Lab 7: Serial Communication(SPI and I2C)

Matthew Friedman 861151348
Souradeep Bhattacharya 861105938
EE128 Section: 021

November 06, 2017

Abstract

The objective of this lab was to get familiar with SPI and I2C programming. Both of these techniques allow us to add additional functionality to our microcontroller.

Experimental System Specification

Part 1: SPI based I/O Extender MCP23S08

In this lab experiment, we will obtain 3-bit information from a switch and display this info by using the lower 3 LEDs. Then we will change the order in which the LEDs light up relative to the switch. Then finally we will switch the location of the switch and LEDs on the I/O expander.

Part 2: I2C-based EEPROM Interfacing and Programming

In this lab experiment we will write a two 8 byte pages to the EEPROM and then read them back to make sure it was written correctly.

Block Diagram

Part 1: SPI based I/O Extender MCP23S08

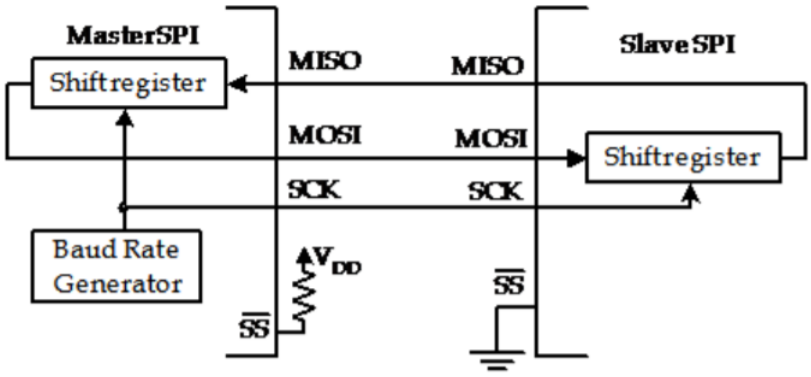

Figure 1: SPI Shift Register Diagram

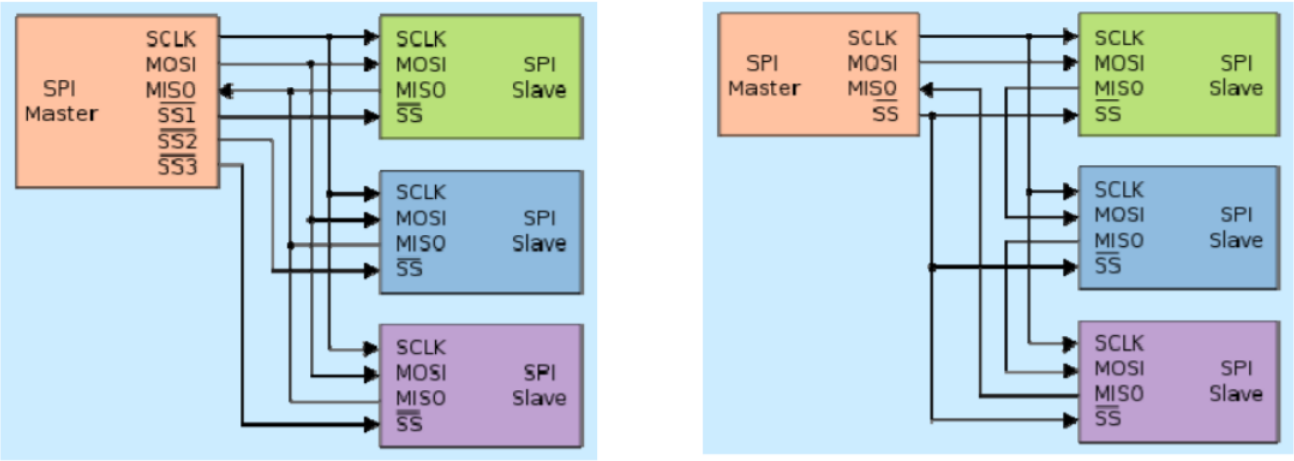

Figure 2: SPI Block Diagram

Part 2: I2C-based EEPROM Interfacing and Programming

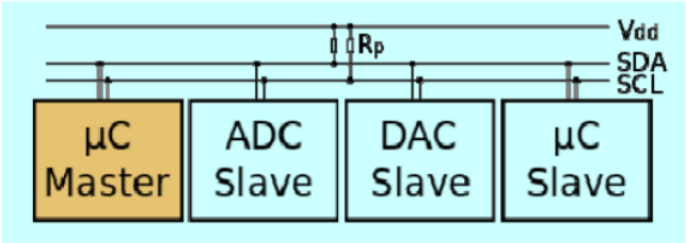

Figure 3: SPI Block Diagram

Detailed Schematic

Part 1: SPI based I/O Extender MCP23S08

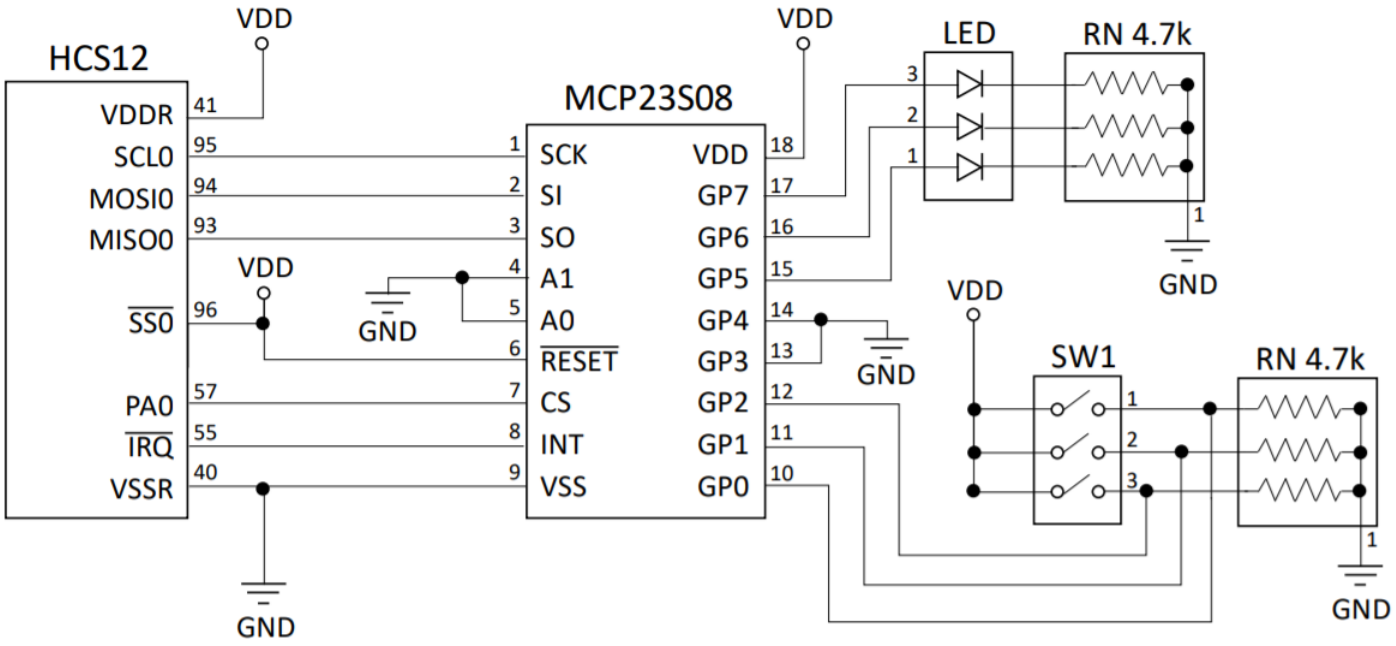

Figure 4: SPI Detailed Diagram

Part 2: I2C-based EEPROM Interfacing and Programming

