```
/**
 *  @file    main.c
 *  @author  Tomas Fryza, Brno University of Technology, Czechia
 *  @version V1.2
 *  @date    Oct 27, 2018
 *  @brief   Scan the TWI bus for all connected slave devices and
transmit
 *           info to lcd.
 */

/* Includes ------------------------------------------------------------
-----*/
#include "settings.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>         /* itoa() function */
#include "twi.h"
#include "uart.h"
#include "lcd.h"


/* Constants and macros ------------------------------------------------
-----*/
/**
 *  @brief Define lcd buad rate.
 */
 #define UART_BAUD_RATE 9600
 #define DHT12_SLAVE 0x5c

 #define TEMP_BYTE_ADDR 0x02
 #define HUMI_BYTE_ADDR 0x00

 #define BUTTON_PRESSED_UP !(PIND & (1<<PIND2)) //default = 1, if press =
0
 #define BUTTON_PRESSED_DOWN !(PIND & (1<<PIND3)) //default = 1, if press
= 0

 #define TEMPERATURE_MAX 32
 #define HUMIDITY_MAX 32


/* Function prototypes -------------------------------------------------
-----*/

void setup(void);
void setup_buttons(void);
void fsm_twi_scanner(void);
void show_bar_up(uint8_t value);
void show_bar_down(uint8_t value);
void setup_timer0(void);

/* Global variables ----------------------------------------------------
-----*/
typedef enum
{
    IDLE_STATE = 1,
    HUMIDITY_STATE,
    TEMPERATURE_STATE,
```

```
    COMPARE_STATE,
    ALARM_STATE
} state_t;
/* FSM for scanning TWI bus */
state_t twi_state = IDLE_STATE;

uint8_t lcd_user_symbols[8*2] = {
     0x00, /* WRITE YOUR DEFINITIONS HERE */};

struct Values {
    uint8_t humidity_integer;
    uint8_t humidity_decimal;
    uint8_t temperature_integer;
    uint8_t temperature_decimal;
};
/* Data structure for humidity and temperature values */
struct Values Meteo_values;

uint8_t flag_alarm_temperature  = 0;
uint8_t flag_alarm_humidity     = 0;

uint8_t flag_window_up          = 0;
uint8_t flag_window_down        = 0;


/* Functions ---------------------------------------------------------
-----*/
int main(void)
{
    /* Initializations */
    setup();
      setup_buttons();
    setup_timer0();

    /* Enables interrupts by setting the global interrupt mask */
    sei();


    /* Forever loop */
    while (1) {
        /* Cycle here, do nothing, and wait for an interrupt */
    }

    return 0;
}


ISR(PCINT2_vect)
{

  if( BUTTON_PRESSED_UP )
  {

    flag_window_up = 1;
  }
  else if( BUTTON_PRESSED_DOWN )
  {
```

```
    flag_window_down = 1;
  }
  else
  {
    flag_window_up =    0;
    flag_window_down =  0;
  }

}

/************************************************************************
*******
 * Function: setup()
 * Purpose:  Initialize lcd, TWI, and Timer/Counter1.
 * Input:    None
 * Returns:  None

************************************************************************
*****/
void setup(void)
{

    /* Initialize lcd: asynchronous, 8-bit data, no parity, 1-bit stop */
    uart_init(UART_BAUD_SELECT(UART_BAUD_RATE, F_CPU));

    /* Initialize TWI */
    twi_init();

    /* Timer/Counter1: update FSM state */
    /* Clock prescaler 64 => overflows every 262 ms */
    TCCR1B |= _BV(CS11) | _BV(CS10);
    /* Overflow interrupt enable */
    TIMSK1 |= _BV(TOIE1);

    uint8_t i;

    /* Initialize display and select type of cursor */
    lcd_init(LCD_DISP_ON_CURSOR_BLINK);

    /* Set pointer to beginning of CG RAM memory */
    lcd_command(1<<LCD_CGRAM);
    /* Store two new characters, i.e. 8x2 bytes */
    for (i = 0; i < (8*2); i++) {
        lcd_data(lcd_user_symbols[i]);
    }

    /* Clear display and set cursor to home position */
    lcd_clrscr();
}

/************************************************************************
*******
 * Function: setup_buttons()
 * Purpose:  Initialize buttons.
