROM is not in internally-timed write cycle.

There are two approaches to solve this problem. One approach is Delay and the other approach is Acknowledge Polling.

According to the hardware manual of HN58X2402SI, when Vcc is 2.7V ~ 5.5V, the maximum internally-timed write cycle takes 10ms. However, in most situations, the actual time expense of EEPROM is much less than the maximum value. Therefore, this approach results in a waste of time.

Acknowledge Polling will save time, although it holds I2C bus for a short time. This operation is initiated by the stop condition after inputting data. This requires the 8-bit device address word following the start condition during an internally-timed write cycle. Acknowledge polling will operate when the R/W bit is “0” (write). Acknowledgment “1” (NACK) shows the EEPROM is in an internally-timed write cycle and acknowledgment “0” (ACK) shows that the internally-timed write cycle is complete. This approach is used in current application.

The time sequence chart of Page Write operation is shown in figure 3.3.

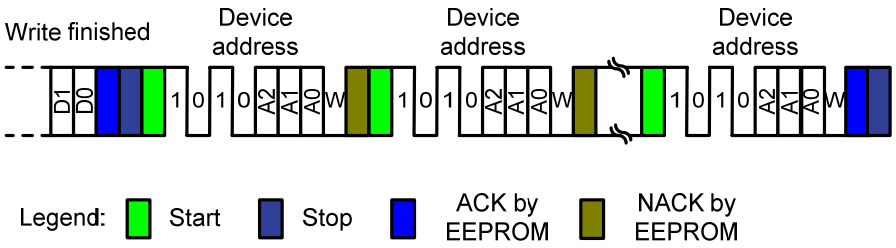


Figure 3.4 Time Sequence Chart of Acknowledge Polling

3.2.4 Current Address Read

Current Address Read accesses the address kept by the internal address counter. The internal address counter maintains the last address accessed during the last read or write operation, with incremented by one.

After receiving a start condition and the device address word with R/W bit (R/W bit is set to “1”, read), the EEPROM outputs the 8-bit current address data from the most significant bit (MSB) following acknowledgment “0” (ACK). If the EEPROM receives acknowledgment “1” (NACK) and a following stop condition, the EEPROM stops the read operation and is turned to a standby state.

In case the EEPROM has accessed the last address of the last page at previous read operation, the current address will roll over and returns to zero address. In case the EEPROM has accessed the last address of the page at previous write operation, the current address will roll over within page addressing and returns to the first address in the same page.

The time sequence chart of Page Write operation is shown in figure 3.5.

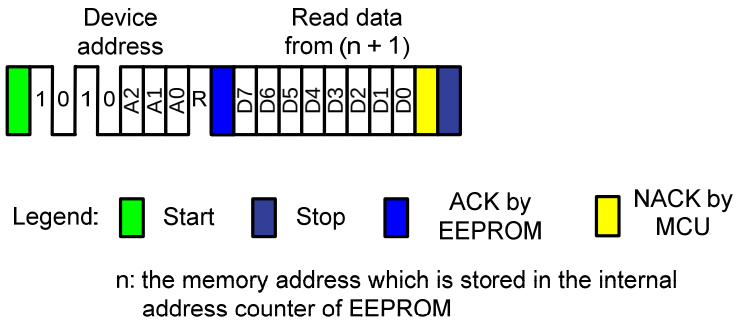


Figure 3.5 Time Sequence Chart of Current Address Read

3.2.5 Random Read

Random Read is a read operation with defined read address. It requires a write operation to set read address. The EEPROM receives a start condition, device address word with R/W bit (R/W bit is set to “0”, write) and memory address sequentially. The EEPROM outputs acknowledgment “0” (ACK) after receiving memory address then enters a current address read with receiving a start condition. The EEPROM outputs the read data of the address which was defined in the previous write operation. After receiving acknowledgment “1” (NACK) and a following stop condition, the EEPROM stops the random read operation and returns to a standby state.

The time sequence chart of Page Write operation is shown in figure 3.6.

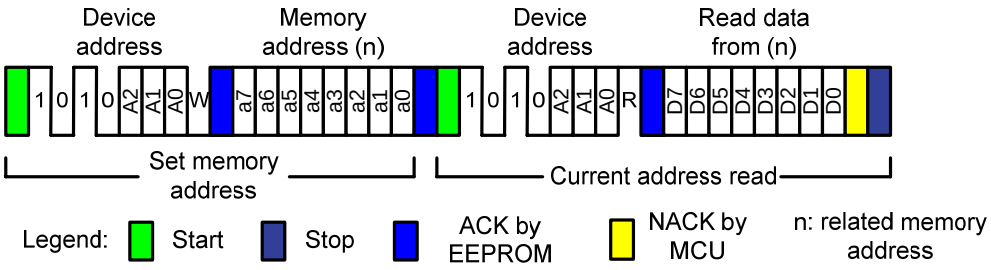


Figure 3.6 Time Sequence Chart of Random Read

3.2.6 Sequential Read

Sequential Read is initiated by either a current address read or a random read. If the EEPROM receives acknowledgment “0” (ACK) after 8-bit read data, the read address is incremented and the next 8-bit read data are coming out. This operation can be continued as long as the EEPROM receives acknowledgment “0” (ACK).