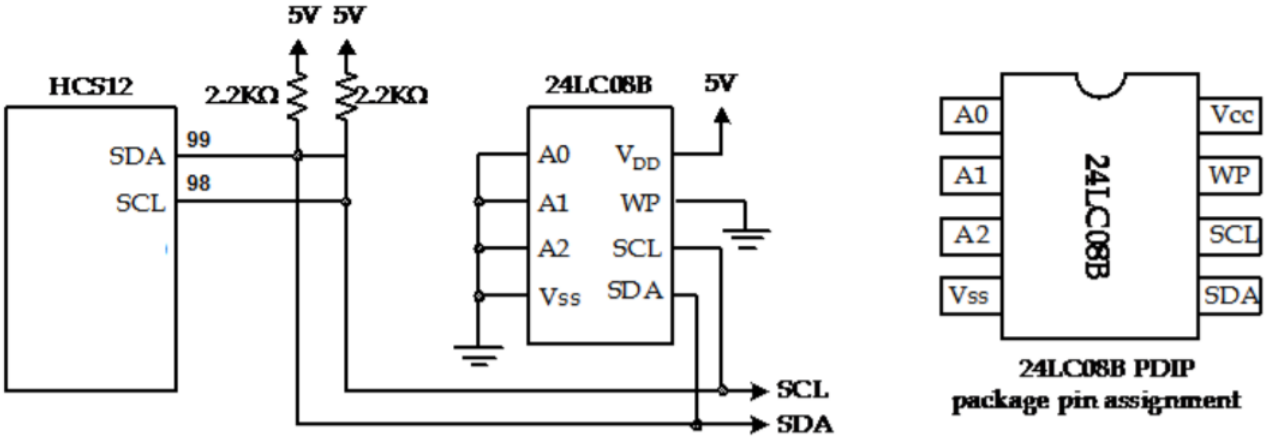

Figure 5: I2C Detailed Diagram

High Level Description of Software

Part 1: SPI based I/O Extender MCP23S08

This device first had to be configured before it could be used properly. We modified the template give to us to enabled it to to what we wanted it to.

Part 2: I2C-based EEPROM Interfacing and Programming

This particular EEPROM had a default page size of 8-bytes and supported page-writing. The command was exactly the same as the normal write command except you do not send the stop condition until after 8 pages had been written. It would handle the auto incrementing of the addresses. We simply did this twice to write the two pages.

