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.

SPI™ EEPROM access thru SCI Emulation

Introduction

The application note shows how the Serial Communication Interface (SCI3) in SLP series can be configured to support SPI and detail communication between a SLP and a SPI EEPROM in order to reduce the system designer's learning curve.

The Serial Peripheral Interface (SPI) provides a channel of full-duplex, synchronous, 8-bit serial communication between master and slave or peripheral devices. The SPI can be programmed from a host CPU using the Serial Communication Interface (SCI).

The Serial Communication Interface 3(SCI3) hardware on the H8/300L Super Low Power (SLP) series provides a simple three-wire connection to an SPI serial EEPROM.

Target Device

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

Contents

1. SPI™ Interface Overview

The SPI requires two control lines (CS and SCK_1) and two data lines (SI_1 and SO_1). With CS (Chip-Select) the corresponding peripheral device is selected. This pin is mostly active-low. In the unselected state the SO_1 lines are hi-Z and therefore inactive.

The master decides with which peripheral device it wants to communicate. The clock line SCK_1 is brought to the device whether it is selected or not. The clock serves as synchronization of the data communication.

(The general features of SPI is discussed in the Application Note on SPI and I^2C)

1.1 SPI EEPROM Control @

The sample code used M95640, a 64 Kbit Serial SPR Bus EEPROM with high speed clock from STMicroelectronics. M95640 implements a SPI protocol for serial EEPROM control.

The instruction set is shown below:

Instruction	Description	Instruction Format
WREN	Write Enable	0000 0110 (0x06)
WRDI	Write Disable	0000 0100 (0x40)
RDSR	Read Status Register	0000 0101 (0x05)
WRSR	Write Status Register	0000 0001 (0x01)
READ	Read from Memory Array	0000 0011 (0x03)
WRITE	Write to Memory Array	0000 0010 (0x02)

* *the sample code in this application note only uses four of the six commands:- WREN, RDSR, READ and WRITE*

The WREN and WRDI control commands are single byte commands. No extra data needs to be transmitted or returned.

The RDSR and WRSR control commands require a second byte transfer to complete the operation.

The READ and WRITE commands require multiple byte transfer. Both command have the following protocol:

<instruction> <address> <N- Byte data>

The <address> field is a 2-byte start address from which the data read/write will begin. The data may be then read/written sequentially with the source/destination address being incremented automatically by the EEPROM. Please note that for this device, the WREN is cleared automatically after each write operation has completed.

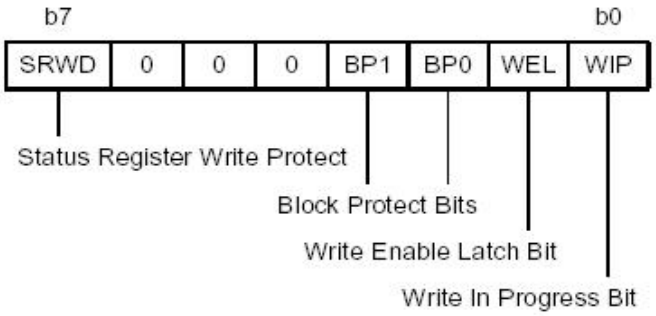
1.2 Bit Reverse

The SPI serial EEPROM communications protocol specifies that data will be transmitted starting with most significant bit (MSB) first. In synchronous mode, the SCI interface on SLP series shift data in and out starting with the least significant bit (LSB) first.

To overcome this, a look up table has been implemented to perform the swapping of the data to be transmitted over the SCI

1.3 Status Register @

The Status Register in ST M95640 contains a number of status and control bits that can be read or set (as appropriate) by specific instructions as following:



1. **WIP bit.** The Write In Progress (WIP) bit indicates whether the memory is busy with a Write or Write Status Register cycle.

2. **WEL bit.** The Write Enable Latch (WEL) bit indicates the status of the internal Write Enable Latch.

3. **BP1, BP0 bits.** The Block Protect (BP1, BP0) bits are non-volatile. They define the size of the area to be software protected against Write instructions.

4. **SRWD bit.** The Status Register Write Disable (SRWD) bit is operated in conjunction with the Write Protect (W) signal. The Status Register Write Disable (SRWD) bit and Write Protect (W) signal allow the device to be put in the Hardware Protected mode. In this mode, the non-volatile bits of the Status Register (SRWD, BP1, BP0) become read-only bits.

