To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

H8/300L

I²C™ EEPROM access thru Port Emulation (I2Ceeprom)

Introduction

This application note provides an overview of the I²C bus interface and the details of an application interfacing the SLP series 38024 (I²C Master) to a two-wire Mircochip 24AA16 16K I²C Serial EEPROM (I²C Slave).

Target Device

H8/300L Super Low Power (SLP) series – H8/38024

Contents

1. I²C™ Interface Overview

The I²C bus uses a two-wire interface consisting of a serial data line (SDA) and a serial clock line (SCL) to exchange information between devices connected to the bus. Each device on the bus has its own unique address and can operate as a transmitter or receiver (depending on its particular function).

Devices are further characterized as Masters or Slaves. A Master is defined as a device that initiates, controls (generates all framing and clock signals), and terminates a transfer. A Slave is defined as any device addressed by a Master.

(The general features of I²C is discussed in Application Note on SPI and I²C)

The following will highlight 3 main topics:

1. I2C control,
2. Write operation, and
3. Read operation.

1.1 I2C EEPROM Control

1.1.1 START & STOP Condition

Data transfers on the I²C bus are controlled (and framed) via two unique bus states generated by the Bus Master. When the bus is free, both lines are HIGH. These bus states are the START and STOP bit conditions.

A START condition is defined as a High-to-Low level transition on SDA while the SCL line is High. A STOP condition is defined as a Low-to-High level transition on SDA while the SCL line is High. Data must always be valid (stable) on the SDA line while SCL is high. The SDA line is only allowed to change during the SCL Low period. One bit of data is transmitted for each SCL period.

Following the START condition, the first 8-bit byte sent in a bus message is a 7-bit Slave address field along with a data direction or R/W bit. (This discussion is limited to the I²C 7-bit addressing mode.) The data direction bit (least significant bit) controls whether or not the Master transmits (0 = write) or receives (1 = read) data from the addressed Slave.

The acknowledge bit is a low-level signal placed on the SDA line by the receiving device (Master or Slave) during the Master-transmitted acknowledge clock pulse (ninth High SCL clock pulse of the byte transmission). If the receiver is unable to receive data (busy Slave receiver) or must signal the end of data condition (Master receiver), a *non-acknowledge* is sent (SDA High during the ninth High SCL clock pulse time).

Following the START and Slave address transmission, data is exchanged between the Master and receiver as required. Upon exchange of the final byte and its acknowledge, the Master issues the STOP condition to end bus usage.

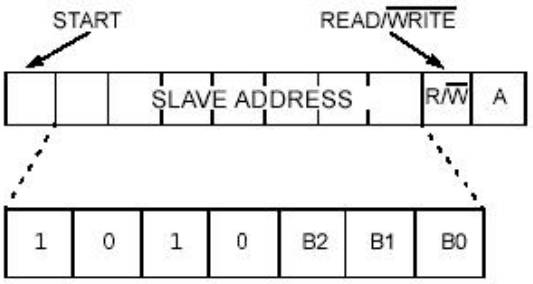
The previous general overview of I²C bus operation provides the foundation necessary to proceed with a serial EEPROM interface (a working knowledge of the I²C 7-bit addressing mode).

(Please refer to Application Note on SPI and I²C for I²C configuration in detail.)

(See also the I²C Bus Specifications published by Phillips Corporation for complete details of all the various modes this bus supports.)

1.1.2 Device Addressing

A control byte is the first byte received following the start condition from the master device. The control byte consists of four bit control code, for the 24AA16, this is set as 1010 binary for read and write operations.



The next three bits of the control byte are the block select bits (B2, B1, B0). They are used by the master device to select which of the eight 256 word blocks of memory are to be accessed. These bits are in effect the three most significant bits of the word address.

The last bit of the control byte defines the operation to be performed. When set to '1', a READ operation is selected. When set to '0', a write operation is selected. Following the START condition, the 24AA16 monitors the SDA bus checking the device type identifier being transmitted, upon 1010 code the slave device output an acknowledge signal on the SDA line. Depending on the state of the R/W bit, the 24AA16 will select a READ or WRITE operation.

Operation	Control Code	Block Select	R/W
Read	1010	Block Address	1
Write	1010	Block Address	0

*Control byte is device dependent and may varies for different devices. Please refer to the device data sheet or I²C bus specification for detailed information before use.

1.1.3 Bit Transfer and Data Validity

The number of data bytes transferred between the START and STOP condition from transmitter to receiver is not limited, and determined by the master device. Each byte, which must be eight bits long, is transferred serially with the most significant bit first, and is followed by an acknowledge bit.

The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the HIGH period of the clock signal.

The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data.

1.1.4 Acknowledge

Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit.

For 24AA16, it does not generate any acknowledge bits if an internal programming cycle is in progress.