 * Input:    None
 * Returns:  None
```

```
*************************************************************************
*****/
void setup_buttons(void)
{

/****************** PIN CHANGE UP BUTTON
*****************************************/
//Set input pin (PD2)
DDRD &= ~_BV(PD2);
//Internal pull-up (PD2)
PORTD |= _BV(PD2);

//External interrupt enable pin change Interrupt by PCINT2
//Enable pin change Interrupt 2
PCICR |= _BV(PCIE2);
//Enable pin change interrupt at pin PCINT0
PCMSK2 |= _BV(PCINT18);

/****************** PIN CHANGE DOWN BUTTON
*****************************************/
//Set input pin (PD3)
DDRD &= ~_BV(PD3);
//Internal pull-up (PD3)
PORTD |= _BV(PD3);

//External interrupt enable pin change Interrupt by PCINT2
//Enable pin change Interrupt 2
PCICR |= _BV(PCIE2);
//Enable pin change interrupt at pin PCINT19
PCMSK2 |= _BV(PCINT19);

}

/*************************************************************************
*******
 * Function: setup_timer0()
 * Purpose:  Initialize timer 0.
 * Input:    None
 * Returns:  None

*************************************************************************
*****/
void setup_timer0(void)
{
  TCCR0B |= _BV(CS02) | _BV(CS00);
  TIMSK0 |= _BV(TOIE0);
}

/*************************************************************************
*******
 * Function: Timer/Counter0 overflow interrupt
 * Purpose:

*************************************************************************
*****/
ISR(TIMER0_OVF_vect)
{
```

```
  static uint8_t i_temp_up = 0;
  static uint8_t i_temp_down = 0;
  static int8_t i = 0;

  if(flag_window_up)
  {
    i_temp_up++;
    if((i_temp_up == 65) && (i < 10)) //Moreless each 65 times is equal
to 1 sec
    {
      i++;
      show_bar_up(i);
      i_temp_up=0;
    }

  }
  else if(flag_window_down)
  {
    i_temp_down++;
    if((i_temp_down == 65) && (i >= 0)) //Moreless each 65 times is equal
to 1 sec
    {
      show_bar_down(i);
      i--;
      i_temp_down=0;
    }

  }

}

/*************************************************************************
*******
 * Function: Timer/Counter1 overflow interrupt
 * Purpose:  Update state of TWI Finite State Machine.

*************************************************************************
*****/
ISR(TIMER1_OVF_vect)
{
    fsm_twi_scanner();
}

/*************************************************************************
*******
 * Function: fsm_twi_scanner()
 * Purpose:  TWI Finite State Machine transmits all slave addresses.
