Muhammad Sharjeel and Naafiul Hossain

115185427

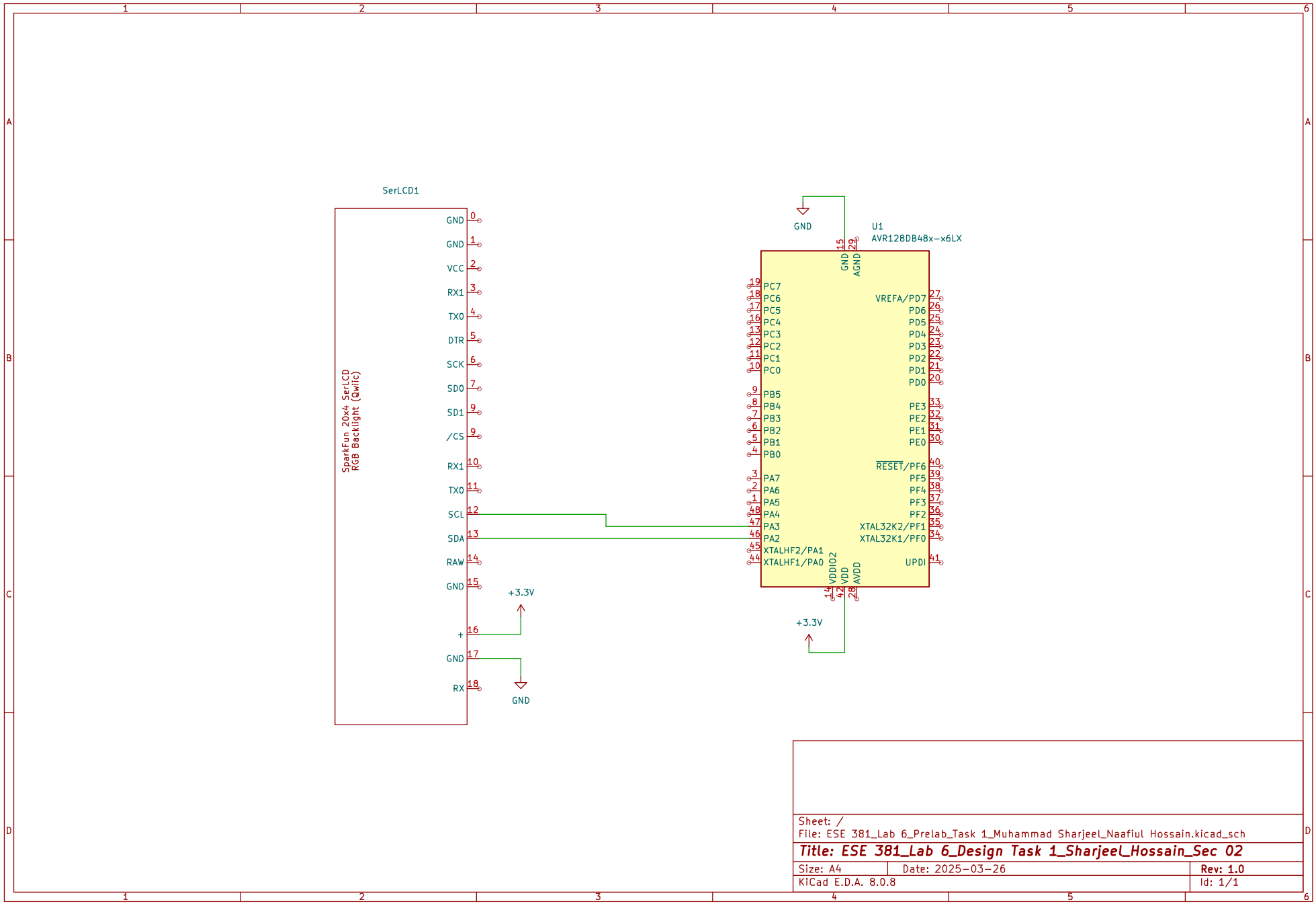
115107623

Lab 6: AVR128DB48 I2C Module with Sparkfun 20 x 4 SerLCD and LM75 Temperature Sensor