The address will roll over and returns address zero if it reaches the last address of the last page. The sequential read can be continued after roll over. The sequential read is terminated if the EEPROM receives acknowledgment “1” (NACK) and a following stop condition.

The time sequence chart of Page Write operation is shown in figure 3.7.

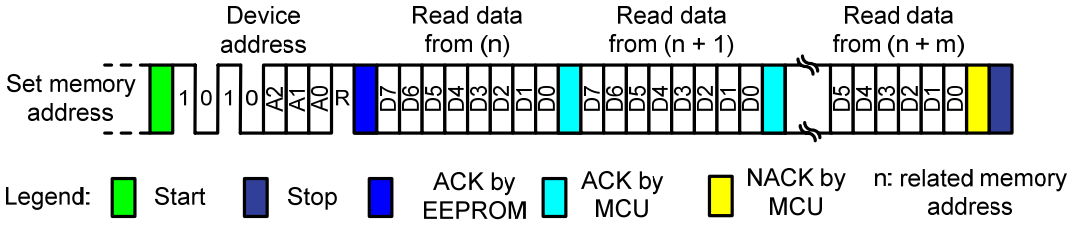


Figure 3.7 Time Sequence Chart of Sequential Read

4. Design Description

4.1 Hardware structure

In current application, the structure of hardware system is shown in figure 4.1.

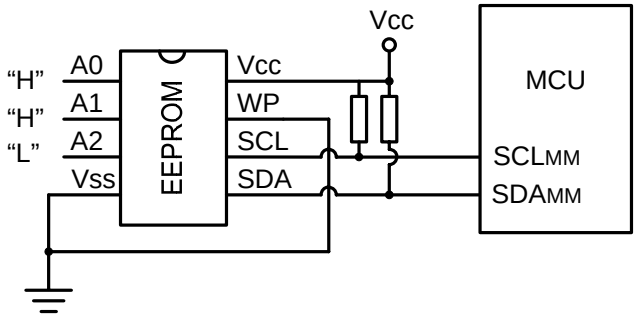


Figure 4.1 Hardware Structure of Application

In figure 4.1, the device address pin of EEPROM A2, A1 and A0 are set to “0”, “1” and “1” separately. The pull-up registers for I²C bus should be selected according to I²C Bus Specification.

4.2 Time-Sequence and Structure of Program

The functions of the attached program are: writing data to EEPROM by byte (minimum 1 byte and maximum 8 bytes) sequentially; checking the status of internally-timed write cycle; reading data from EEPROM sequentially (minimum 1 byte and no maximum limit). The state transition diagram of I²C handling program is shown in figure 4.2.

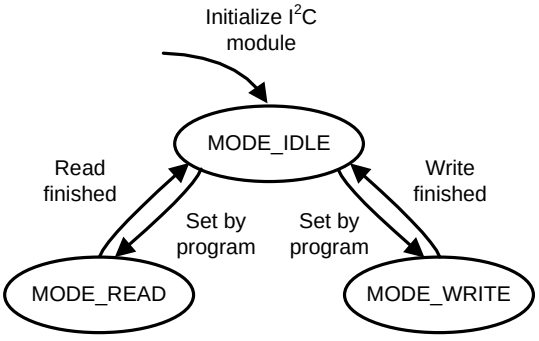


Figure 4.2 State Transition Diagram of I²C Handling Program

The time sequence of write operation (no matter Byte Write or Page Write) with Acknowledge Polling is shown in figure 4.3. Accordingly, the state transition diagram of sub-mode on write operation is shown in figure 4.4.