 * Input:    None
 * Returns:  None

*************************************************************************
*****/
void fsm_twi_scanner(void)
{
    /* Static variable inside a function keeps its value between callings
*/
    uint8_t twi_status;
    char lcd_string[5];
```

```
    char humidity_string;
    char temperature_string;

    switch (twi_state) {
    case IDLE_STATE:

        twi_state = HUMIDITY_STATE;
        break;

    /* Transmit address of TWI slave device and check status */
      case HUMIDITY_STATE:
        twi_status = twi_start((DHT12_SLAVE<<1) + TWI_WRITE);
    //    lcd_puts("humidity: ");

        if (twi_status==0) {
            twi_write(0x00);
            twi_stop();
            twi_start((DHT12_SLAVE<<1) + TWI_READ);
            Meteo_values.humidity_integer = twi_read_ack();
            Meteo_values.humidity_decimal = twi_read_nack();
            twi_stop();
            lcd_gotoxy(0, 0);
            lcd_puts("H:");
            itoa(Meteo_values.humidity_integer, humidity_string, 10);
            lcd_puts(humidity_string);
            lcd_puts(",");
            itoa(Meteo_values.humidity_decimal, humidity_string, 10);
            lcd_puts(humidity_string);
            lcd_puts("%");
            twi_state = TEMPERATURE_STATE;

        }
        else {
            itoa(twi_status, lcd_string, 10);
            lcd_puts(lcd_string);
            twi_state = IDLE_STATE;
        }

        break;

    /* Received ACK from slave */
      case TEMPERATURE_STATE:

        twi_status = twi_start((DHT12_SLAVE<<1) + TWI_WRITE);

        if (twi_status==0) {
            twi_write(0x02);
            twi_stop();
            twi_start((DHT12_SLAVE<<1) + TWI_READ);
            Meteo_values.temperature_integer = twi_read_ack();
            Meteo_values.temperature_decimal = twi_read_nack();
            twi_stop();
            lcd_gotoxy(9, 0);
            lcd_puts("T:");
            itoa(Meteo_values.temperature_integer, temperature_string,
10);
            lcd_puts(temperature_string);
            lcd_puts(",");
```

```
            itoa(Meteo_values.temperature_decimal, temperature_string,
10);
            lcd_puts(temperature_string);
            lcd_puts("C");
            twi_state = COMPARE_STATE;
        }
        else {
            twi_state = HUMIDITY_STATE;
        }

        break;

           case COMPARE_STATE:

             if(Meteo_values.temperature_integer > TEMPERATURE_MAX)
             {
               flag_alarm_temperature = 1;
               twi_state = ALARM_STATE;
             }

             if(Meteo_values.humidity_integer > HUMIDITY_MAX)
             {
               flag_alarm_humidity = 1;
               twi_state = ALARM_STATE;
             }

             if((flag_alarm_temperature == 0) && (flag_alarm_humidity ==
0))
             {
               twi_state = IDLE_STATE;
             }

             break;

           case ALARM_STATE:

                if(flag_alarm_temperature)
                {
                  lcd_gotoxy(9, 0);
                  lcd_puts("ALARM  ");
                  flag_alarm_temperature = 0;
                }

                if(flag_alarm_humidity)
                {
                  lcd_gotoxy(0, 0);
                  lcd_puts("ALARM  ");
                  flag_alarm_humidity = 0;
                }

                twi_state = IDLE_STATE;

                break;

    default:
        twi_state = IDLE_STATE;
    } /* End of switch (twi_state) */
}
```

```
void show_bar_up(uint8_t value)
{
    char porcentage_string[5];
    uint8_t porcentage = value*10;