1.4 SPI Implementation @

The SPI interface was designed noting the following points:

- The WREN must be set for each write operation.
- The CS line cannot go high until all bytes have been successfully read/written from/to the EEPROM.
- The operation is only completed when the CS line goes high
- Write operation may take up to 10ms(worst case) internally in the EEPROM after the write data transfer has completed. Thus the EEPROM may appear “busy” for some time after a write transfer has completed.
- The “busy” state is denoted by the Status Register being read as 0xFF.
- The format of the read/write instructions is basically the same, only the data line changes. Thus a similar coding methodology could be used for both types of operation.
- When the SI line is being driven by the controller, the SO line remain high.
- When the SO line is being driven by the EEPROM, the SI line not sampled by the EEPROM.

Because there is no official specification, what exactly SPI is and what not, it is necessary to consult the data sheets of the components one wants to use.

1.5 CS Line Control and Implementation

CS line must be carefully controlled to ensure correct operation since the application must be able to determine the precise point at which this should occur. When using the synchronous serial port of SLP, there are two methods which maybe used:-

i) The Transmit End Interrupt Enable (TEIE) maybe used to signal to the CPU that the last bit has been shifted out of the Transmit Shift Register (TSR) and that there is no data waiting to be transmitted in the Transmit Data Register (TDR).

ii) If the receiver is enabled, data is received in synchronous with transmission, as they both use the SCK line, thus the application code can determine when a byte has been transmitted by the fact that one has also been received.

As the majority of the SPI operations are two-way, the second option is most desirable, as this will automatically give both two-way data exchange and CS line control with minimal code/data overhead. The TEIE based solution would require an extra interrupt handler.

1.6 Read/Write Control and Implementation

To read data, the relevant instruction and address data are transmitted to the EEPROM, and the required data read in- this is achieved by transmitting “dummy” data bytes until the required amount has been read in. The dummy data maybe random or initialized to 0xFF (as in this sample code). A receive interrupt handler is used to read data and control the CS line. A count is maintained of the number of bytes which are expected to be received, when this count has expired the CS line is taken high and further receive interrupt disabled.

To write data, the relevant instruction and address data are transmitted, followed by the write data. Once again the receive interrupt handler decrements a count and controls the CS lone when the required number of bytes has been written, in this instance the received data is irrelevant (and will in fact be all 0xFF values).

1.7 Connecting to the SPI Bus @

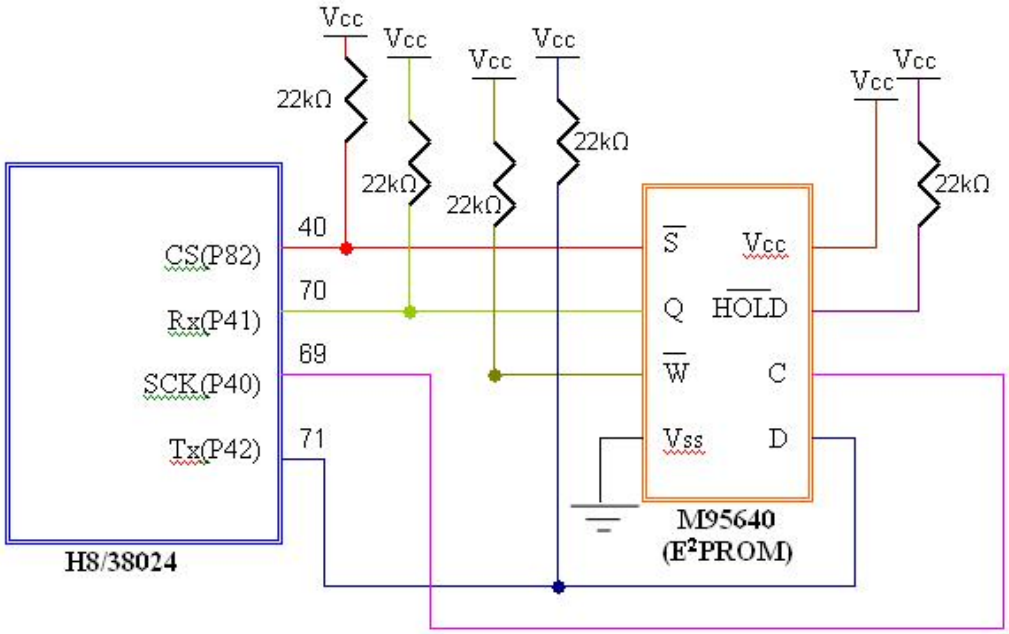
ST M95640 is fully compatible with the SPI protocol. All instructions, addresses and input data bytes are shifted in to the device, most significant bit first. The Serial Data Input (D) is sampled on the first rising edge of the Serial Clock (C) after Chip Select (S) goes Low.