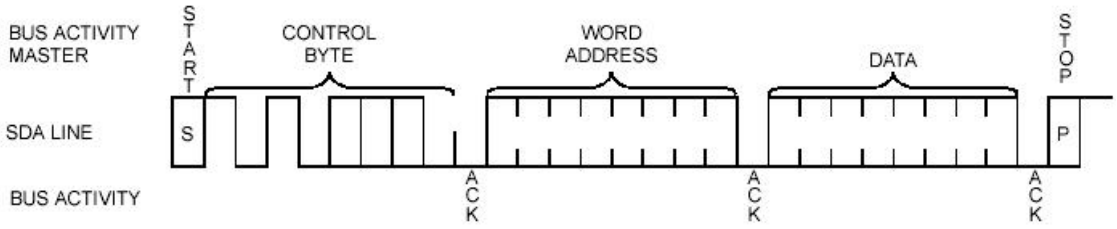
1.2 WRITE Operation

Write operations are initiated when the R/W bit of the slave address is set to '0'. There are two type of write operation: byte write and page write.

1.2.1 Byte Write

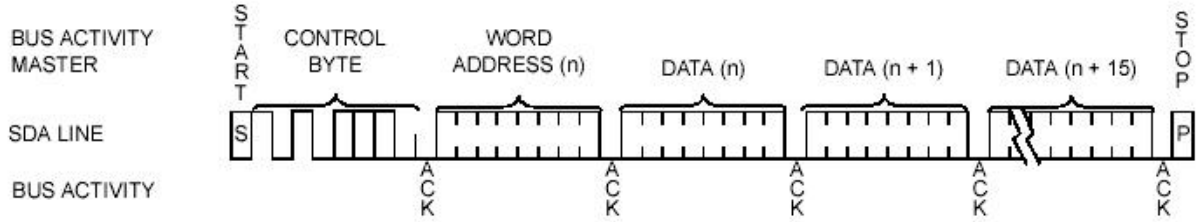
Byte operations allow a random EEPROM address to be written. Byte write operations require transmission of the following:

- START condition (Master)
- EEPROM Device Address with R/W Bit = 0 (Master)
- Acknowledge Bit (EEPROM)
- [*1]Target EEPROM Word Address to be Written (Master)
- Acknowledge Bit (EEPROM)
- Data Byte to be Written (Master)
- Acknowledge bit (EEPROM)
- STOP Condition (Master)



1.2.2 Page Write

Page write operations are identical to the byte write operations described previously, except that instead of writing one data byte, [*2]16 bytes may be sent between the EEPROM address and STOP bit transmission. During these page writes, the EEPROM automatically increments its internal address pointer between bytes.



[*1] The number of word address is device dependent. Please refer to respective device data sheet for detail.

[*2] The number of byte for page writes operation is device dependent. Please refer to respective device data sheet for verification.

1.3 READ Operation

Three types of read operations are supported: Current Address, Random, and Sequential Read. Read operations begin just like write operations, except that the R/W bit is set to 1 for the device address byte.

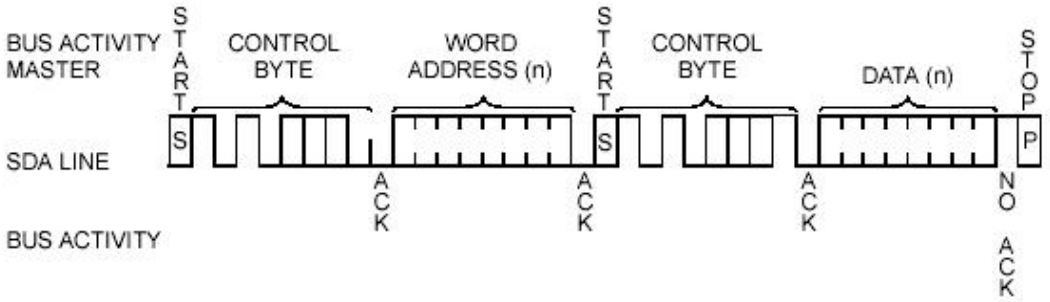
1.3.1 Current Address Read

For the Current Address Read mode, no EEPROM byte address is written as the data transmitted by the addressed Slave to the Master is read from the location of the most recent access (incremented by one). This read transmission type sequence appears as:

- START condition (Master)
- EEPROM device address with R/W bit = 1 (Master)
- Acknowledge bit (EEPROM)
- Data byte to be read (EEPROM bytes sent from the addressed Slave's most recent pointed-to memory location incremented by 1)
- Non-acknowledge bit (Master)
- STOP condition (Master)

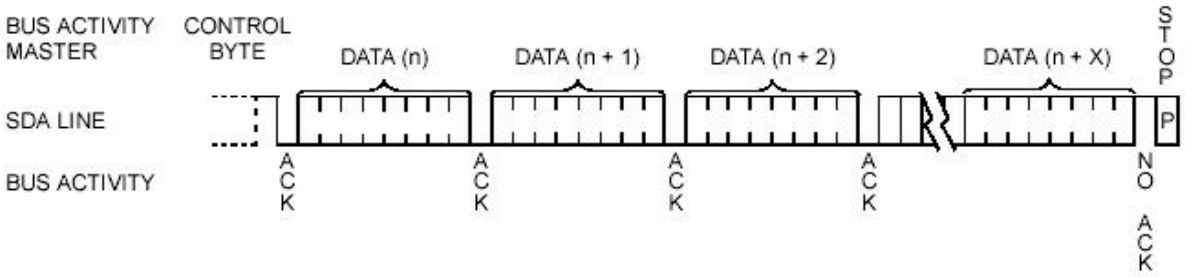
1.3.2 Random Read

The Random Read mode is begun with a dummy byte write cycle (Master sends a START condition followed by the device address and [*1]target word address) followed by a Current Address Read mode cycle as described previously.