ESE 381 Section L02

Bench 7

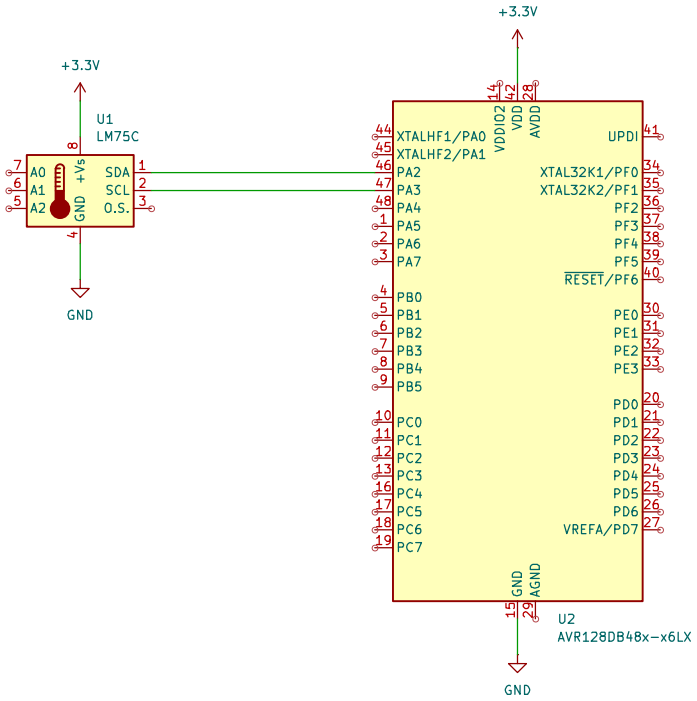
Breadboard: K2



Sheet: /
File: ESE 381_Lab 6_Prelab_Task 1_Muhammad Sharjeel_Naafiul Hossain.kicad_sch

Title: ESE 381_Lab 6_Design Task 1_Sharjeel_Hossain_Sec 02

Size: A4	Date: 2025-03-26	Rev: 1.0
KiCad E.D.A. 8.0.8		Id: 1/1



Sheet: /
File: ESE 381_Lab 6_Prelab_Task 2_Sharjeel_Hossain_schematic.kicad_sch

Title: ESE 381_Lab 6_Prelab_Design Task 2_Sharjeel_Hossain_Sec02

Size: A4	Date: 2025-03-26	Rev: 1.0
KiCad E.D.A. 8.0.8		Id: 1/1

```c
//
//  write_SerLCD_image_TWI0.c
//
//
//  Created by Muhammad Sharjeel on 3/27/25.
//

#define F_CPU 4000000UL
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

#define LINE_LENGTH 20

// Display buffers (each line holds 20 characters plus a null terminator)
char dsp_buff1[LINE_LENGTH + 1];
char dsp_buff2[LINE_LENGTH + 1];
char dsp_buff3[LINE_LENGTH + 1];
char dsp_buff4[LINE_LENGTH + 1];


void init_twi0_SerLCD (void) {

    TWI0.MBAUD = 0x01;                          // this is the calculated baud value of 15
    TWI0.MCTRLA |= TWI_ENABLE_bm;              // enable the TWI module: 0x01

    TWI0.MSTATUS = (TWI0.MSTATUS & (~0x03)) | TWI_BUSSTATE_IDLE_gc;
    // set the bus state to 'idle': 0x01

    PORTA.DIR |= 0x03;          // setting PA3 as output for SCL
}



int write_twi0_SerLCD (uint8_t saddr, uint8_t data) {

    while ((TWI0.MSTATUS & 0x03) != 0x01)   // wait until idle
    {
        ; // Null, do nothing
    }

    TWI0.MADDR = ((saddr << 1) | 0x00);     // send slave address and write command

    TWI0.MCTRLB = 0x02;                     // acknowledge

    TWI0.DATA = data;                       // send data