All output data bytes are shifted out of the device, most significant bit first. The Serial Data Output (Q) is latched on the first falling edge of the Serial Clock (C) after the instruction (such as the Read from Memory Array and Read Status Register instructions) have been clocked into the device.

@ Note:

The explanations given in this section is based on ST M95640 and is device dependent. It might be different from the EEPROM user is using. Please consult the respective data sheet and specification for clarification.

2. Hardware Design



*Signal Names

S	*Chip Select*	*Vcc*	*Supply Voltage*
Q	*Serial Data Output*	*HOLD*	*Hold*
W	*Write Protected*	*C*	*Serial Clock*
Vss	*Ground*	*D*	*Serial Data Input*

* *The Write Protected ($\overline{W}$) and Hold ($\overline{HOLD}$) signals should be driven, High or Low as appropriate.*

3. Function Overview

Functions in `spi.c`:

- `void SPISetup (void)`
- `uchar ReadSR (void)`
- `void SetWREN (void)`
- `void Write (ushort, ushort, uchar *)`
- `void WriteByte (ushort, uchar)`
- `void Read (ushort, ushort, uchar *)`
- `void main (void)`

Function in `intprg.c (__interrupt(vect=18)`:

- `void INT_SCI3(void)`

`void SPISetup (void)`

This routine initializes the entire initialization for:

i) Serial Communication Interface (SCI)

a) baud rate(BRR),

b) data transfer format(SMR),

c) serial control setting(SCR), and

d) serial status(SSR)

**please refer to the Hardware Manual for details about SCI setting*

ii) EEPROM by setting the CS line (PDR8).

`uchar ReadSR (void)`

The Read Status Register (RDSR) instruction allows the Status Register to be read. CS line must be set to low to initialize EEPROM and set to high after completed reading the register.

`void SetWREN (void)`

This routine must be called before it can proceed to any write operation. The routine will issue the set write enable instruction (to set the WREN). After the write operation has completed and the CS line has gone high, the EEPROM will automatically resets the WREN bit, so this function must be called before all write operation.

`void WriteByte (ushort, uchar)`

```
Parameters:
address to write data to (ushort),
data to be written (uchar)
```

The WriteByte routine issues a write command, followed by the address passed as a parameter. Then the data parameter is written into the EEPROM. This allows only a single byte to be written. The RX interrupt is used to set the CS line on completion.

`void Write (ushort, ushort, uchar *)`

```
Parameters:
address to write data to (ushort)
number of data bytes to be written (uchar)
address of buffer containing data (uchar *)
```

The Write function transmits the WRITE instruction, plus the bit reversed of two byte address to the EEPROM, then uses a polled loop to transmit the rest of the data from the buffer passes as parameter 3 (*DataPtr).

void Read (ushort, ushort, uchar *)

```
Parameters:
address to read data from (ushort)
number of data bytes to be read (uchar)
address of buffer for data store (uchar *)
```

The Read function transmits the Read instruction, plus the bit reversed two byte address to the EEPROM, then uses a polled loop to read the rest of the data. The receive interrupt is used to keep count of the transmitted bytes and then to take the CS line high on completion, in addition to store the received data in the buffer defined by parameter 3 (*Buffer).

`void main()`

The main function demonstrates/tests the usage of the EEPROM control and communication functions by writing a single byte to the address before reading back the data from EEPROM.

Then the function will write a series of data into EEPROM and read back the data from EEPROM.