Program Listing

Part 1: SPI based I/O Extender MCP23S08

```
1   /* ****************************************************************** *
2   * Project:    SPI_IO_Expander                                        *
3   * Purpose:    EE128 Lab 7, SPI and I2C Communication                 *
4   *                                                                    *
5   * Notes:      1. See Lab 7 manual for a detailed testbench schematic *
6   *             2. Switches are supposed to be initially ALL OFF *
7   * ****************************************************************** */
8
9   #include <hidef.h>      /* common defines and macros */
10  #include <mc9s12dg256.h> /* derivative information */
11  #include "spi.h"        /* MC9S12 SPI Library */
12  #include "mcp23s08.h"   /* I/O Expander Registers and Bits */
13
14  #define INPORT    0x00    /* INPUT Port */
15  #define OUTPORT   0xFF    /* OUTPUT Port */
16
17  char temp = 0x00;
18
19  extern void putcspi0(char);
20  extern void putsspi0(char*);
21  extern char getcspi0(void);
22  extern void getsspi0(char*, char);
23
24  void        SPI_SetIOXregister(unsigned char,unsigned char);
25  unsigned char SPI_GetIOXregister(unsigned char);
26
27  __interrupt void IRQISR(void);
28
29  static unsigned char regByte; /* temporal variable for register data */
30
31  #define IOX_WR_OP 0x40 /* R/W bit = 0, address is always hardware 00 (grounded) */
32  #define IOX_RD_OP 0x41 /* R/W bit = 1, address is always hardware 00 (grounded) */
33
34  /* ****************************************************************** *
35  *                                                                    *
36  *                           MAIN                                     *
37  *                                                                    *
38  * ****************************************************************** */
39
```

```
40  void main(void) {
41
42  /* ****************************************************************** *
43  *                                                                    *
44  *                        SETUP                                       *
45  *                                                                    *
46  * ****************************************************************** */
47
48      DDRA = OUTPORT; /* control \CS pin of IOX chip */
49      PORTA = 0x01;  /* initially deselect IOX chip */
50
51  /*** Setup MC9S12 Interrupt Service ***/
52
53      PORTE = 0x02;  /* IRQ PIN PE1 PULL HIGH */
54      INTCR = 0xC0;  /* enable IRQ interrupt on falling edge */
55      asm("cli");    /* enable interrupt globally */
56
57  /*** Setup MC9S12 SPI Module ***/
58
59      SPI0BR = 0x77; /* set baudrate to the minimum possible 11.719 kHz*/
60      SPI0CR1 = 0x50; /* enable SPI, master mode, disable interrupt, SCK idle low,
61  data shift on SCK's rising edge, CPHA=0 */
62      SPI0CR2 = 0x02; /* disable bidirectional mode, SPI stops in wait mode */
63      WOMS   = 0;    /* enable Port S pull-up; otherwise use external resistors for pull-up */
64
65  /*** Setup MCP23S08 I/O Expander Registers ***/
66
67      regByte = 0x78; /* GP7..GP5 - output (LED's); GP2..GP0 -  input (switches) */
68      SPI_SetIOXregister(IOX_IODIR,regByte);
69
70      regByte = 0xF0; /* enable interrupt-on-change for GP2..GP0 */
71      SPI_SetIOXregister(IOX_GPINTEN,regByte);
72
73  /*** Setup Initial Output in I/O output pins ***/
74
75      regByte = 0x00; /* set zero initial output values */
76      SPI_SetIOXregister(IOX_GPIO,regByte);
77
78  /* ****************************************************************** *
79  *                                                                    *
80  *                        LOOP                                        *
81  *                                                                    *
82  * ****************************************************************** */
83
84      for(;;){} // wait for interrupts caused by switch changes
85
86  }  /* main */
87
88  /* ****************************************************************** *
89  *                                                                    *
90  *                   INTERRUPT ROUTINES                               *
91  *                                                                    *
92  * ****************************************************************** */
93
94  __interrupt void IRQISR(void)
95  /* interrupt routine to set new LED/Oscilloscope outputs */
96  {
97      regByte = SPI_GetIOXregister(IOX_GPIO);
98      regByte >>= 4;
```

```
99          temp = regByte;
100         //regByte = ((temp & 0x80) >> 7) | ((temp & 0x40) >> 5) | ((temp & 0x20) >> 3);
101         SPI_SetIOXregister(IOX_GPIO,regByte);
102     }
103
104     /* ********************************************************************* *
105     *                                                                       *
106     *                 AUXILIARY SPI ROUTINES                                *
107     *                                                                       *
108     * ********************************************************************* */
109
110     void SPI_SetIOXregister(unsigned char ioxAddress,
111     unsigned char regValue ) {
112         /* Write-Op to set registers,
113     and using PTA0 pin of PORTA to control CS pin of IOX */
114
115         PORTA = 0x00;       /* chip select, using PORTA */
116         putcspi0(IOX_WR_OP); /* schedule a writing op to a register */
117         putcspi0(ioxAddress); /* write to the address of IODIR register */
118         putcspi0(regValue); /* write to IODIR register */
119         PORTA = 0x01;       /* chip diselect */
120
121     }
122
123     unsigned char SPI_GetIOXregister(unsigned char ioxAddress) {
124         /* Read-Op to read registers,
125     and using PTA0 pin of PORTA to control CS pin of IOX */
126
127         unsigned char regValue;
128
129         PORTA = 0x00;       /* chip select using PORTA */
130         putcspi0(IOX_RD_OP); /* schedule a writing op to a register */
131         putcspi0(ioxAddress); /* write to the address of IODIR  register */
132         regValue = getcspi0(); /* write to IODIR register */
133         PORTA = 0x01;       /* chip diselect */
134
135         return regValue;
136     }
```