```

```
48      while ((TWI0.MSTATUS & 0x40) == 0);     // wait until byte is sent
49
50      TWI0.MCTRLB = 0x07;                     // Nack
51
52      return 0;
53  }
54
55
56  // Update the SerLCD with our buffers (we only use line 1 and 2 here)
57  void update_SerLCD(void) {
58
59      // Create pointer for the display buffers
60      char* buffers[] = { dsp_buff1, dsp_buff2};
61
62      // Clear the display
63      write_spi0_SerLCD ('|');
64      write_spi0_SerLCD ('-');
65
66      // loop through the 4 display buffers
67      for (int i = 0; i < 2; i++) {
68          // Calculate the starting cursor position for each line:
69          //uint8_t position = 0x80 + (i * 0x40);
70          // Set cursor to start of each line.
71          //write_spi0_SerLCD(0xFE); // Command Character
72          //write_spi0_SerLCD(position); // Each line's start address
73          // Send the characters
74          for (int j = 0; j < LINE_LENGTH; j++) {
75              write_spi0_SerLCD(buffers[i][j]);
76          }
77      }
78
79      _delay_ms(10);
80  }
81
82
83
84
85  int main(void) {
86
87      init_twi0_SerLCD ();
88
89      sprintf(disp_buff1, "The first line_____");
90
91      while (1) {
92          write_twi0_SerLCD(0x72, 'A');
93          update_SerLCD ();
94          _delay_ms(1000); // Update every 1 second.
95      }
96
```

```
 97
 98      return 0;
 99  }
100
101
102
103
104
105
106
107
```

```c
/*
 * read_LM75_temp_test.c
 *
 * Created: 3/23/2025 5:32:27 PM
 * Author : Naafiul Hossain
 */

#include <avr/io.h>
#define F_CPU 4000000   //Frequency used by delay macros.
#include <util/delay.h> //Header for delay macros and functions.
#include <string.h>
#include <stdio.h>

#define LM75_ADDR 0x48     // LM75 address
#define TEMPERATURE_ADDR 0x00   //slave address of temperature register


uint8_t temp_reg_high;  //high byte of LM75 Temperature register
uint8_t temp_reg_low;   //low byte of LM75 Temperature register
uint16_t LM75_temp_reg = 0; //LM75 Temperature register contents
int16_t LM75_temp = 0;  //right adjusted 2's complement, resolution 0.1C
char dsp_buff1[17];     //buffer for line 1 of LCD image
//function protypoes

void TWI0_LM75_init(void);  //Initialize TWI0 module to talk to LM75
uint16_t TWIO_LM75_read(uint8_t saddr);
int TWI0_LM75_write(unsigned char saddr, unsigned char raddr, unsigned char
  data);

void TWI0_LM75_init(){
    TWI0_MBAUD=0x01;
    TWI0_MCTRLA=0X01;
    TWI0.DBGCTRL = 0x01;
    TWI0.MSTATUS = 0x01; //Force bus state to idle


}

int TWI0_LM75_write(uint8_t saddr, uint8_t raddr, uint8_t data) {
    while((TWI0.MSTATUS & 0x03) != 0x01) ; /* wait until idle */

    TWI0.MADDR = saddr << 1;         /* send address for write */
    while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until saddr sent */

    //The next write clears the WIF flag
    TWI0.MDATA = raddr;              /* send memory address */
    while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until raddr sent */

    //The next write clears the WIF flag
```

```
49      TWI0.MDATA = data;                /* send data */
50      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until data sent */
51
52      //The next write clears the WIF flag
53      TWI0.MCTRLB |= 0x03;          /* issue a stop */
54
55      return 0;
56  }
57
58  uint16_t TWI0_LM75_read(uint8_t saddr)
59  {
60      while((TWI0.MSTATUS & 0x03) != 0x01) ;  // wait until idle