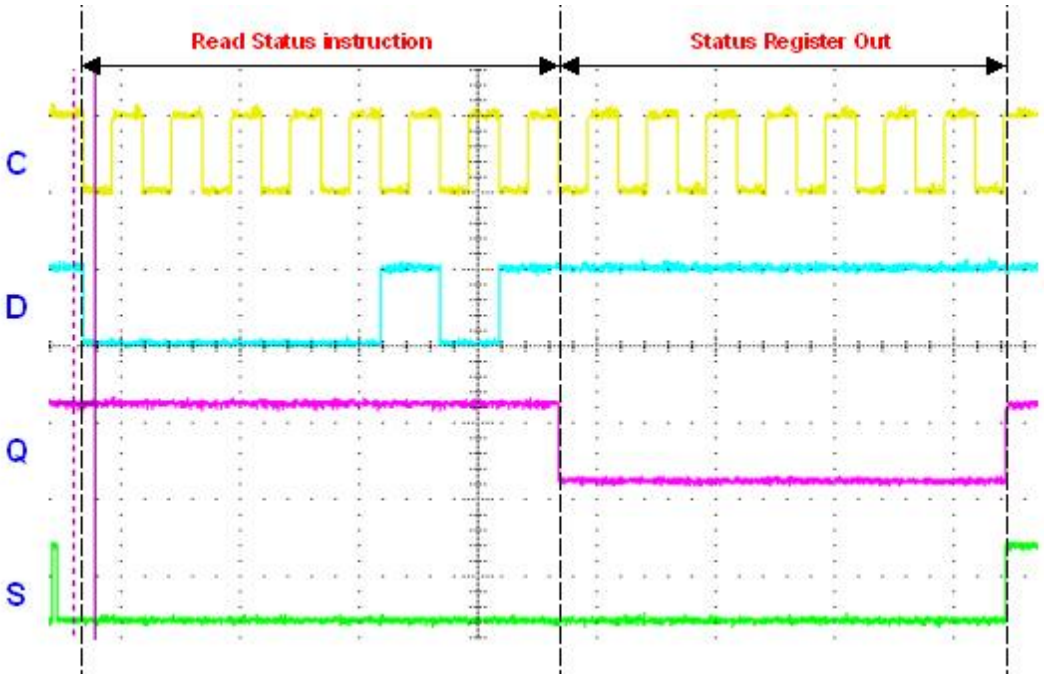
`void INT_SCI3(void)`

The Receive Interrupt is to be used to monitor the number of bytes, which have been shifted out from the TSR register (completed the transmission). The exact time when the full read cycle has completed can be determined when the CS line taken high. The received data is always placed into the RxBuffer, as the overhead of saving unwanted data is minimal, and this will also removed the need for an extra test. This function should be inserted in the interrupt program, vector 18.

4. Program Analysis

4.1 READ Status

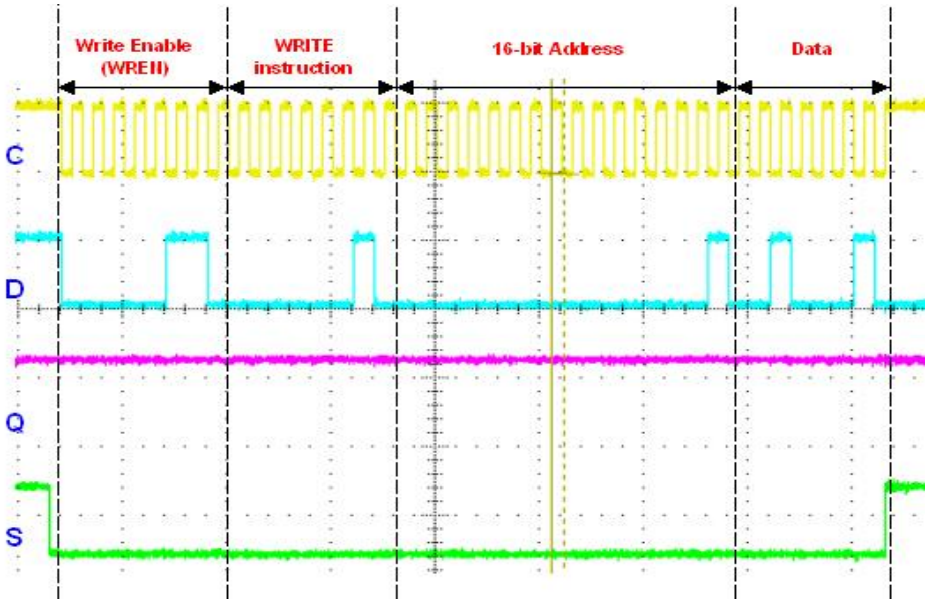
The following figure shows the operation of Read Status: -



The average time for reading the status register is *0.34ms.