    itoa(porcentage, porcentage_string, 10);
     lcd_gotoxy(11, 1);
     lcd_puts(porcentage_string);
     lcd_gotoxy(14, 1);
     lcd_puts("%");

     switch (value)
     {
       case 0:
         lcd_gotoxy(0, 1);
         lcd_putc(0xFF);
       break;

       case 1:
         lcd_gotoxy(1, 1);
         lcd_putc(0xFF);
       break;

       case 2:
         lcd_gotoxy(2, 1);
         lcd_putc(0xFF);
       break;

       case 3:
         lcd_gotoxy(3, 1);
         lcd_putc(0xFF);
       break;

       case 4:
         lcd_gotoxy(4, 1);
         lcd_putc(0xFF);
       break;

       case 5:
         lcd_gotoxy(5, 1);
         lcd_putc(0xFF);
       break;

       case 6:
         lcd_gotoxy(6, 1);
         lcd_putc(0xFF);
       break;

       case 7:
         lcd_gotoxy(7, 1);
         lcd_putc(0xFF);
       break;

       case 8:
         lcd_gotoxy(8, 1);
         lcd_putc(0xFF);
```

```
       break;

       case 9:
         lcd_gotoxy(9, 1);
         lcd_putc(0xFF);
       break;
     }

  /*  if (value==1) {
        lcd_gotoxy(0, 1);
        lcd_putc(0x00);
    }
    else if (value==2) {
        lcd_gotoxy(1, 1);
        lcd_putc(0x00);
    }
    else if (value==3) {
        lcd_gotoxy(2, 1);
        lcd_putc(0x00);
    }
    else if (value==4) {
        lcd_gotoxy(3, 1);
        lcd_putc(0x00);
    }
    else if (value==5) {
        lcd_gotoxy(4, 1);
        lcd_putc(0x00);
    }
    else if (value==6) {
        lcd_gotoxy(5, 1);
        lcd_putc(0x00);
    }
    else if (value==7) {
        lcd_gotoxy(6, 1);
        lcd_putc(0x00);
    }
    else if (value==8) {
        lcd_gotoxy(7, 1);
        lcd_putc(0x00);
    }
    else if (value==9) {
        lcd_gotoxy(8, 1);
        lcd_putc(0x00);
    }
    else if (value==10) {
        lcd_gotoxy(9, 1);
        lcd_putc(0x00);
    }*/
}

void show_bar_down(uint8_t value)
{
   char string_porcentage[5];
   uint8_t porcentage = value*10;
   itoa(porcentage, string_porcentage, 10);

   lcd_gotoxy(11, 1);
   lcd_puts(string_porcentage);
```

```
lcd_gotoxy(14, 1);
lcd_puts("%");

switch (value)
{
  case 0:
    lcd_gotoxy(0, 1);
    lcd_putc(0x00);
    lcd_gotoxy(12, 1);
    lcd_putc(0x00);
  break;

  case 1:
    lcd_gotoxy(1, 1);
    lcd_putc(0x00);
  break;

  case 2:
    lcd_gotoxy(2, 1);
    lcd_putc(0x00);
  break;

  case 3:
    lcd_gotoxy(3, 1);
    lcd_putc(0x00);
  break;

  case 4:
    lcd_gotoxy(4, 1);
    lcd_putc(0x00);
  break;

  case 5:
    lcd_gotoxy(5, 1);
    lcd_putc(0x00);
  break;

  case 6:
    lcd_gotoxy(6, 1);
    lcd_putc(0x00);
  break;

  case 7:
    lcd_gotoxy(7, 1);
    lcd_putc(0x00);
  break;

  case 8:
    lcd_gotoxy(8, 1);
    lcd_putc(0x00);
  break;

  case 9:
    lcd_gotoxy(9, 1);
    lcd_putc(0x00);
    lcd_gotoxy(13, 1);
    lcd_putc(0x00);
  break;
```

```
   }

  /*if (value==1) {
      lcd_gotoxy(0, 1);
      lcd_putc(0xFF);
  }
  else if (value==2) {
      lcd_gotoxy(1, 1);
      lcd_putc(0xFF);
  }
  else if (value==3) {
      lcd_gotoxy(2, 1);
      lcd_putc(0xFF);
  }
  else if (value==4) {
      lcd_gotoxy(3, 1);
      lcd_putc(0xFF);
  }
  else if (value==5) {
      lcd_gotoxy(4, 1);
      lcd_putc(0xFF);
  }
  else if (value==6) {
      lcd_gotoxy(5, 1);
      lcd_putc(0xFF);
  }
  else if (value==7) {
      lcd_gotoxy(6, 1);
      lcd_putc(0xFF);
  }
  else if (value==8) {
      lcd_gotoxy(7, 1);
      lcd_putc(0xFF);
  }
  else if (value==9) {
      lcd_gotoxy(8, 1);
      lcd_putc(0xFF);
  }
  else if (value==10) {
      lcd_gotoxy(9, 1);
      lcd_putc(0xFF);
  }*/

}

/* END OF FILE
*************************************************************/
```