61
62      TWI0.MADDR = ((saddr << 1) | 0x01);     // send slave address and read
        command
63
64      while((TWI0.MSTATUS & 0x80) == 0);      // RIF flag, wait until byte is
        received
65      // WIF flag does not work here
66      temp_reg_high = TWI0.MDATA;             //clears the RIF flag
67
68      TWI0.MCTRLB = 0x02;                 //MCMD - issue ack followed by a byte
        read operation
69
70      while((TWI0.MSTATUS & 0x80) == 0);      // RIF flag, wait until data
        received
71
72      temp_reg_low = TWI0.MDATA;              //clears the RIF flag
73
74      TWI0.MCTRLB = 0x07;                 //MCMD issue nack followed by a stop
75
76      return (uint16_t)((temp_reg_high << 8) | (temp_reg_low & 0x80));    //read
         data from received data buffer
77  }
78
79
80  int main(void)
81  {
82     TWI0_LM75_init();
83      while (1)
84      {
85          while((TWI0.MSTATUS & 0x03) != 0x01) ; /* wait until I2C bus idle */
86          LM75_temp_reg = TWI0_LM75_read(LM75_ADDR);
87          LM75_temp = ((int16_t)LM75_temp_reg) >> 7;
88          //      sprintf(dsp_buff1, "Temp = %4d", (LM75_temp >> 1)); //integer
            result
89          //      sprintf(dsp_buff1, "Temp = %4d.%d", (LM75_temp >> 1),
            ((LM75_temp%2) ? 5 : 0) ); //only for pos temps
90          sprintf(dsp_buff1, "Temp = %.1f", ((float)(LM75_temp)/2.0)); //
```

```
            requires vprintf library
91          //      _delay_ms(1000);
92      }
93  }
94
95
```

```c
/*
 * display_LM75_temp.c
 *
 * Created: 3/26/2025 12:41:18 AM
 * Author : Naafiul Hossain
 */

#include <avr/io.h>
#define F_CPU 4000000UL  // Clock frequency for delay functions
#include <util/delay.h>
#include <stdio.h>

#define LM75_ADDR 0x48     // LM75 I2C address
#define TEMPERATURE_ADDR 0x00  // Address of the temperature register in LM75

uint8_t temp_reg_high;  //high byte of LM75 Temperature register
uint8_t temp_reg_low;   //low byte of LM75 Temperature register
uint16_t LM75_temp_reg = 0; //LM75 Temperature register contents
int16_t LM75_temp = 0;  //right adjusted 2's complement, resolution 0.1C
//char dsp_buff1[21];       //buffer for line 1 of LCD image

// SPI and LCD specific definitions
void init_spi0_SerLCD(void);
void write_spi0_SerLCD(unsigned char data);
void select_SS(void);
void deselect_SS(void);
void update_SerLCD(void);
void clear_display_buffs(void);

// I2C and LM75 specific definitions
void TWI0_LM75_init(void);
uint16_t TWI0_LM75_read(uint8_t saddr);
int TWI0_LM75_write(uint8_t saddr, uint8_t raddr, uint8_t data);



// Global display buffers
char dsp_buff1[21];
char dsp_buff2[21];
char dsp_buff3[21];
char dsp_buff4[21];

// Initialize SPI for SerLCD
void init_spi0_SerLCD(void) {
    PORTA.DIRSET = PIN7_bm | PIN6_bm | PIN4_bm;  // PA7: SS, PA6: SCK, PA4: MOSI
    PORTA.DIRCLR = PIN5_bm;                      // PA5: MISO
    SPI0.CTRLA = SPI_MASTER_bm | SPI_PRESC_DIV16_gc | SPI_ENABLE_bm; // SPI mode settings
```

```
48      SPI0.CTRLB = SPI_SSD_bm | SPI_MODE_0_gc;    // More SPI settings
49      SPI0.CTRLB &= ~SPI_BUFEN_bm;                // Normal unbuffered mode
50  }
51
52  // Initialize I2C for LM75
53  void TWI0_LM75_init(void) {
54      TWI0_MBAUD = 0x01;
55      TWI0_MCTRLA = 0x01;
56      TWI0.MSTATUS = 0x01;  // Force bus state to idle
57  }
58
59  uint16_t TWI0_LM75_read(uint8_t saddr)