4.2 Byte WRITE

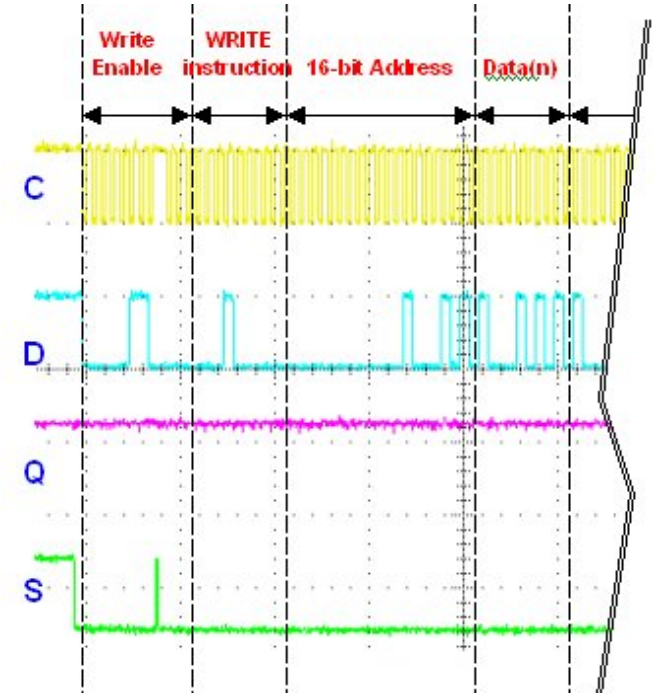
The following figure shows the operation of writing a byte of data: -



The average time for writing a byte of data is *0.85ms.

4.3 Page WRITE

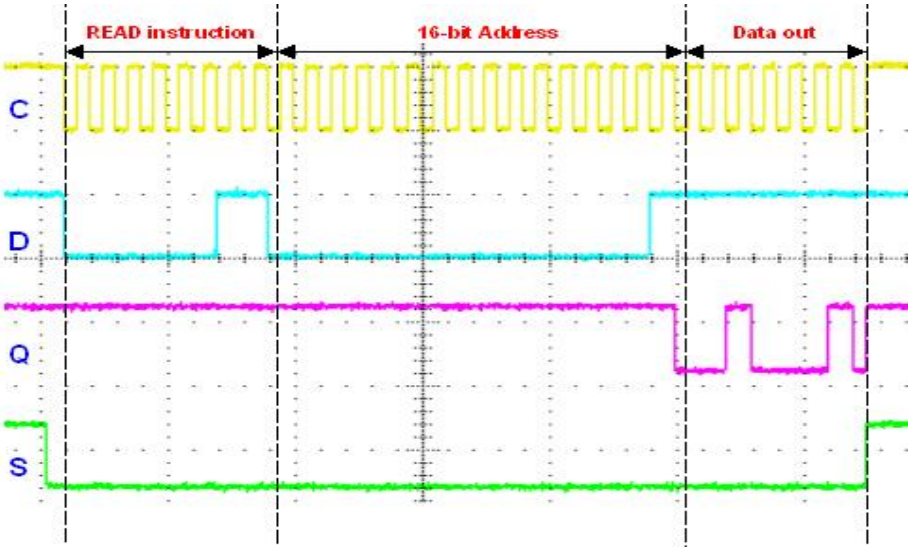
The following figure shows the operation of writing a page of data: -



The average time for writing a page of 32 bytes of data is *5.5ms.