Part 2: I2C-based EEPROM Interfacing and Programming

I2C Header

```
1   /* ********************************************** *
2   *       I2C Library for EEPROM 24LC01B      *
3   * ********************************************** */
4
5   #include <hidef.h>
6   #include <stdio.h>
7   #include <mc9s12dg256.h>
8   #include "i2c.h"
9
10  //#define DBG_ON
11
12  /* OpenI2C */
13  void Init_I2C (char ibc, char I2C_ID)
14  {
```

```
15      #ifdef DBG_ON
16          printf("9S12: Configure I2C\r\n");
17      #endif
18          IBCR |= IBEN;       /* enable I2C module */
19          IBFD = ibc;         /* set up I2C baud rate */
20          IBAD = I2C_ID;      /* set up I2C slave address */
21          IBCR &= ~IBIE;      /* disable I2C interrupt */
22          IBCR |= IBSWAI;     /* disable i2C in wait mode */
23      }
24
25      /* SendSlaveID */
26      void SendSlaveID(char cx)
27      {
28
29      #ifdef DBG_ON
30          printf("I2C: 9S12 Send Slave ID\r\n");
31      #endif
32          while(IBSR & IBB);  /* wait until I2C bus is idle */
33
34      #ifdef DBG_ON
35          printf("I2C:  Bus is Idle\r\n");
36      #endif
37
38      #ifdef DBG_ON
39          printf("I2C: 9S12 Generate Start condition\r\n");
40      #endif
41
42          IBCR |= TXRX + MSSL; /* generate a start condition */
43          IBDR = cx;          /* send out slace ID with R/W set to 0 */
44          while(!(IBSR & IBIF)); /* wait for completion of transmission */
45
46      #ifdef DBG_ON
47          printf("I2C: 9S12 Start condition ... OK\r\n");
48      #endif
49
50          IBSR = IBIF;        /* clear IBIF flag */
51
52      #ifdef DBG_ON
53          printf("I2C: 9S12 Send Slave ID ... done\r\n");
54      #endif
55      }
56
57      //#define DBG_ON_R
58
59      /* EERandomRead */
60      char EErandomRead(char ID, char addr)
61      {
62          char dummy;
63
64      #ifdef DBG_ON_R
65          printf("I2C: 9S12 Read a byte from EEPROM\r\n");
66      #endif
67
68          SendSlaveID(ID);
69
70      #ifdef DBG_ON_R
71          printf("I2C:9S12 Wait for Transmission Acknowledgment\r\n");
72      #endif
73          if (IBSR & RXAK)    /* wait for transmission ackn nowledgment */
```

```
74          return -1;
75      #ifdef DBG_ON_R
76          printf("I2C:EEPROM ... Ack\r\n");
77      #endif
78
79      #ifdef DBG_ON_R
80          printf("I2C: 9S12 Send Address\r\n");
81      #endif
82          IBDR = addr;        /* send out EEPROM address */
83
84          while(!(IBSR & IBIF)); /* wait util the address is sent out */
85
86          IBSR = IBIF;        /* clear IBIF flag */
87
88          if (IBSR & RXAK)
89              return -1;
90
91      #ifdef DBG_ON_R
92          printf("I2C: EEPROM ... Ack\r\n");
93      #endif
94
95      #ifdef DBG_ON_R
96          printf("I2C: 9S12 Generate Restart condition and prepare to read\r\n");
97      #endif
98          IBCR |= RSTA;       /* generate restart condition */
99          IBDR = ID | 0x01;   /* prepare to read */
100
101         while (!(IBSR & IBIF));
102
103         IBSR = IBIF;
104         if (IBSR & RXAK)
105         return -1;
106     #ifdef DBG_ON_R
107         printf("I2C: EEPROM ... Ack\r\n");
108     #endif
109
110         IBCR |= TXAK; /* prepare to send NACK */
111         IBCR &= ~TXRX; /* perform reception */
112     #ifdef DBG_ON_R
113         printf("I2C:9S12 Trigger 9 clock pulses to read\r\n");
114     #endif
115         dummy = IBDR; /* dummy read to trigger 9 clock pulses */
116         while(!(IBSR & IBIF)); /* wait for data to shift in */
117     #ifdef DBG_ON_R
118         printf("I2C: EEPROM Send Byte... OK\r\n");
119     #endif
120         IBSR = IBIF;
121         IBCR &= ~MSSL; /* generate a stop condition */
122     #ifdef DBG_ON_R