60  {
61      while((TWI0.MSTATUS & 0x03) != 0x01) ;  // wait until idle
62
63      TWI0.MADDR = ((saddr << 1) | 0x01);     // send slave address and read
        command
64
65      while((TWI0.MSTATUS & 0x80) == 0);      // RIF flag, wait until byte is
        received
66      // WIF flag does not work here
67      temp_reg_high = TWI0.MDATA;             //clears the RIF flag
68
69      TWI0.MCTRLB = 0x02;                 //MCMD - issue ack followed by a byte
        read operation
70
71      while((TWI0.MSTATUS & 0x80) == 0);      // RIF flag, wait until data
        received
72
73      temp_reg_low = TWI0.MDATA;              //clears the RIF flag
74
75      TWI0.MCTRLB = 0x07;                 //MCMD issue nack followed by a stop
76      return (uint16_t)((temp_reg_high << 8) | (temp_reg_low & 0x80));    //read
           data from received data buffer
77  }
78
79
80  int TWI0_LM75_write(uint8_t saddr, uint8_t raddr, uint8_t data) {
81      while((TWI0.MSTATUS & 0x03) != 0x01) ; /* wait until idle */
82
83      TWI0.MADDR = saddr << 1;         /* send address for write */
84      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until saddr sent */
85
86      //The next write clears the WIF flag
87      TWI0.MDATA = raddr;              /* send memory address */
88      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until raddr sent */
89
90      //The next write clears the WIF flag
91      TWI0.MDATA = data;               /* send data */
```

```
92      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until data sent */
93
94      //The next write clears the WIF flag
95      TWI0.MCTRLB |= 0x03;          /* issue a stop */
96
97      return 0;
98  }
99
100 // Main function where both devices are used
101 int main(void) {
102     init_spi0_SerLCD();  // Initialize SPI for SerLCD
103     TWI0_LM75_init();    // Initialize I2C for LM75
104
105     clear_display_buffs();  // Clear display buffers for LCD
106
107     while (1) {
108         // Read temperature from LM75 using I2C
109         uint16_t LM75_temp_reg = TWI0_LM75_read(LM75_ADDR);
110         int16_t LM75_temp = ((int16_t)LM75_temp_reg) >> 7;
111         sprintf(dsp_buff1, "Temp = %.1f C", ((float)(LM75_temp) / 2.0)); //
              Display temperature
112
113         // Update SerLCD display with SPI
114         sprintf(dsp_buff2, "More data 1");
115         sprintf(dsp_buff3, "More data 2");
116         sprintf(dsp_buff4, "More data 3");
117         update_SerLCD();  // Send updated buffers to SerLCD via SPI
118
119         _delay_ms(1000);  // Delay to slow down updates (optional)
120     }
121 }
122
123 // Functions for SPI communication
124 void write_spi0_SerLCD(unsigned char data) {
125     select_SS();
126     SPI0.DATA = data;
127     while (!(SPI0.INTFLAGS & SPI_IF_bm));
128     deselect_SS();
129 }
130
131 void select_SS(void) { PORTA.OUT &= ~PIN7_bm; }
132 void deselect_SS(void) { PORTA.OUT |= PIN7_bm; }
133
134 void update_SerLCD(void) {
135     write_spi0_SerLCD(0xFE);
136     write_spi0_SerLCD(0x80); // First line
137     for (uint8_t i = 0; i < 20; i++) {
138         write_spi0_SerLCD(dsp_buff1[i]);
139     }
```

```
140
141     write_spi0_SerLCD(0xFE);
142     write_spi0_SerLCD(0xC0); // Second line
143     for (uint8_t i = 0; i < 20; i++) {
144         write_spi0_SerLCD(dsp_buff2[i]);
145     }
146
147     write_spi0_SerLCD(0xFE);