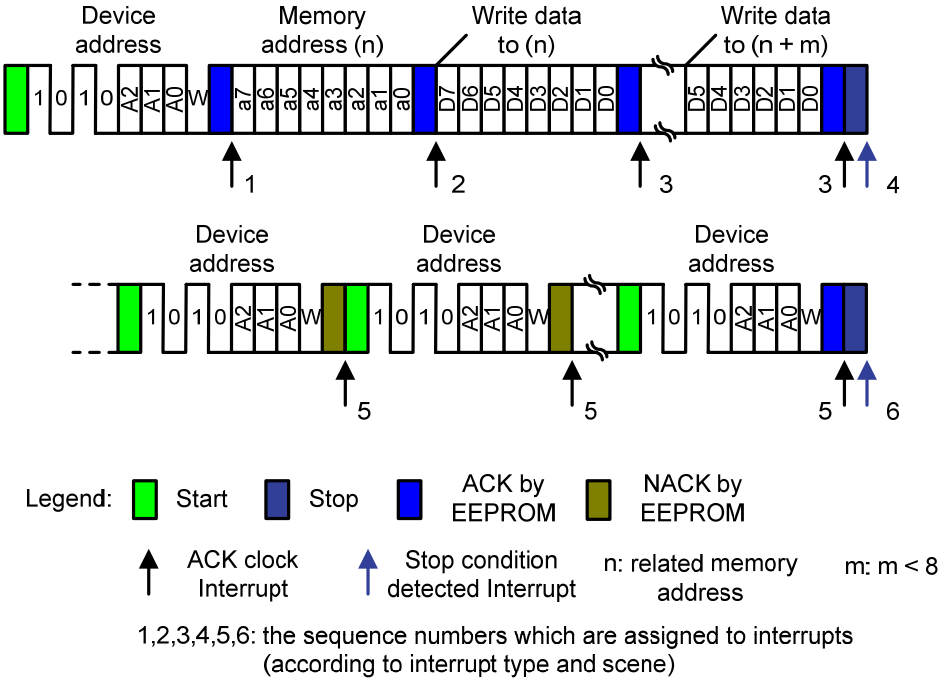
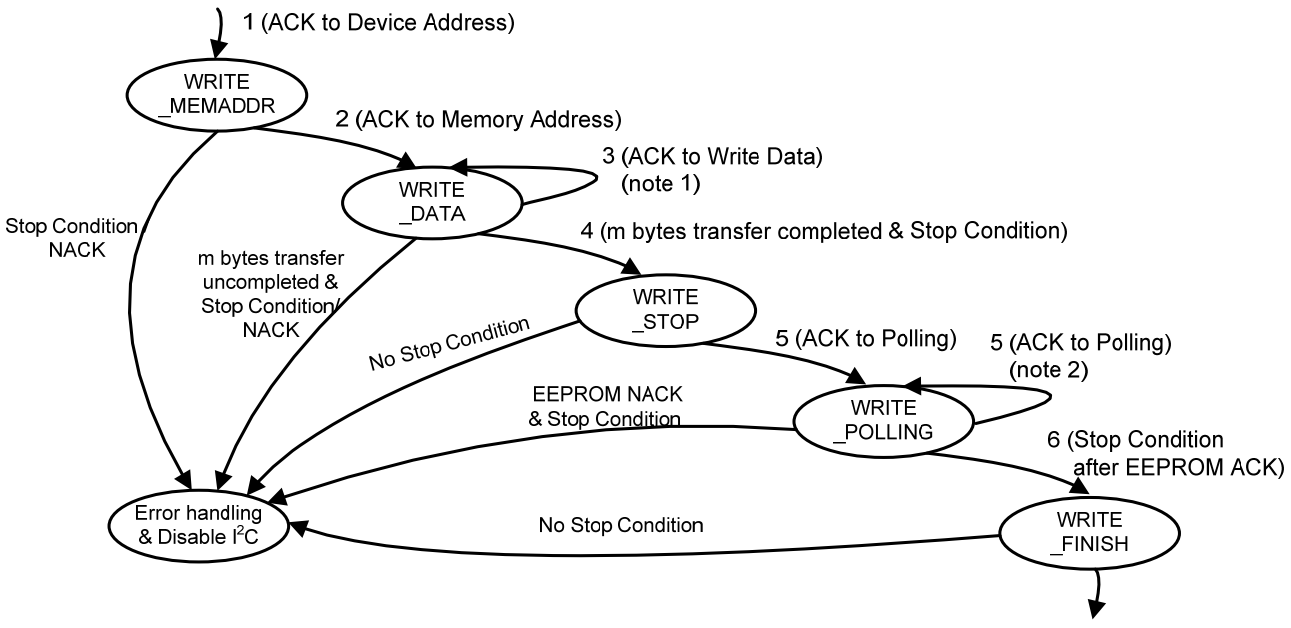


Figure 4.3 Time Sequence of Write Operation with Acknowledge Polling



1 to 6: the sequence numbers which are assigned to interrupts
Note 1: the status of transfer completed / uncompleted are judged in WRITE_DATA state
Note 2: the status of ACK / NACK are judged in WRITE_POLLING state

Figure 4.4 State Transition Diagram of Sub-mode on Write Operation

In figure 4.4, the transition of sub-modes is triggered by interrupts which are marked in figure 4.3 accordingly. If unexpected interrupts are received, MCU terminated I²C operation.

The time sequence of read operation is shown in figure 4.5. Accordingly, the state transition diagram of sub-mode on read operation is shown in figure 4.6.