123         printf("I2C: 9S12 Generate Stop condition and finish reading a byte\r\n");
124     #endif
125         return IBDR;
126     }
127
128     /* EEbyteWrite */
129     char EEbyteWrite(char ID, char addr, char data)
130     {
131
132     #ifdef DBG_ON
```

```
133         printf("I2C: 9S12 Write a byte to EEPROM\r\n");
134     #endif
135
136     #ifdef DBG_ON
137         printf("I2C: 9S12 Wait for Acknowledgment\r\n");
138     #endif
139         SendSlaveID(ID);
140
141         if (IBSR & RXAK) /* error if EEPROM does not acknowledge */
142             return -1;
143     #ifdef DBG_ON
144         printf("I2C: EEPROM ... Ack\r\n");
145     #endif
146
147     #ifdef DBG_ON
148         printf("I2C: 9S12 Send Address\r\n");
149     #endif
150         IBDR = addr;   /* send out address of the location to be written */
151         while(!(IBSR & IBIF));
152         IBSR = IBIF;   /* clear the IBIF flag */
153         if (IBSR & RXAK) /* error if EEPROM does not acknowledge */
154             return -1;
155     #ifdef DBG_ON
156         printf("I2C: EEPROM ... Ack\r\n");
157     #endif
158
159     #ifdef DBG_ON
160         printf("I2C: 9S12 Send Data\r\n");
161     #endif
162         IBDR = data;   /* send out the data byte */
163         while(!(IBSR&IBIF));
164         IBSR = IBIF;   /* clear the IBIF flag */
165         if (IBSR & RXAK) /* error if EEPROM does not respond */
166             return -1;
167     #ifdef DBG_ON
168         printf("I2C: EEPROM ... Ack\r\n");
169     #endif
170
171         IBCR &= ~MSSL; /* generate a stop condition */
172     #ifdef DBG_ON
173         printf("I2C: 9S12 Generate Stop condition and finish reading a byte\r\n");
174     #endif
175         return 0;      /* normal write code */
176     }
177
178     /* eeAckPoll */
179     void eeAckPoll(char ID)
180     {
181         SendSlaveID(ID);
182         while(IBSR & RXAK){
183         IBCR |= RSTA; /* generate a restart condition */
184         IBDR = ID;    /* send out EEPROM ID */
185         while(!(IBSR & IBIF));
186             IBSR = IBIF;  /* clear the IBIF flag */
187         } ; /* continue if EEPROM did not acknowledge */
188         IBCR &= ~MSSL;   /* generate a stop condition */
189     }
190
191     /* EEbyteWrite */
```

```
192  char EEPageWrite(char ID, char addr, char * data)
193  {
194      char i;
195  #ifdef DBG_ON
196      printf("I2C: 9S12 Write a byte to EEPROM\r\n");
197  #endif
198
199  #ifdef DBG_ON
200      printf("I2C: 9S12 Wait for Acknowledgment\r\n");
201  #endif
202      SendSlaveID(ID);
203
204      if (IBSR & RXAK) /* error if EEPROM does not acknowledge */
205          return -1;
206  #ifdef DBG_ON
207      printf("I2C: EEPROM ... Ack\r\n");
208  #endif
209
210  #ifdef DBG_ON
211      printf("I2C: 9S12 Send Address\r\n");
212  #endif
213      IBDR = addr;   /* send out address of the location to be written */
214      while(!(IBSR & IBIF));
215      IBSR = IBIF;   /* clear the IBIF flag */
216      if (IBSR & RXAK) /* error if EEPROM does not acknowledge */
217          return -1;
218  #ifdef DBG_ON
219      printf("I2C: EEPROM ... Ack\r\n");
220  #endif
221
222  //#ifdef DBG_ON
223      printf("I2C: 9S12 Send Data\r\n");
224  //#endif
225      for(i = 0; i < 8; i++) {
226
227      IBDR = data[i];  /* send out the data byte */
228      while(!(IBSR&IBIF));
229      IBSR = IBIF;   /* clear the IBIF flag */
230      if (IBSR & RXAK) /* error if EEPROM does not respond */
231  return -1;
232
233  //#ifdef DBG_ON
234          printf("I2C: EEPROM ... Ack\r\n");
235  //#endif
236      }
237      IBCR &= ~MSSL; /* generate a stop condition */
238  #ifdef DBG_ON
239      printf("I2C: 9S12 Generate Stop condition and finish reading a byte\r\n");
240  #endif
241      return 0;      /* normal write code */
242  }
```