148     write_spi0_SerLCD(0x94); // Third line
149     for (uint8_t i = 0; i < 20; i++) {
150         write_spi0_SerLCD(dsp_buff3[i]);
151     }
152
153     write_spi0_SerLCD(0xFE);
154     write_spi0_SerLCD(0xD4); // Fourth line
155     for (uint8_t i = 0; i < 20; i++) {
156         write_spi0_SerLCD(dsp_buff4[i]);
157     }
158 }
159
160 void clear_display_buffs(void) {
161     memset(dsp_buff1, ' ', 20);
162     dsp_buff1[20] = '\0';
163     memset(dsp_buff2, ' ', 20);
164     dsp_buff2[20] = '\0';
165     memset(dsp_buff3, ' ', 20);
166     dsp_buff3[20] = '\0';
167     memset(dsp_buff4, ' ', 20);
168     dsp_buff4[20] = '\0';
169 }
170
```

```c
/*
 * temp_meas_LM75.c
 *
 * Created: 3/27/2025 1:58:21 PM
 * Author : Naafiul Hossain
 */

#include <avr/io.h>
#define F_CPU 4000000UL  // Clock frequency for delay functions
#include <util/delay.h>
#include <stdio.h>

#define LM75_ADDR 0x48     // LM75 I2C address
#define TEMPERATURE_ADDR 0x00  // Address of the temperature register in LM75

uint8_t temp_reg_high;  //high byte of LM75 Temperature register
uint8_t temp_reg_low;   //low byte of LM75 Temperature register
uint16_t LM75_temp_reg = 0; //LM75 Temperature register contents
int16_t LM75_temp = 0;  //right adjusted 2's complement, resolution 0.1C
//char dsp_buff1[21];       //buffer for line 1 of LCD image

// SPI and LCD specific definitions
void init_spi0_SerLCD(void);
void write_spi0_SerLCD(unsigned char data);
void select_SS(void);
void deselect_SS(void);
void update_SerLCD(void);
void clear_display_buffs(void);

// I2C and LM75 specific definitions
void TWI0_LM75_init(void);
uint16_t TWI0_LM75_read(uint8_t saddr);
int TWI0_LM75_write(uint8_t saddr, uint8_t raddr, uint8_t data);



// Global display buffers
char dsp_buff1[21];
char dsp_buff2[21];
char dsp_buff3[21];
char dsp_buff4[21];

// Initialize SPI for SerLCD
void init_spi0_SerLCD(void) {
    PORTA.DIRSET = PIN7_bm | PIN6_bm | PIN4_bm;  // PA7: SS, PA6: SCK, PA4: MOSI
    PORTA.DIRCLR = PIN5_bm;                      // PA5: MISO
    SPI0.CTRLA = SPI_MASTER_bm | SPI_PRESC_DIV16_gc | SPI_ENABLE_bm; // SPI mode settings
```

```
48      SPI0.CTRLB = SPI_SSD_bm | SPI_MODE_0_gc;    // More SPI settings
49      SPI0.CTRLB &= ~SPI_BUFEN_bm;                // Normal unbuffered mode
50  }
51
52  // Initialize I2C for LM75
53  void TWI0_LM75_init(void) {
54      TWI0_MBAUD = 0x01;
55      TWI0_MCTRLA = 0x01;
56      TWI0.MSTATUS = 0x01;  // Force bus state to idle
57  }
58
59  uint16_t TWI0_LM75_read(uint8_t saddr)
60  {
61      while((TWI0.MSTATUS & 0x03) != 0x01) ;  // wait until idle
62
63      TWI0.MADDR = ((saddr << 1) | 0x01);     // send slave address and read
        command
64
65      while((TWI0.MSTATUS & 0x80) == 0);      // RIF flag, wait until byte is
        received
66      // WIF flag does not work here
67      temp_reg_high = TWI0.MDATA;             //clears the RIF flag
68
69      TWI0.MCTRLB = 0x02;                 //MCMD - issue ack followed by a byte
        read operation
70
71      while((TWI0.MSTATUS & 0x80) == 0);      // RIF flag, wait until data
        received
72
73      temp_reg_low = TWI0.MDATA;              //clears the RIF flag
74
75      TWI0.MCTRLB = 0x07;                 //MCMD issue nack followed by a stop
76      return (uint16_t)((temp_reg_high << 8) | (temp_reg_low & 0x80));    //read
           data from received data buffer
77  }