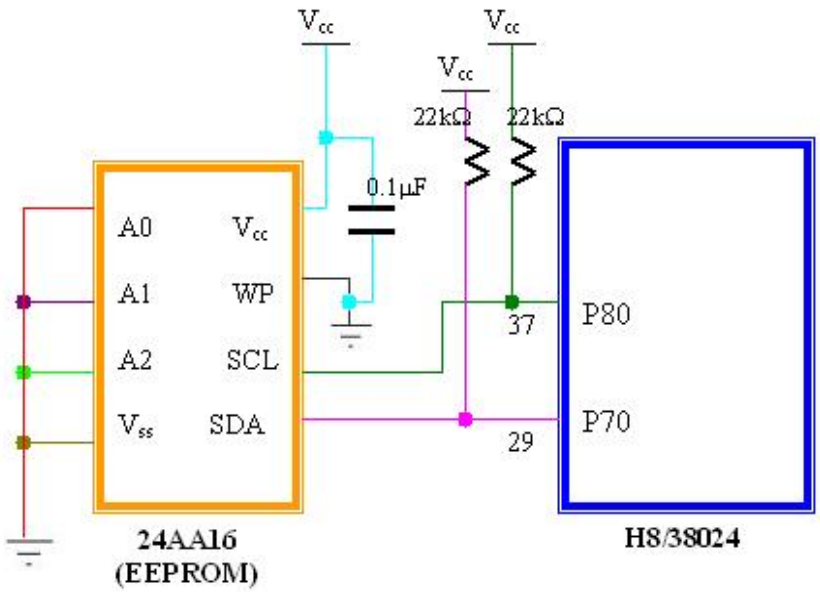


1.3.3 Sequential Read

The Sequential Read mode is initiated with a Random Read. Instead of the Master terminating the read after a single byte exchange (with a non-acknowledge), the Master responds with a valid acknowledge after each received data byte. This acknowledge instructs the Slave EEPROM to continue the read operation and transmit out the next data byte. Sequential reads continue until terminated by the Master via issuance of a non-acknowledge on the most recent byte read followed by the STOP condition



2. Hardware Design



*Signal Name

A0,A1,A2	*User Configuration Chip Selects*
V_{ss}	*Ground*
SDA	*Serial Data*
SCL	*Serial Clock*
WP	*Write Protected Input*
V_{cc}	*Supply Voltage*

3. Function Overview

Function in I2C.c:

- `void main(void)`

Functions in RW.c:

- `unsigned char SclIn (void)`
- `unsigned char SdaIn (void)`
- `void SclOut (unsigned char )`
- `void SdaOut (unsigned char)`
- `void Delay(void)`
- `void Delay2x(void)`
- `unsigned char CheckBusState(void)`
- `void SendStartBit(void)`
- `void SendBit (unsigned char)`
- `unsigned char GetBit (void)`
- `unsigned char GetAck (void)`
- `unsigned char SendByte(unsigned char)`
- `unsigned char GetByte(void)`
- `void SendStopBit(void)`
- `unsigned char I2cWrite(unsigned char, unsigned char *,unsigned char, unsigned char )`
- `unsigned char I2cRead(unsigned char, unsigned char *,unsigned char, unsigned char )`
- `char CheckWriteReady(void)`
- `unsigned char I2cCurrentRead(unsigned char , unsigned char *)`

`void main(void)`

The main function demonstrates/tests the usage of the EEPROM control with I²C protocol. The function writes into the EEPROM using byte write and page write before reading it from the EEPROM using current address read, sequential read or random read.

`unsigned char SclIn (void)`

This function contains the coding for controlling whether the port pins, SclIn as an input or output pin. Clearing the bit to 0 makes the pin an input pin. It also checks whether the SCL port is low or high.

`unsigned char SdaIn (void)`

This function contains the coding for controlling whether the port pins, SdaIn as an input or output pin. Clearing the bit to 0 makes the pin an input pin. It also checks the port status whether is in low or high.

`void SclOut (unsigned char )`

```
Parameter:
Status read from checking SCL signal level (unsigned char)
```

This function contains the coding for controlling whether the port pins, SclOut as an input or output pin. Setting the bit to 1 makes the corresponding pin an output pin.

`void SdaOut (unsigned char)`

```
Parameter:
Status read from checking SDA signal level (unsigned char)
```

This function contains the coding for controlling whether the port pins, SdaOut as an input or output pin. Setting the bit to 1 makes the corresponding pin an output pin.

`void Delay(void)`

Provide an internal minimum delay time to bridge the undefined region of a falling edge of SCL to avoid unintended generation of unwanted signal.

`void Delay2x(void)`

Provide double delay time based on function `void Delay(void).`

`unsigned char CheckBusState(void)`

This function will determine whether the I2C bus is free (both SCL and SDA= HIGH) or in busy state.

`void SendStartBit(void)`

The function will send a START condition.

`void SendBit (unsigned char)`

```
Parameter:
Data bit to be sent (unsigned char)
```

Send out data in bit format.

`unsigned char GetBit (void)`

Receive data input in bit format.