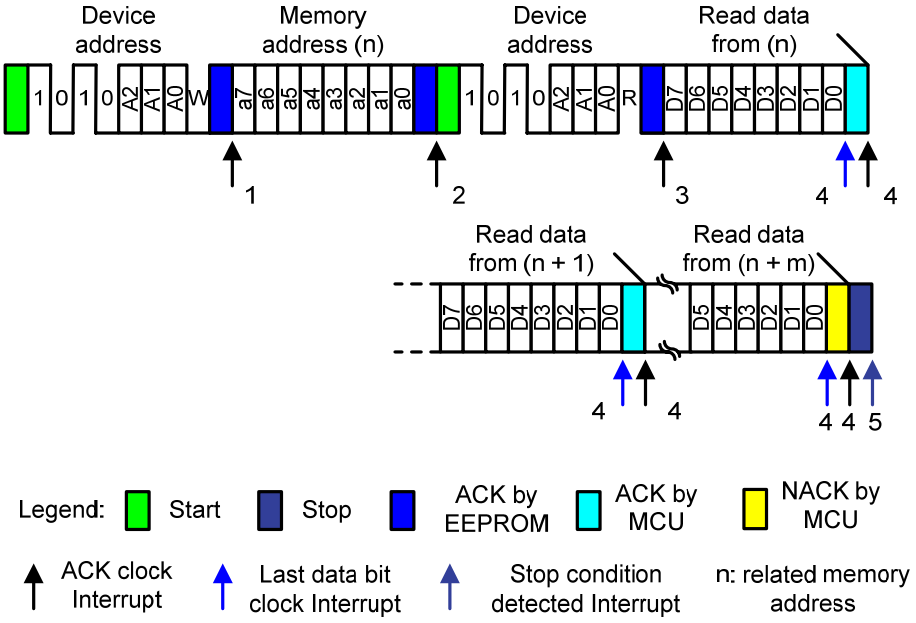


Figure 4.5 Time Sequence of Read Operation

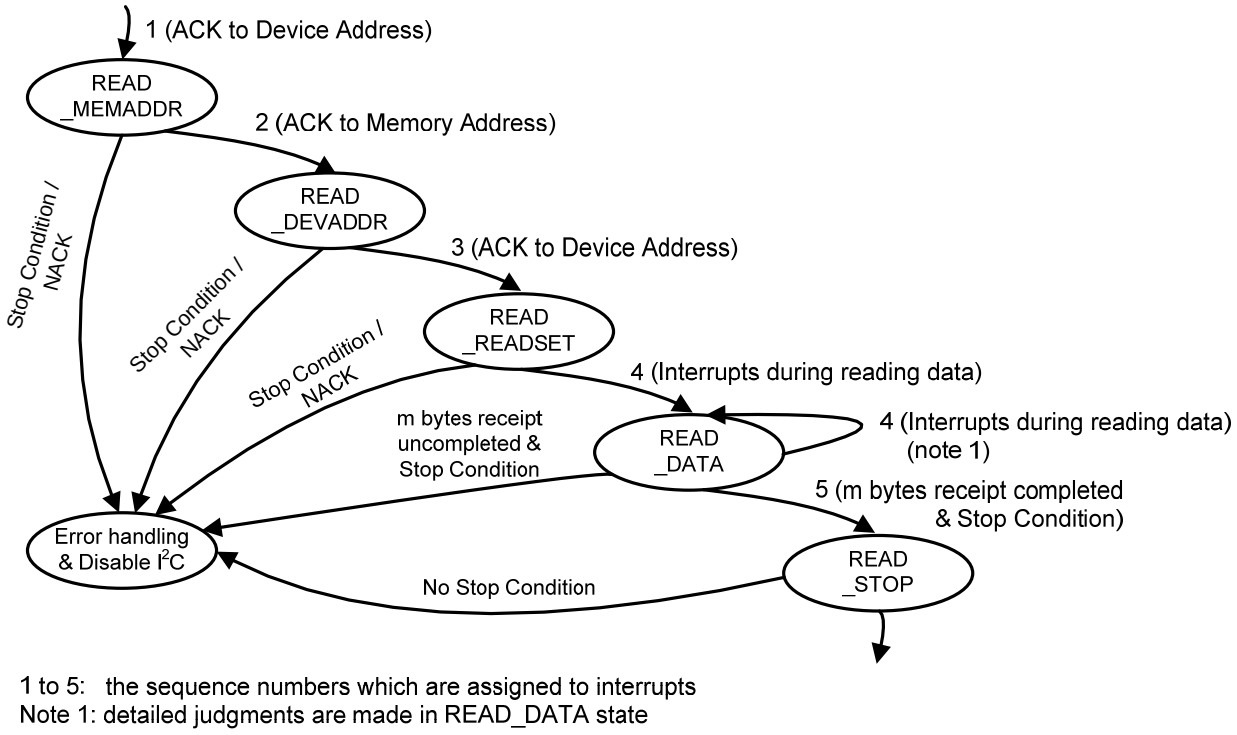


Figure 4.6 State Transition Diagram of Sub-mode on Read Operation

In figure 4.6, the transition of sub-modes is triggered by interrupts which are marked in figure 4.5 accordingly. If unexpected interrupts are received, MCU terminated I²C operation.