4.4 Byte READ

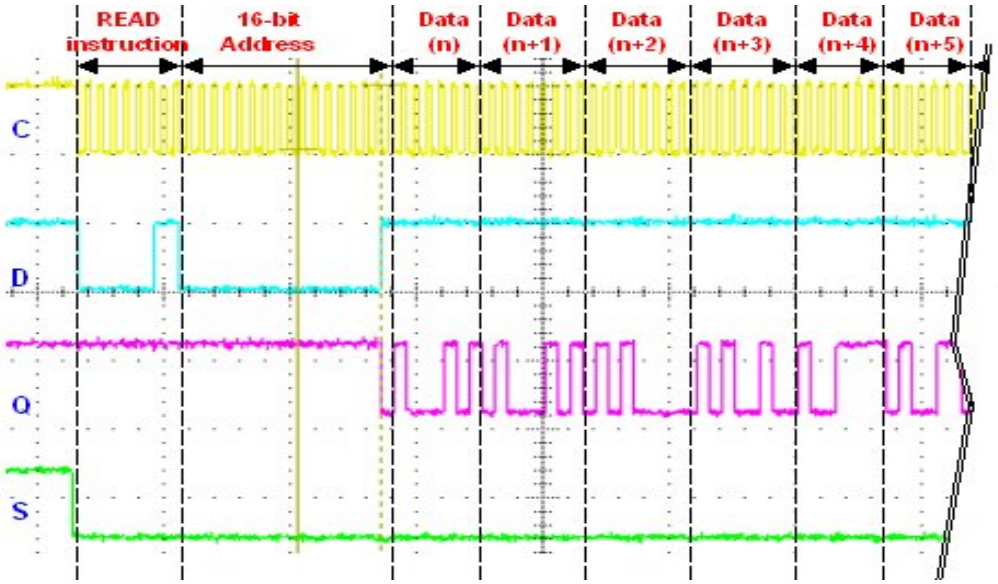
The following figure shows the operation of reading a byte of data: -



The average time for reading a byte of data is ***0.53ms.**

4.5 Page READ

The following figure shows the operation of reading a page of 32 bytes data: -



The average time for reading a page of data is *__4.6ms.__

Note:

*All the average times given were measured using a **10MHz clock** and ½ **system clock divider**.*