Main

```
1  /* ******************************************************************** *
2  * Project:    I2C_EEPROM                                               *
3  * Purpose:    EE128 Lab 7, SPI and I2C Communication                   *
```

```
4       *                                                                   *
5       * Notes:     See Lab 7 manual for a detailed testbench schematic *
6       * ******************************************************************** */
7
8       #include <hidef.h>
9       #include <stdio.h>
10      #include <mc9s12dg256.h>
11      #include "i2c.h"
12
13
14      #define DBG_ON            /* print out debugging information if enabled */
15
16
17      void clearscreen(void)
18      {
19          putchar(0x0C);
20      }
21
22      void main(void) {
23
24          const char *msgMemo = "I2C based EEPROM Reading and Writing";
25          char rd;
26          char memchar = 0x00;
27
28          char eeprom_address = 0x0A; /* address in EEPROM to write a byte to */
29          char eeprom_data = 0x05; /* data byte to be written to EEPROM */
30
31          char data1[8] = {0x01,0x02,0x04,0x08,0x01,0x02,0x04,0x08};
32          char data2[8] = {0x0a,0x0b,0x0c,0x0d,0x0d,0x0c,0x0b,0x0a};
33
34          char i;
35
36          clearscreen();
37          printf("EE128 LAB 7\r\n");
38          printf("I2C based EEPROM Reading and Writing\r\n");
39
40          printf("**************************************\r\n");
41          printf("  9S12 will write a byte to EEPROM\r\n ");
42          printf("**************************************\r\n");
43          printf("EEPROM address: 0x%x\r\n", eeprom_address);
44          printf("EEPROM data: 0x%x\r\n", eeprom_data);
45
46      #ifdef DBG_ON
47          printf("- running in DEBUG mode -\r\n");
48      #endif
49
50          Init_I2C(0x1F,0xFE);     /* configure I2C module */
51
52          rd = EEbyteWrite(0xA0,eeprom_address,eeprom_data);
53          eeAckPoll(0xA0);        /* make sure internal write operation is complete */
54
55          printf("\r\n");
56          printf("**************************************\r\n");
57          printf("  9S12 will read a byte from EEPROM\r\n");
58          printf("**************************************\r\n");
59
60          memchar = EErandomRead(0xA0,eeprom_address);
61
62          printf("EEPROM @ 0x%x = 0x%x\r\n", eeprom_address, memchar);
```

```
63
64          eeprom_address = 16;
65
66          printf("\r\n");
67          printf("***************************************\r\n");
68          printf("  9S12 will write pages to EEPROM\r\n");
69          printf("***************************************\r\n");
70
71          EEPageWrite(0xA0, eeprom_address, data1);
72          eeAckPoll(0xA0);
73          EEPageWrite(0xA0, eeprom_address + 8, data2);
74          eeAckPoll(0xA0);
75
76          printf("\r\n");
77          printf("***************************************\r\n");
78          printf("  9S12 will read pages back EEPROM\r\n");
79          printf("***************************************\r\n");
80
81          for(i = 0; i < 16; i++) {
82
83          printf("Reading Data %d ", i);
84              memchar = EErandomRead(0xA0,eeprom_address+i);
85
86              printf("EEPROM @ 0x%x = 0x%x\r\n", eeprom_address+i,  memchar);
87          }
88
89
90          for(;;) {} /* wait forever */
91      }
```

Technical Problems

There were no major technical problems encountered.

Conclusion

In this lab we successfully used two very common serial communication protocols. They showed us how we can extend our I/O and how we can store data in non volatile memory.