5. Sample Program

```
/*****************************************************************/
/*    Project         : M16C/28, M16C/29 Application Note        */
/*                    : Communication with I2C serial EEPROM     */
/*                    : by Multi-master I2C bus Interface        */
/*    MCU             : M30290FCTHP                              */
/*    C Compiler      : NC30WA, version 5.40                     */
/*    File name       : i2c_eeprom.c                             */
/*    Function        : Read / Write I2C Serial EEPROM           */
/*    Code Version    : 1.0                                      */
/*                                                               */
/*    Copyright (C) 2007 Renesas Technology Corp.                */
/*    All right reserved.                                        */
/*****************************************************************/

/*****************************************************************/
/*    Header Including                                           */
/*****************************************************************/
 #include "sfr29.h"

/*****************************************************************/
/*    Macro Definition                                            */
/*****************************************************************/
 #define EEPROM_ADDR   0b01010011    /*  EEPROM Device Address       */
                                     /* (lower 7 bits)               */
 #define WRITE         0x0
 #define WRITE_ADDR    0b1111000     /*  the memory to be write to   */
 #define READ          0x1
 #define READ_ADDR     0b1111000     /*  the memory to be read from  */

/*****************************************************************/
/*    Function Declaration                                        */
/*****************************************************************/
 void main(void);

 extern void iic_ini(unsigned char);
 extern void init_main(void);
 extern unsigned char eeprom_operation(unsigned char slave,
                            unsigned char memory,
                            unsigned char *buf,
                            unsigned char len,
                            unsigned char rw);
 extern void eeprom_delay(void);
```

```
/*************************************************************/
/*    Variable Definition                                    */
/*************************************************************/
 unsigned char iic_tr[8] = {1,2,3,4,5,6,7,8};   /*  buffer for transfer  */
 unsigned char iic_re[8] = {0,0,0,0,0,0,0,0};   /*  buffer for receive   */

/*************************************************************
      Function:      main
      Description:   main function
      Calls:         init_main(void)
                     ini_ini(unsigned char)
                     unsigned char eeprom_operation(unsigned char slave,
                                    unsigned char memory,
                                    unsigned char *buf,
                                    unsigned char len,
                                    unsigned char rw);
      Input:         None
      Output:        None
      Return:        None
*************************************************************/

 void main()
 {
   init_main();           /*  Initialize MCU          */
    iic_ini(1);           /*  Initialize I2C bus      */
    asm("fset I");        /*  Enable interrupt        */

   /* write data       */
    if(eeprom_operation(EEPROM_ADDR,WRITE_ADDR,iic_tr,8,WRITE)!=0) {;
    }
   else {;
    }

   eeprom_delay();        /*  waiting for the end of writing  */

   /* read data        */
    if(eeprom_operation(EEPROM_ADDR,READ_ADDR,iic_re,8,READ)!=0) {;
    }
   else {;
    }

   eeprom_delay();        /*  waiting for the end of reading  */
```

```
   iic_ini(0);             /*  disable I2C bus function */

   while(1){
   }
 }

/************************************************************
      Function:      init_mcu
      Description:   initialize MCU
      Calls:         None
      Input:         None
      Output:        None
      Return:        None
************************************************************/
void init_main(void)
{
   /* Setting system clock related registers */
   /* Setting system clock related registers */
   prcr = 0x01;                  /* cm0,cm1,cm2 writing enable          */
   cm2  = 0x00;                  /* system register2 Initialize         */
   cm1  = 0x20;                  /* system register1 Xcin-Xcout:High    */
   cm0  = 0x08;                  /* system register0 Xcin-Xcout:High    */
   prcr = 0x00;                  /* cm0,cm1,cm2 writing disable         */

   prcr = 0x04;                  /* pacr writing enable                 */
   pacr = 0x03;                  /* 80pin type                          */
   prcr = 0x00;                  /* pacr writing disable                */
}
```

```
/************************************************************/
/*    Project         : M16C/28, M16C/29 Application Note   */
/*                    : Communication with I2C serial EEPROM*/
/*                    : by Multi-master I2C bus Interface   */
/*    MCU             : M30290FCTHP                         */
/*    C Compiler      : NC30WA, version 5.40                */
/*    File name       : i2C_functions.c                     */
/*    Function        : Read / Write I2C Serial EEPROM      */
/*    Code Version    : 1.0                                 */
/*                                                          */
/*    Copyright (C) 2007 Renesas Technology Corp.           */
/*    All right reserved.                                   */
/************************************************************/

/************************************************************/
/*    Header Including                                      */
/************************************************************/
#include "sfr29.h"

/************************************************************/
/*    Macro definition                                      */
/************************************************************/
#define  BYTE_LIMIT    8      /*  limit of continuous write   */

unsigned char iic_mode;       /*  I2C bus working modes          */