5. Sample Code

```
/************************************************************************/
/*                                                                      */
/*  FILE        :SPI.c                                                  */
/*  DATE        :Fri, Dec 20, 2002                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/38024F                                              */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/
#include "iodefine.h"
#include "applicationdemo.h"
#include <machine.h>
#include <stdio.h>


//Function Prototypes
void moreByte_access(void);
void byte_access(void);
void delay(void);
void SPISetup (void);
uchar ReadSR (void);
void SetWREN (void);
void Write (ushort, ushort, uchar *);
void WriteByte (ushort, uchar);
void Read (ushort, ushort, uchar *);


//Global Data
ushort RxCount;             //No. of bytes to be received
ushort TxCount;             //No. of bytes to be transmitted
uchar TxBuffer[MAXRXCOUNT]; //Buffer used to store transmit data
uchar *RxBuffer_ptr;        //buffer for received data
uchar *TxBuffer_ptr;        //pointer to data for transmission
uchar RxBuffer[MAXTXCOUNT];
int i=0;

const char table[256] = {
0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0,
0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0,
0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8,
0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8,
0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4,
0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4,
0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC,
0x1C,0x9C,0x5C,0xDC,0x3C,0xBC,0x7C,0xFC,
0x02,0x82,0x42,0xC2,0x22,0xA2,0x62,0xE2,
0x12,0x92,0x52,0xD2,0x32,0xB2,0x72,0xF2,
0x0A,0x8A,0x4A,0xCA,0x2A,0xAA,0x6A,0xEA,
0x1A,0x9A,0x5A,0xDA,0x3A,0xBA,0x7A,0xFA,
0x06,0x86,0x46,0xC6,0x26,0xA6,0x66,0xE6,
```

```
0x16,0x96,0x56,0xD6,0x36,0xB6,0x76,0xF6,
0x0E,0x8E,0x4E,0xCE,0x2E,0xAE,0x6E,0xEE,
0x1E,0x9E,0x5E,0xDE,0x3E,0xBE,0x7E,0xFE,
0x01,0x81,0x41,0xC1,0x21,0xA1,0x61,0xE1,
0x11,0x91,0x51,0xD1,0x31,0xB1,0x71,0xF1,
0x09,0x89,0x49,0xC9,0x29,0xA9,0x69,0xE9,
0x19,0x99,0x59,0xD9,0x39,0xB9,0x79,0xF9,
0x05,0x85,0x45,0xC5,0x25,0xA5,0x65,0xE5,
0x15,0x95,0x55,0xD5,0x35,0xB5,0x75,0xF5,
0x0D,0x8D,0x4D,0xCD,0x2D,0xAD,0x6D,0xED,
0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD,
0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3,
0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3,
0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB,
0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB,
0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7,
0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7,
0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF,
0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF
};

uchar message[] = "EEPROM SPI Access ";


int main (void)
{
  SPISetup();

  byte_access();

  moreByte_access();

  while(1);

return (0);
}


void byte_access(void)
{
while(ReadSR() == 0xFF);     //check that we are talking
WriteByte(0x0001,0x33);      //try writing a byte

while(ReadSR() == 0xFF);     //check that we are talking
while(ReadSR()&01 == 0x01);  //wait until write completed(WIP=0)
Read(0x0001,1,&RxBuffer[0]); //read it back
}


void moreByte_access(void)
{
while(ReadSR() == 0xFF);      //check that we are talking
Write (0x0000,15,&message[0]);//write a 15 byte message
```

```
while (ReadSR() == 0xFF);     //wait on the internal operation
while(ReadSR()&01 == 0x01);   //wait until write completed(WIP=0)
Read(0x0000,15,&RxBuffer[0]); //read back the message
}


void SPISetup (void)
{
int i;

//the Null buffer could be a constant table to save RAM

for (i=0;i<MAXRXCOUNT;i++)
TxBuffer[i] = 0xFF;

P_SCI3.BRR = 249;         //50K bps
P_SCI3.SMR.BYTE = 0x80;  //sync mode; 8 bits
P_SCI3.SSR.BYTE &=0x87;  //clear error flags
P_SPCR.BYTE = 0xE0;       //select pin P42/TXD is used as TXD
P_SCI3.SCR3.BYTE = 0x30; //enb tx, rx, SCK out

P_IO.PCR8 = 0x04;        //set as output
P_IO.PDR8.BYTE |= 0x04;  //set CS line high
delay();
P_IO.PDR8.BYTE &= 0xFB;  //set CS line low to initialise EEPROM
delay();
P_IO.PDR8.BYTE |= 0x04;  //set CS line high
}


void delay(void)         //delay time approximately 42µs
{
   for(i=0;i<10;i++)
   ;
 }


//Returns: SR bits as unsigned char
uchar ReadSR (void)
{
unsigned char dummy;

P_SCI3.SSR.BYTE &= 0xBF;                        //clear RDRF
P_IO.PDR8.BYTE &= 0xFB;                    //set CS line low to initialise
EEPROM

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until ready to Tx
P_SCI3.TDR = READSR;                        //send readSR instruction (05)

while ((P_SCI3.SSR.BYTE & 0x40) != 0x40); //wait to get data0
P_SCI3.SSR.BYTE &= 0xBF;                   //clear RDRF
dummy=P_SCI3.RDR;                           //do dummy read
```

```
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = 0xff;                        //send dummy byte to receive data


while ((P_SCI3.SSR.BYTE & 0x40) != 0x40); //wait to get data1
P_SCI3.SSR.BYTE &= 0xBF;                 //clear RDRF

P_IO.PDR8.BYTE |= 0x04;                   //set CS line high

return(P_SCI3.RDR);
}


void SetWREN (void)
{