78
79
80  int TWI0_LM75_write(uint8_t saddr, uint8_t raddr, uint8_t data) {
81      while((TWI0.MSTATUS & 0x03) != 0x01) ; /* wait until idle */
82
83      TWI0.MADDR = saddr << 1;         /* send address for write */
84      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until saddr sent */
85
86      //The next write clears the WIF flag
87      TWI0.MDATA = raddr;              /* send memory address */
88      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until raddr sent */
89
90      //The next write clears the WIF flag
91      TWI0.MDATA = data;               /* send data */
```

```
92      while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until data sent */
93
94      //The next write clears the WIF flag
95      TWI0.MCTRLB |= 0x03;          /* issue a stop */
96
97      return 0;
98  }
99
100 // Main function where both devices are used
101 // Main function where both devices are used
102 int main(void) {
103     init_spi0_SerLCD();  // Initialize SPI for SerLCD
104     TWI0_LM75_init();    // Initialize I2C for LM75
105
106     clear_display_buffs();  // Clear display buffers for LCD
107
108     while (1) {
109         // Read temperature from LM75 using I2C
110         uint16_t LM75_temp_reg = TWI0_LM75_read(LM75_ADDR);
111         int16_t LM75_temp_celsius = ((int16_t)LM75_temp_reg) >> 7;
112         float temperature_celsius = ((float)(LM75_temp_celsius) / 2.0);
113         float temperature_fahrenheit = (temperature_celsius * 9.0 / 5.0) +
              32.0;
114
115         sprintf(dsp_buff1, "Temp = %.1f C", temperature_celsius); // Display
              temperature in Celsius
116         sprintf(dsp_buff2, "Temp = %.1f F", temperature_fahrenheit); //
              Display temperature in Fahrenheit
117
118         // Update SerLCD display with SPI
119         update_SerLCD();  // Send updated buffers to SerLCD via SPI
120
121         _delay_ms(1000);  // Delay to slow down updates (optional)
122     }
123 }
124 // Functions for SPI communication
125 void write_spi0_SerLCD(unsigned char data) {
126     select_SS();
127     SPI0.DATA = data;
128     while (!(SPI0.INTFLAGS & SPI_IF_bm));
129     deselect_SS();
130 }
131
132 void select_SS(void) { PORTA.OUT &= ~PIN7_bm; }
133 void deselect_SS(void) { PORTA.OUT |= PIN7_bm; }
134
135 void update_SerLCD(void) {
136     write_spi0_SerLCD(0xFE);
137     write_spi0_SerLCD(0x80); // First line
```

```
138     for (uint8_t i = 0; i < 20; i++) {
139         write_spi0_SerLCD(dsp_buff1[i]);
140     }
141
142     write_spi0_SerLCD(0xFE);
143     write_spi0_SerLCD(0xC0); // Second line
144     for (uint8_t i = 0; i < 20; i++) {
145         write_spi0_SerLCD(dsp_buff2[i]);
146     }
147
148     write_spi0_SerLCD(0xFE);
149     write_spi0_SerLCD(0x94); // Third line
150     for (uint8_t i = 0; i < 20; i++) {
151         write_spi0_SerLCD(dsp_buff3[i]);
152     }
153
154     write_spi0_SerLCD(0xFE);
155     write_spi0_SerLCD(0xD4); // Fourth line
156     for (uint8_t i = 0; i < 20; i++) {
157         write_spi0_SerLCD(dsp_buff4[i]);
158     }
159 }
160
161 void clear_display_buffs(void) {
162     memset(dsp_buff1, ' ', 20);
163     dsp_buff1[20] = '\0';
164     memset(dsp_buff2, ' ', 20);
165     dsp_buff2[20] = '\0';
166     memset(dsp_buff3, ' ', 20);
167     dsp_buff3[20] = '\0';
168     memset(dsp_buff4, ' ', 20);
169     dsp_buff4[20] = '\0';
170 }
171
```