#define MODE_IDLE    0x00
#define MODE_READ    0x01
#define MODE_WRITE   0x02

unsigned char submode;        /*  I2C bus working sub modes      */
#define WRITE_MEMADDR  0x00
#define WRITE_DATA     0x01
#define WRITE_STOP     0x02
#define WRITE_POLLING  0x03
#define WRITE_FINISH   0x04
#define READ_MEMADDR   0x05
#define  READ_DEVADDR  0x06
#define READ_READSET   0x07
#define READ_DATA      0x08
#define READ_STOP      0x09

unsigned char ErrorCode;      /*  error code                  */
```

```
#define NORMAL          0x00
#define TOF_ERROR       0x01
#define STOP_ERROR      0x02
#define NACK_ERROR      0x03
#define UNKNOWN_ERROR   0x04

/************************************************************/
/*    Function declaration                                  */
/************************************************************/
         void iic_ini(unsigned char);
unsigned char eeprom_operation(unsigned char slave,unsigned char memory,
                        unsigned char *buf,unsigned char len,
                        unsigned char rw);
         void eeprom_delay(void);
         void iic_int(void);

  static void master_transfer(void);
  static void master_receive(void);

/************************************************************/
/*    Variable definition                                   */
/************************************************************/
typedef union{
   struct{
      unsigned char b0:1;
   }bit;
   unsigned char all;
}byte_dt;

static byte_dt iic_sl;                /* address and control word          */
#define iic_slave iic_sl.all
#define iic_rw iic_sl.bit.b0          /* 0:write   1:read                  */

static unsigned char iic_length;      /* data (bytes) to be transferred    */
static unsigned char iic_memaddr;     /* Memory of store address           */
static unsigned char *iic_pointer;    /* pointer of transfer/receive buffer */

/************************************************************
      Function:      iic_ini
      Description:   initialize I2C bus
      Calls:         None
      Input:         0: Disable I2C module
                     1: Enable I2C module
```

```
      Output:        None
      Return:        None
*****************************************************************/
void iic_ini(unsigned char ini)
{
   asm("pushc FLG");   /*  Protect FLG register                  */

   if(ini == 1) {      /*  initialize and enable I2C bus module  */
      s1d0 = 0x00;     /*  8-bit data, addressing format,     */
                       /*  reset release, I2C bus input       */
      s10 = 0x00;      /*  initialize I2C0 Status Register    */
      s20 = 0x85;      /*  with ACK clock, ACK is returned,   */
                        /* standard clock mode                */
                        /* CLK frequency 100 kHz @ VIIC=4MHz  */
      s3d0 = 0x03;     /*  enable STOP condition detection interrupt */
                       /*  enable data receive completion interrupt  */
      s4d0 =0x19;      /*  VIIC = 1/5 fIIC                    */
                       /*  enable time out detection, long time      */
      s2d0 = 0x98;     /*  set detection condition of         */
                       /*  START/STOP condition               */

       asm("fclr I");          /*  disable interrupt                    */
      ifsr27 = 1;              /*  set I2C bus interrupt priority level  */
      iicic = 0x01;            /*  enable I2C bus interrupt              */

      iic_mode = MODE_IDLE;    /*  set I2C bus working mode as IDLE      */
      ErrorCode = NORMAL;      /*  set error code as NORMAL              */
       es0 = 1;                /*  enable I2C bus function               */
   }
   else {                      /*  disable I2C bus module                */
       asm("fclr I");          /*  disable interrupt                     */
       iicic = 0x00;           /*  disable I2C bus interrupt             */
       iic_mode = MODE_IDLE;   /*  set I2C bus working mode as IDLE      */
       es0 = 0;                /*  enable I2C bus module                 */
   }
   asm("popc FLG");            /*  restore FLG register (enable interrupt)*/
}

/*****************************************************************
      Function:      eeprom_operation
      Description:   trigger I2C bus operation (Read / Write)
      Calls:         None
      Input:         unsigned char slave    : device address
```

```
                    unsigned char memaddr  : memory address
                    unsigned char *buffer  : pointer of buffer
                    unsigned char length   : data length (bytes)
                    unsigned char rw       : R/W bit
      Output:       None
      Return:       0: fail to start operation
                    1: success to start operation
***************************************************************/

unsigned char eeprom_operation(unsigned char slave,  unsigned char memaddr,
                               unsigned char *buffer, unsigned char length,
                               unsigned char rw)
{
   if((bb == 1) || (iic_mode != MODE_IDLE)) {
      return(0);                    /*  fail to start operation        */
   }
   else {
      asm("pushc FLG");             /*  protect FLG register           */
      asm("fclr I");                /*  disable interrupt              */
      iic_slave = slave << 1;       /*  set device address             */
      iic_rw = 0;                   /*  set read/write bit             */
      iic_length = length;          /*  set data (bytes) in operation  */
      iic_pointer = buffer;         /*  set buffer pointer             */
      iic_memaddr = memaddr;        /*  set memory address             */

      if(rw == 0) {                     /*  write operation            */
         if(iic_length > BYTE_LIMIT) {  /*  limit the data (bytes) in  */
                                        /*  page write operation       */
            iic_length = BYTE_LIMIT;
         }
         iic_mode = MODE_WRITE;
         submode = WRITE_MEMADDR;
      }
      else {                            /*  read operation             */
         iic_mode = MODE_READ;
         submode = READ_MEMADDR;
      }

       s10 = 0xE0;                  /*  start condition                */