`unsigned char GetAck (void)`

Getting ACK is similar to GetBit, but this is critical operation since master must pull SDA high before it finds out whether there is a ACK (SDK is low) or not.

`unsigned char SendByte(unsigned char)`

```
Parameter:
Data in byte to be sent (unsigned char)
```

Function will send out a byte starting with most significant bit (MSB) first.

`unsigned char GetByte(void)`

Function will get a byte of data starting with most significant bit (MSB).

`void SendStopBit(void)`

Send a STOP condition to terminate the operation.

`unsigned char I2cWrite(unsigned char, unsigned char *,unsigned char, unsigned char )`

```
Parameter:
Slave address (unsigned char),
Address of buffer containing data (unsigned char*),
Word address of data to be written(unsigned char),
Length of data (unsigned char)
```

The function will perform a WRITE operation applicable for both byte write and page write by :

i) checking the bus state,

ii) send a START condition,

iii) send the control code/slave address,

iv) send the word address to be written in,

v) write the data into EEPROM based on the length of data to be written, and

vi) a STOP condition.

`unsigned char I2cRead(unsigned char, unsigned char *,unsigned char, unsigned char )`

```
Parameter:
Slave address (unsigned char),
Address of buffer containing data (unsigned char*),
Word address of data to be read from(unsigned char),
Length of data (unsigned char)
```

The function will perform a READ operation applicable for both random read and sequential read by :

i) checking the bus state,

ii) send a START condition,

iii) send the dummy control code/slave address,

iv) send the word address to be read from,

v) pull up the SDA line,

vi) a START bit,

vii) send the control code/slave address,

viii) read the data from EEPROM based on the length of data to be read, and

ix) a STOP condition.

`char CheckWriteReady(void)`

For device from Microchip such as 24AA16, the device will not acknowledge during a write cycle, thus this function can be used to determine when the cycle is complete. If the cycle is complete, the device will return the ACK and the master can then proceed with the next read or write command.

`unsigned char I2cCurrentRead(unsigned char , unsigned char *)`

```
Parameter:
Slave address (unsigned char),
Address of buffer containing data (unsigned char*),
```

The function will perform a READ operation applicable for both random read and sequential read by :

i) checking the bus state,

ii) send a START condition,

iii) send the control code/slave address,

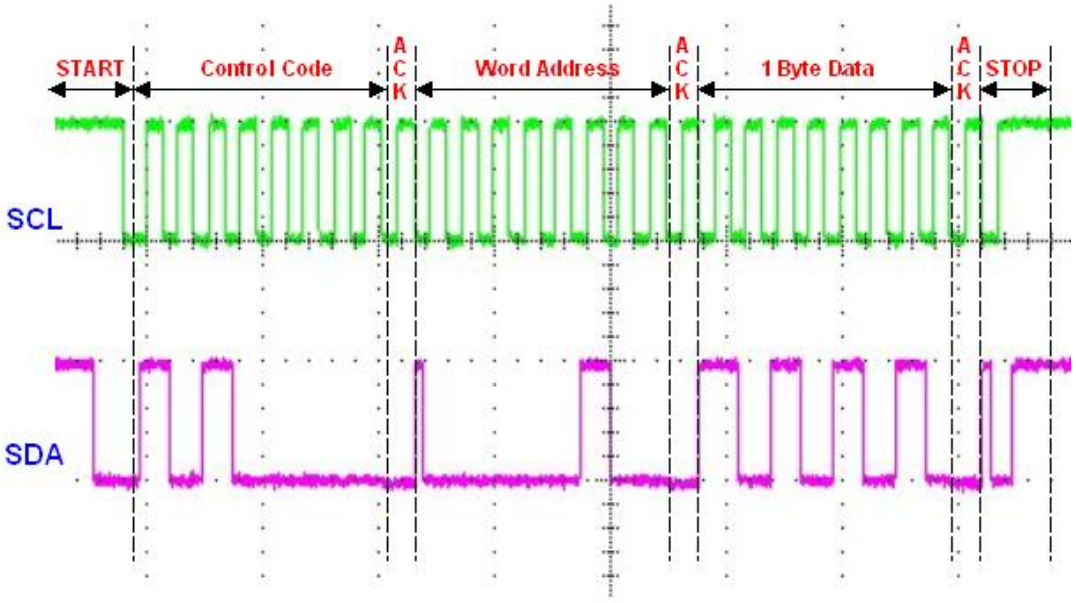
iv) read a byte data from EEPROM, and

v) a STOP condition.