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
P_SCI3.SCR3.BYTE &= 0x20;                //disable RE
P_IO.PDR8.BYTE &= 0xFB;                   //set CS line low to init. EEPROM

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until ready to TX instruction
P_SCI3.TDR = SETWREN;                     //send setWREN instruction (06)

while ((P_SCI3.SSR.BYTE & 0x04) != 0x04); //wait to finish TX (denoted by RX)
P_SCI3.SCR3.BYTE = 0x30;                  //enable TE and RE
P_IO.PDR8.BYTE |= 0x04;                   //set CS line high to complete
}


void WriteByte (ushort Adrs, uchar Data)
{
unsigned char dummycount;

SetWREN();                                //set write enable flag
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
P_SCI3.SCR3.BYTE = 0x20;                  //disable RE
P_IO.PDR8.BYTE &= 0xFB;                   //set CS line low to start operation

//First write the instruction
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until ready to tx
P_SCI3.TDR = WRITE;                       //write ins (02)

//Then send the address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = swap((uchar)Adrs>>8);        //MSB of address

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = swap((uchar)Adrs);           //LSB of address

//Finally write data byte
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = swap(Data);                  //write data byte
```

```
while ((P_SCI3.SSR.BYTE & 0x04) != 0x04); //wait until TEND
P_SCI3.SCR3.BYTE = 0x30;                   //enable RE
P_IO.PDR8.BYTE |= 0x04;                    //set CS line high to complete
}


void Write (ushort Adrs, ushort Count, uchar *DataPtr)
{
SetWREN();                                 //set write enable flag
P_SCI3.SSR.BYTE &= 0xBF;                   //clear RDRF
P_SCI3.SCR3.BYTE &= 0x20;                  //disable RE
P_IO.PDR8.BYTE &= 0xFB;                    //reset CS line to start operation

//issue the Write Instruction
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = WRITE;                        //Write ins (02)

//send MSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs>>8);

//send LSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs);

//now set up for interrupt transfer
TxBuffer_ptr = DataPtr; //Set transmit buffer pointer

//the following is for polled
for (i=0;i<Count;i++)
{
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = swap(*TxBuffer_ptr);
TxBuffer_ptr++;
}

while ((P_SCI3.SSR.BYTE & 0x04) != 0x04); //wait until TEND
P_SCI3.SCR3.BYTE = 0x30; //enable RE
P_IO.PDR8.BYTE |= 0x04; //set CS line high to complete
}



void Read (ushort Adrs, ushort Count, uchar *Buffer)
{
int i;

RxCount = Count;          //total bytes to be transferred
RxBuffer_ptr = Buffer;    //set pointer to receive buffer
P_SCI3.SCR3.BYTE = 0x20;  //disable RE
P_SCI3.SSR.BYTE &= 0xBF;  //clear RDRF
P_IO.PDR8.BYTE &= 0xFB;   //reset CS line to start operation
```

```
//issue the Read Instruction
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = READ;                        //send Read instruction (03)

//send MSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80);  //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs>>8);

//send LSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80);  //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs);

while ((P_SCI3.SSR.BYTE & 0x04) != 0x04);  //wait until TX end
P_SCI3.SCR3.BYTE = 0x70;                    //enable RE, TE and RE Interrupts

//now set up for interrupt transfer
TxBuffer_ptr = TxBuffer; //transmit null data

//the following is for polled
for (i=0;i<Count;i++)
{
  while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
    P_SCI3.TDR = *TxBuffer_ptr;             //swap not required for null data
    TxBuffer_ptr++;
}

while(RxCount);
P_SCI3.SCR3.BYTE = 0x30;      //enable RE, TE and disable RE Interrupts
P_IO.PDR8.BYTE |= 0x04;       //set CS line high
}
```

```
/*************************************************************/
/*  FILE        :applicationdemo.h                           */
/*  DATE        :Fri, Dec 20, 2002                           */
/*  DESCRIPTION :Definition of variable                      */
/*  CPU TYPE    :H8/38024F                                   */
/*************************************************************/

//Type definitions
typedef unsigned char uchar;
typedef unsigned short ushort;

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

//system constants
#define MAXRXCOUNT 35
#define MAXTXCOUNT 35            /* 32 bytes of data plus ins & address */

#define swap(x) (table[x])       /* swap macro */
```

Reference

1. http://www.mct.net/faq/spi.html

2. Leonard Haile ,*Interfacing the H8/3644 to a Serial E²PROM.-How to use the SCI Interface to emulate an SPI interface(Revision 1.0*), 8/3/98,Hitachi Semiconductor (America) Inc.

3. M95640/M95320-64/32 Kbit Serial SPI Bus EEPROM With High Speed Clock, 2002, STMicroelectronics. *http://us.st.com/stonline/books/pdf/docs/5711.pdf*

4. *H8/38024 Series,H8/38024F-ZTZT Hardware Manual(version 2.0)*, 20 Feb 2002, Renesas Ltd.

Revision Record

Rev.	Date	Description	
		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.