       s00 = iic_slave;
       asm("popc FLG");             /*  restore FLG register           */
       return(1);                   /*  success starting operation     */
   }
```

```
}

/************************************************************************
      Function:      iic_int
      Description:   I2C interrupt handler
      Calls:         void master_transfer(void)
                     void master_receive(void)
                     void iic_ini(void)
      Input:         None
      Output:        None
      Return:        None
************************************************************************/

#pragma INTERRUPT iic_int
void iic_int(void) {

   if(tof) {                        /*  time out interrupt       */
      ErrorCode = TOF_ERROR;        /*  set error code           */
      iic_ini(0);                   /*  disable I2C bus module   */
      return;                       /*  exit interrupt           */
   }

   /* checking I2C bus working mode  */
   switch (iic_mode) {
       case MODE_WRITE:
          master_transfer();
          break;
       case MODE_READ:
          master_receive();
          break;
       case MODE_IDLE:
          break;
       default:
          break;
   }
}

/************************************************************************
      Function:      master_transfer
      Description:   I2C bus write operation with acknowledge polling
      Calls:         None
      Input:         None
      Output:        None
```

```
      Return:        None
***********************************************************/
static void master_transfer(void) {

   /* checking sub mode under write mode */
   switch(submode) {

      case WRITE_MEMADDR:

         if(scpin) {                /*  receive stop condition interrupt   */
            scpin = 0;              /*  clear related interrupt flag       */
            ErrorCode = STOP_ERROR; /*  set error code                     */
            iic_ini(0);             /*  disable I2C bus module             */
            return;                 /*  exit function                      */
         }

         if(lrb == 1) {             /*  NACK received           */
            ErrorCode = NACK_ERROR; /*  set error code          */
            iic_ini(0);             /*  disable I2C bus module  */
            return;                 /*  exit function           */
         }
         else {
            s00 = iic_memaddr;      /*  send memory address     */
            submode = WRITE_DATA;   /*  change sub mode         */
         }
         break;

      case WRITE_DATA:

         if(scpin) {                /*  receive stop condition interrupt   */
            scpin = 0;              /*  clear related interrupt flag       */
            ErrorCode = STOP_ERROR; /*  set error code                     */
            iic_ini(0);             /*  disable I2C bus module             */
            return;                 /*  exit function                      */
         }

         if(lrb == 1) {             /*  NACK received           */
            ErrorCode = NACK_ERROR; /*  set error code          */
            iic_ini(0);             /*  disable I2C bus module  */
            return;                 /*  exit function           */
         }
         else {
            if(iic_length == 0) {   /*  all data (bytes) are sent   */
```

```
            s10 = 0xC0;           /*  stop condition              */
            s00 = 0xff;           /*  dummy write                 */
            submode = WRITE_STOP; /*  change sub mode             */
         }
         else{                    /*  continue sending data          */
            s00 = *iic_pointer;   /*  send 1 byte data               */
            iic_pointer++;        /*  increase buffer pointer        */
            iic_length--;         /*  decrease data (bytes) counter  */
         }
      }
      break;

   case WRITE_STOP:

      if(scpin) {                 /*  receive stop condition interrupt   */
         scpin = 0;               /*  clear related interrupt flag       */
         s10 = 0xE0;              /*  start condition                    */
         s00 = iic_slave;
         submode = WRITE_POLLING; /*  change sub mode                    */
      }
       else {                  /*  No stop condition interrupt received   */
          ErrorCode = UNKNOWN_ERROR;  /*  set error code       */
          iic_ini(0);             /*  disable I2C bus module   */
          return;                 /*  exit function            */
       }
      break;

   case WRITE_POLLING:

      if(scpin) {                 /*  receive stop condition interrupt   */
         scpin = 0;               /*  clear related interrupt flag       */
         ErrorCode = STOP_ERROR;  /*  set error code                     */
         iic_ini(0);              /*  disable I2C bus module             */
         return;                  /*  exit function                      */
      }

      if(lrb == 0){               /*  ACK received      */
         s10 = 0xC0;              /*  stop condition    */
         s00 = 0xff;              /*  dummy write       */
         submode = WRITE_FINISH;  /*  change sub mode   */
      }
      else{
         s10 = 0xE0;              /*  start condition   */
```

```
            s00 = iic_slave;
         }
         break;

      case WRITE_FINISH:

         if(scpin) {                /*  receive stop condition interrupt   */
            scpin = 0;              /*  clear related interrupt flag       */
            iic_mode = MODE_IDLE;   /*  set I2C bus working mode           */
         }
          else {
             ErrorCode = UNKNOWN_ERROR;  /*  set error code       */
             iic_ini(0);            /*  disable I2C bus module   */
             return;                /*  exit function            */
          }
         break;

      default:
         break;
   }
}

/*****************************************************************
      Function:      master_receive
      Description:   I2C bus read operation
      Calls:         None
      Input:         None
      Output:        None
      Return:        None
*****************************************************************/
static void master_receive(void){