4. Program Analysis

4.1 Byte Write

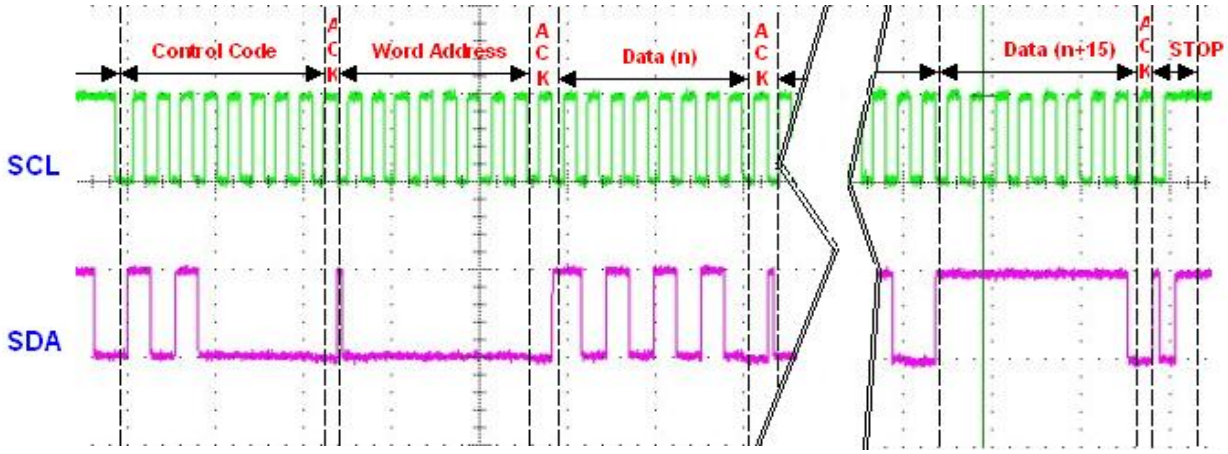
The following figure shows the operation of byte write: -



The average time for writing a byte is *16.0ms.

4.2 Page Write

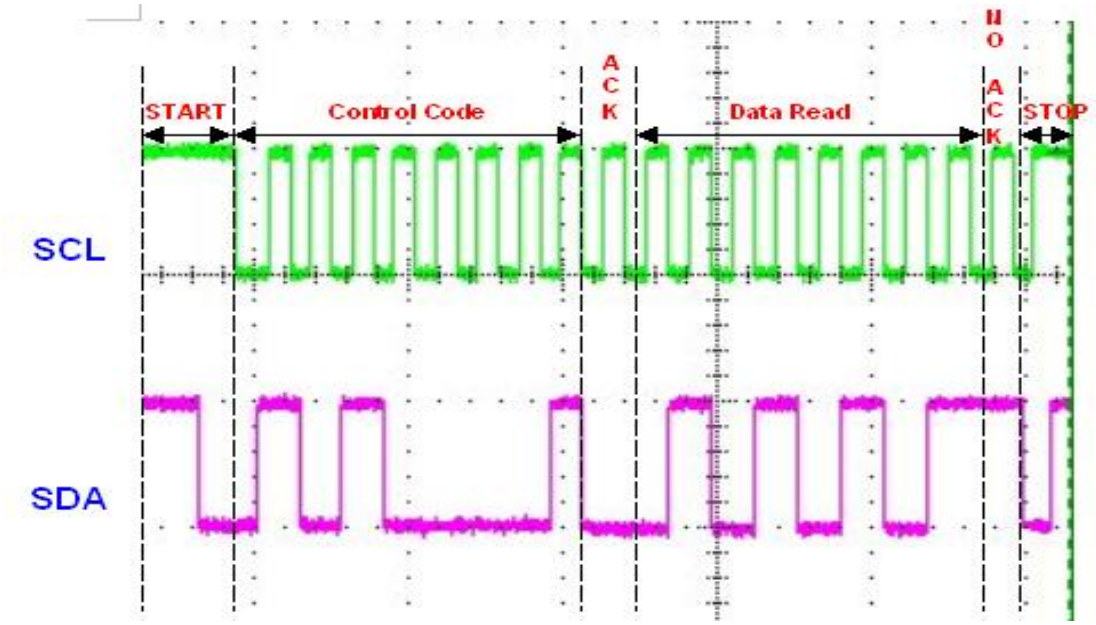
The following figure shows the operation of page write: -



The average time for writing a page of 16kbyte is *89.5ms.

4.3 Current Read

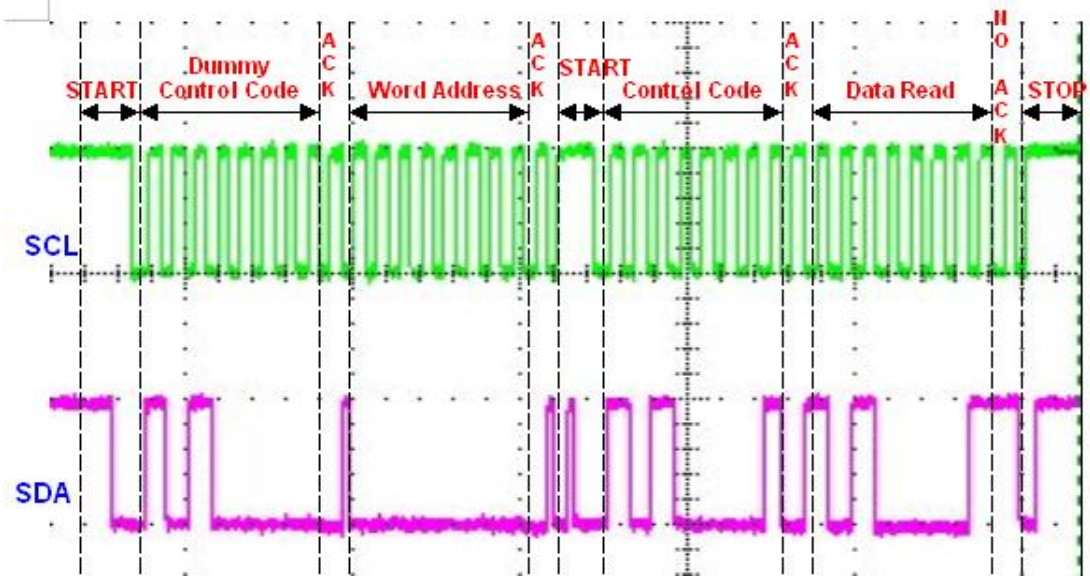
The following figure shows the operation of current read: -



The average time for reading a byte using current read operation is *11.2ms.

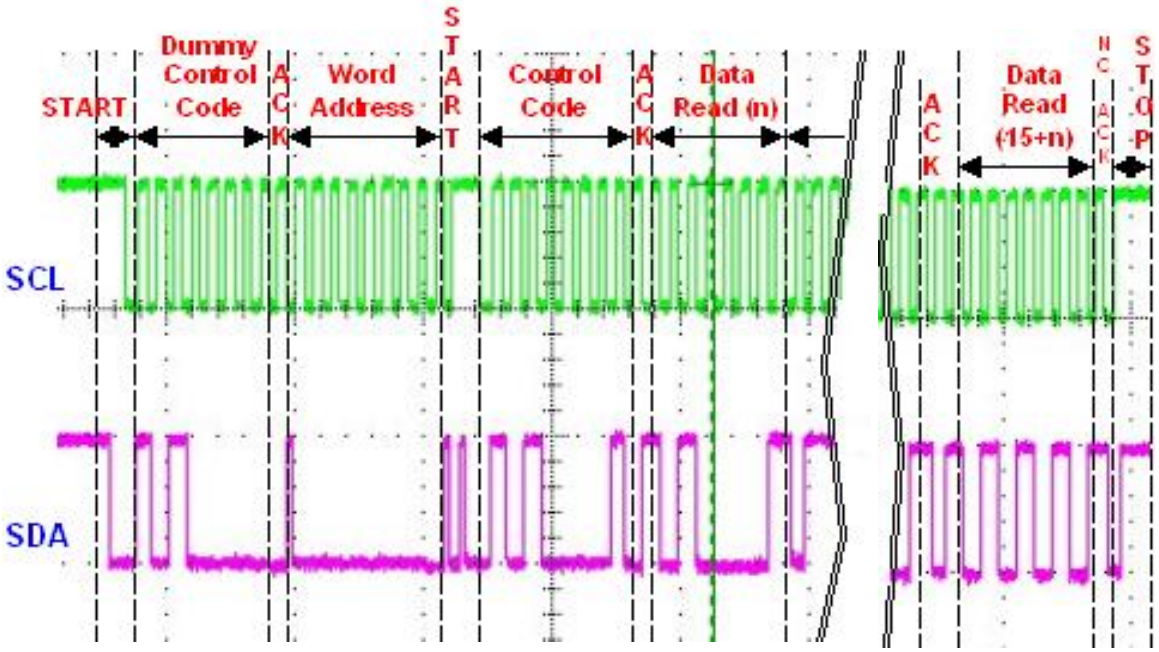
4.4 Random Read

The following figure shows the operation of random read: -



The average time for reading a byte of data using random read operation is *22.3ms.

4.5 Sequential Read



The average time for reading a page of 16kbytes using sequential read operation is *__97.8ms.__

Note:

All the average times given were measured using a: -

i) 10MHz clock, and

ii) *½ system clock divider.*

5. Sample Code

```
/************************************************************************/
/*                                                                      */
/*  FILE        :I2C.c                                                  */
/*  DATE        :Fri, Dec 27, 2002                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/38024F                                              */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/

#include "iodefine.h"
#include "I2C.h"
#include <stdio.h>

#define NODE_ADDR       0xa0

unsigned int i;

unsigned char buf[16]={0xaa, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                       0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff  };

void main(void)
{
  byte_write();

  for (i=0;i<1000;i++);               //delay approximately 500ms

  page_write();

  current_address_read();

  random_or_sequential_read();
}

void byte_write(void)
{
  //byte write
  i = I2cWrite(NODE_ADDR, buf, 1, 0x00);
     if (CheckWriteReady() ==1)
  i = I2cWrite(NODE_ADDR, buf, 2, 0x02);
     if (CheckWriteReady() ==1)
  i = I2cWrite(NODE_ADDR, buf, 1, 0x04);
}

void page_write(void)
{
  //page write
  i = I2cWrite(NODE_ADDR, buf, 16, 0x00);
}
```

```
void current_address_read(void)
{
  //current address read
  i = I2cCurrentRead(NODE_ADDR, buf);
}

void random_or_sequential_read(void)
{
  //random / sequential read
  i = I2cRead(NODE_ADDR, buf,16, 0x00);
}
```

```
/************************************************************************/
/*                                                                      */
/*  FILE        :RW.c                                                   */
/*  DATE        :Fri, Dec 27, 2002                                      */
/*  DESCRIPTION :Function Program                                       */
/*  CPU TYPE    :H8/38024F                                              */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/



#include "i2c.h"
#include "iodefine.h"

char CheckWriteReady(void);


/* Check SCL signal level */
unsigned char SclIn (void)
{
    SCL_IO_REG &= SCL_IO_RESET_BIT;             //It is Input
    if(SCL_DATA_REG & SCL_DATA_SET_BIT)     //check Port status
    {
      return(HIGH);
    }
    else
    {
      return(LOW);
    }
}

/* Check SDA signal level */
unsigned char SdaIn (void)
{
    SDA_IO_REG &= SDA_IO_RESET_BIT;             //It is Input
```

```
    if(SDA_DATA_REG & SDA_DATA_SET_BIT)     //check Port status
    {
       return(HIGH);
    }
    else
    {
      return(LOW);
    }
}

/* Drive SCL bus */
void SclOut (unsigned char status)
{
    if (status == LOW)
    {
   SCL_DATA_REG = 0;               //Drive Port LOW
   SCL_IO_REG |= SCL_IO_SET_BIT;   //Port is output

    }
   else
   {
       SCL_DATA_REG = 1;             //Port is Input & using external pull-up
                                     //resistor to go high
    SCL_IO_REG |= SCL_IO_SET_BIT; //Port is output
      }
}

/* Drive SDA bus */
void SdaOut (unsigned char status)
{
    if (status == LOW)
   {
     SDA_DATA_REG = 0;               //Drive Port LOW
        SDA_IO_REG |= SDA_IO_SET_BIT;   //Port is output
   }
   else
   {
        SDA_DATA_REG = 1;             //Port is Input & using external pull-up
                                      //resistor to go high
     SDA_IO_REG |= SDA_IO_SET_BIT; //Port is output
   }
}

void Delay(void)                      //Delay approximately 10ms
{
    unsigned char i;

    i=0;
    while(i<20)
    {
        i++;
    }
}
```

```
void Delay2x(void)                    //Delay approximately 20ms
{
    Delay();
    Delay();
}

/* All codes below here are independent with hardware, such as microprocessor,
I/O port */
unsigned char CheckBusState(void)
{
    if ((SclIn() == HIGH) && (SdaIn() == HIGH))
    {
      return(TRUE);
    }
    else
    {
      return(FALSE);
    }
}

/* It is nice to send out data at the middle of clock low */
void SendBit (unsigned char data_byte)
{
    SclOut(LOW);
    Delay();
    if (data_byte != 0)
    {
      SdaOut(HIGH);
    }
    else
    {
      SdaOut(LOW);
    }

    Delay();
    SclOut(HIGH);
    while (SclIn() != HIGH) {}  //wait for slow device to release clock
    Delay2x();
}

void SendStartBit(void)
{
    Delay();
    SdaOut(LOW);

    Delay2x();
    Delay2x();

    SclOut(LOW);
    Delay();
}
```

```
/* It is nice to sample data input at the middle of clock high */
unsigned char GetBit (void)
{
    unsigned char temp;

    SclOut(LOW);
    temp = SdaIn();
    Delay2x();

    SclOut(HIGH);
    while (SclIn() != HIGH) {}  //wait for slow device to release clock
    Delay();

    temp = SdaIn();
    Delay();

    return(temp);
}

/* Getting ACK is similar to GetBit, but it is a little tricky since master
must pull SDA high
before it find out whether there is ACK (SDA is low) or not */
unsigned char GetAck (void)
{
    unsigned char temp;

    SclOut(LOW);
    Delay();

    SdaOut(HIGH);
    temp = SdaIn();
    Delay();

    SclOut(HIGH);
    while (SclIn() != HIGH) {}  //wait for slow device to release clock
    Delay();

    temp = SdaIn();
    Delay();

    return(temp);
}

/* Note master need get ACK after sending out every byte */
unsigned char SendByte(unsigned char data_byte)
{
    unsigned char i;
    unsigned char mask;

    mask = 0x80;        //send out MSB first
    for (i=0; i<8; i++)
   {
        SendBit(data_byte & mask);
```

```
        mask /= 2;
      }

    return(GetAck());
}

unsigned char GetByte(void)
{
    unsigned char temp1, temp2;
    unsigned char i,mask;

    mask = 0x80;
    temp2 = 0;
    for (i=0; i<8; i++)
    {
        temp1 = GetBit() * mask;
        temp2 += temp1;
        mask /= 2;
    }

    return(temp2);
}

void SendStopBit(void)
{
    SclOut(LOW);
    Delay();

    SdaOut(LOW);