   /* checking sub mode under read mode */
   switch(submode){

      case READ_MEMADDR:

         if(scpin) {                /*  receive stop condition interrupt   */
            scpin = 0;              /*  clear related interrupt flag       */
            ErrorCode = STOP_ERROR; /*  set error code           */
            iic_ini(0);             /*  disable I2C bus module   */
            return;                 /*  exit function            */
         }
```

```
    if(lrb == 1) {              /*  NACK received           */
       ErrorCode = NACK_ERROR;  /*  set error code          */
       iic_ini(0);              /*  disable I2C bus module  */
       return;                  /*  exit function           */
    }
    else {
       s00 = iic_memaddr;       /*  send memory address     */
       submode = READ_DEVADDR;  /*  change sub mode         */
    }
    break;

case READ_DEVADDR:

    if(scpin) {                 /*  receive stop interrupt          */
       scpin = 0;               /*  clear related interrupt flag    */
       ErrorCode = STOP_ERROR;  /*  set error code          */
       iic_ini(0);              /*  disable I2C bus module  */
       return;                  /*  exit function           */
    }

    if(lrb == 1) {              /*  NACK received           */
       ErrorCode = NACK_ERROR;  /*  set error code          */
       iic_ini(0);              /*  disable I2C bus module  */
       return;                  /*  exit function           */
    }
    else {
       iic_rw = 1;              /*  change read/write bit   */
       s10 = 0xE0;              /*  start condition         */
       s00 = iic_slave;
       submode = READ_READSET;  /*  change sub mode         */
    }
    break;

case READ_READSET:

    if(scpin) {                 /*  receive stop condition interrupt   */
       scpin = 0;               /*  clear related interrupt flag       */
       ErrorCode = STOP_ERROR;  /*  set error code          */
       iic_ini(0);              /*  disable I2C bus module  */
       return;                  /*  exit function           */
    }
```

```
      if(lrb == 1) {              /*  NACK received           */
         ErrorCode = NACK_ERROR;  /*  set error code          */
         iic_ini(0);              /*  disable I2C bus module  */
         return;                  /*  exit function           */
      }
      else {
         s10 = 0xa0;              /*  set I2C bus receive mode    */
         s00 = 0xff;              /*  dummy write                 */
         submode = READ_DATA;     /*  change sub mode             */
      }
      break;

   case READ_DATA:

      if(scpin) {                 /*  receive stop condition interrupt   */
         scpin = 0;               /*  clear related interrupt flag       */
         ErrorCode = STOP_ERROR;  /*  set error code          */
         iic_ini(0);              /*  disable I2C bus module  */
         return;                  /*  exit function           */
      }

      if(wit == 1) {              /*  interrupt on the falling edge of   */
                                  /*  the last bit of data clock         */
         iic_length--;
         if(iic_length == 0) {
            ackbit = 1;           /*  NACK on the next interrupt  */
          }
         else {
           ackbit = 0;           /*   ACK on the next interrupt   */
         }
         return;                  /*  exit function    */
      }
      else {                      /*  interrupt for ACK clock     */
         *iic_pointer = s00;      /*  receive data                */
         iic_pointer++;           /*  increase the pointer of buffer  */

         if(iic_length == 0) {    /*  the last data is received   */
            ackbit = 0;           /*  ACK on the next ACK clock interrupt*/
            submode = READ_STOP;  /*  change sub mode             */
            s10 = 0xC0;           /*  stop condition              */
            s00 = 0xff;           /*  dummy write                 */
         }
         else {                   /*  still have data to be received  */
```

```
                s00 = 0xff;          /*  dummy write                    */
             }
          }
          break;

       case READ_STOP:

          if(scpin) {                /*  receive stop condition interrupt   */
             scpin = 0;              /*  clear related interrupt flag       */
             iic_mode = MODE_IDLE;   /*  set I2C bus working mode           */
          }
          else {                     /*  No stop condition interrupt received*/
             ErrorCode = UNKNOWN_ERROR;  /*  set error code       */
             iic_ini(0);             /*  disable I2C bus module   */
             return;                 /*  exit function            */
          }
          break;

       default:
          break;
    }
}

/*****************************************************************
       Function:      eeprom_delay
       Description:   wait while I2C bus busy or not in IDLE mode
       Calls:         None
       Input:         None
       Output:        None
       Return:        None
*****************************************************************/

void eeprom_delay(void){
   while((bb == 1) || (iic_mode != MODE_IDLE)){;
   }
}
```

6. Reference

Renesas Technology Corporation Home Page

http://www.renesas.com

Enquiry

http://www.renesas.com/inquiry

csc@renesas.com

Hardware Manual

M16C/28 Group Hardware Manual

M16C/29 Group Hardware Manual

HN58X2402SI Hardware Manual

(Use the latest version on the home page: http://www.renesas.com)

Technical Updates / Technical News

(Use the latest information on the home page: http://www.renesas.com)

REVISION HISTORY

Rev.	Date	Description	
		Page	Summary
1.00	2007.04.05	-	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corp. 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 Corp. 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 Corp. or a third party.
2. Renesas Technology Corp. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. by various means, including the Renesas Technology Corp. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. for further details on these materials or the products contained therein.