    Delay();

    SclOut(HIGH);
    Delay2x();

    SdaOut(HIGH);
}


unsigned char I2cWrite(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr) {
    unsigned int i;

    if( CheckBusState() != TRUE)
    {
     return(BUS_BUSY);
    }

    SendStartBit();

    if (SendByte((slave_addr) & 0xfe) != LOW) //Send address and write command
    {
     return(NO_RESPONSE);
    }
```

```
    if (SendByte(word_addr) != LOW)             //Send low word address
    {
    return(NO_RESPONSE);
    }


   for(i=0; i<length; i++)
   {
        if(SendByte(*buf_ptr++) != LOW)
       {
            return(ERR_RESPONSE);
       }
   }

    SendStopBit();

    return(OP_DONE);
}

unsigned char I2cRead(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr)
{
    unsigned char i=0,j=0;
    unsigned char DataBuffer[500];
    unsigned char LastData;

    if( CheckBusState() != TRUE)
   {
        return(BUS_BUSY);
      }

    SendStartBit();

    if (SendByte((slave_addr) & 0xfe) != LOW)  //Send address and read command
   {
        return(NO_RESPONSE);
      }

   if (SendByte(word_addr) != LOW)          //Send high word address
   {
     return(NO_RESPONSE);
      }

    SendStopBit();                              //Pull-up SDA line
    SendStartBit();

    if (SendByte((slave_addr) | 0x01) != LOW)  //Send address and read command
   {
        return(NO_RESPONSE);
      }

    for(i=0; i<length-1; i++)
```

```
{
      DataBuffer[i]= GetByte();
     SendBit(LOW);                    //ack it low
      }

    DataBuffer[i]= GetByte();               //get last data and ack high

    SendBit(HIGH);
    SendStopBit();

    return(OP_DONE);
}

/*Since Microchip devices such as 24AA16 will not acknowledge during a write
cycle, this can be used to determined when the cycle is complete. So that the
master can proceed with next operation*/
char CheckWriteReady(void)
{
   unsigned int i=0;

   while(i<4)
   {
       SendStartBit();

       if (SendByte((0xa0) | 0x00) == LOW)
       {
        SendStopBit();
        return (1);
        }

        SendStopBit();
     i++;
      }
}

unsigned char I2cCurrentRead(unsigned char slave_addr, unsigned char *buf_ptr)
{
    unsigned char i=0,j=0;
    unsigned char DataBuffer[500];
    unsigned char LastData;

   SendStartBit();

    if (SendByte((slave_addr) | 0x01) != LOW)  //Send address and read command
   {
        return(NO_RESPONSE);
      }

    *buf_ptr = GetByte();                      //get last data and ack high

    SendBit(HIGH);
    SendStopBit();
```

```
    return(OP_DONE);
}
```

```
/*****************************************************************/
/*                                                               */
/*  FILE        :i2c.h                                           */
/*  DATE        :Fri, Dec 27, 2002                               */
/*  DESCRIPTION :Definition of constant and functions            */
/*  CPU TYPE    :H8/38024F                                       */
/*                                                               */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).*/
/*                                                               */
/*****************************************************************/

//bit patterns for EEPROM access instructions
#define READSR 0xA0 /* (bit reversed 05) */
#define SETWEL 0x60 /* (bit reversed 06) */
#define WRITE 0x40  /* (bit reversed 02) */
#define READ 0xC0   /* (bit reversed 03) */

/* Set as 1010 binary for read and write operation (device dependent)*/
#define NODE_ADDR       0xa0

/* Misc */
#define HIGH    1
#define LOW     0

#if !defined(TRUE)
#define TRUE    1
#endif

#if !defined(FALSE)
#define FALSE   0
#endif

#define OP_DONE         0x00
#define BUS_BUSY        0x01
#define NO_RESPONSE     0x02
#define ERR_RESPONSE    0x04

// SDA and SCL port def.

/* control SDA port as input or output */
#define SDA_IO_REG              P_IO.PCR7.BYTE
#define SDA_IO_SET_BIT          0x01    //output
#define SDA_IO_RESET_BIT        0xfe    //input

/* check SDA port low or high */
#define SDA_DATA_REG            P_IO.PDR7.BYTE
#define SDA_DATA_SET_BIT        0x01
```

```
#define SDA_DATA_RESET_BIT      0xfe

/* control SCL port as input or output */
#define SCL_IO_REG              P_IO.PCR8.BYTE
#define SCL_IO_SET_BIT          0x01    //output
#define SCL_IO_RESET_BIT        0xfe    //input

/* check SCL port low or high */
#define SCL_DATA_REG            P_IO.PDR8.BYTE
#define SCL_DATA_SET_BIT        0x01
#define SCL_DATA_RESET_BIT      0xfe

//I2C modules
void byte_write(void);
void page_write(void);
void current_address_read(void);
void random_or_sequential_read(void);
void startbit(void);
void stopbit(void);
void sendbit(unsigned char );
void controlbyte(unsigned char );
void ReadBuffer(void);
extern char checkwrirdy();
unsigned char getACK(void);
unsigned char getbit(unsigned char);
unsigned char I2cWrite(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr);
unsigned char I2cRead(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr);
unsigned char I2cCurrentRead(unsigned char slave_addr, unsigned char
*buf_ptr);
```

Reference

1. *The I²C-Bus Specification (Version 2.1)*, January 2000, Philips Semiconductor.
2. *24AA16/ 24LC16B 16K I²C Serial EEPROM*, 2002, Microchip Technology Inc.
   http://www.microchip.com/download/lit/pline/memory/ic/21703b.pdf
3. Zhen Jiang, *I²C Interface With The Hitachi SH Microprocessor (Revision 1.1)*, 21 Jan 1998, Hitachi Micro System, Inc.
4. http://www.esacademy.com/faq/i2c/
5. *H8/38024 Series,H8/38024F-ZTZT Hardware Manual(version 2.0)*, 20 Feb 2002, Hitachi Ltd.
6. Leonard Haile, *Hitachi H8/3437 Series Microcontroller I²C Peripheral-A practical SMBus/I²C Firmware Design Guide (Revision 1.2)*, 12 June 1998, Hitachi Semiconductor (America) Inc.

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sep.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.