```
//  File:     LabFA.c
//  Author:   Bradley Anderson
//  Created:  Dec-6, 2017
//
//  Collaboration: Kenzie Brian


#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h> // sprintf
#include "LabFA.h"
#include "hd44780.h"
#include "button7segFunctions.h"
#include "encoderFunctions.h"
#include "lm73_functions.h"
#include "twi_master.h"
#include "uart_functions.h"
#include "si4734.h"

bool a = TRUE;
bool b = FALSE;

//enum states {DISP_TIME, SET_TIME, ALARM, SNOOZE, SET_ALARM};
volatile enum states STATE = DISP_TIME;

// Variables for ADC
uint8_t  i;              //dummy variable
uint16_t adc_result;     //holds ADC result

// TWI interface buffers
extern uint8_t lm73_wr_buf[2];
extern uint8_t lm73_rd_buf[2];
const uint8_t lm73_address_local = 0b10010000;     // Model 0, pin floating

// Clock hour, minute, second
volatile uint8_t clock_s=0;
volatile uint8_t clock_m=1;
volatile uint8_t clock_h=12;
volatile uint8_t alarm_s=0;
volatile uint8_t alarm_m=0;
volatile uint8_t alarm_h=12;
volatile uint8_t snuze_s=0;
volatile uint8_t snuze_m=0;
volatile uint8_t snuze_h=12;
uint8_t hours12_24 = 12;
uint8_t am_pm = 0;
uint8_t alarmBeep=0;
uint8_t timeToCheckForRX = 1;
uint8_t timeToSPI = 0;
uint8_t timeTo7Seg = 0;

// Text to be displayed to LCD
volatile char *lcdText1 = "Welcome";
volatile char *lcdText2 = "Welcome";
volatile char *lcdTextTemp = "Temperature";
volatile char *lcdTextVolume = "Volume";
volatile char *uartString = "UART";
volatile uint8_t volume = 100;
```



```
// Number displayed to 7seg
volatile uint16_t segNum = 0;

// Number displayed to bargraph
volatile uint8_t barNum = 0;

// Radio variables
volatile enum radio_band current_radio_band = FM;
extern volatile uint8_t STC_interrupt;
volatile uint8_t freqTime=0;
volatile uint8_t needToChangeStation = 0;

uint16_t eeprom_fm_freq;
uint16_t eeprom_am_freq;
uint16_t eeprom_sw_freq;
uint8_t  eeprom_volume;

uint16_t current_fm_freq = 10630;
uint16_t current_am_freq;
uint16_t current_sw_freq;
uint8_t  current_volume;

// Function prototypes
void spiTxRx();
void spi_init();
void timer_init();
void digit_init();
void update7Seg();
void stateSwitcher();



//*********************************************************************************
//      -- Serial Peripheral Interface Initialization --
//  Modified from Roger Traylor's source file
//*********************************************************************************
void spi_init(void) {

    // -- LCD INIT --
    /* Run this code before attempting to write to the LCD.*/
    DDRF  |= 0x08;  //port F bit 3 is enable for LCD
    PORTF &= 0xF7;  //port F bit 3 is initially low

    DDRB  |= 0x07;  //Turn on SS, MOSI, SCLK
    PORTB |= _BV(PB1);  //port B initialization for SPI, SS_n off
    //see: /$install_path/avr/include/avr/iom128.h for bit definitions

    //Master mode, Clock=clk/4, Cycle half phase, Low polarity, MSB first
    SPCR=(1<<SPE) | (1<<MSTR); //enable SPI, clk low initially, rising edge samp
le
    SPSR=(1<<SPI2X);           //SPI at 2x speed (8 MHz)

    // -- Bargraph INIT --
    // Direction Registers
    DDRB |= (1<<RCLK) | (1<<SCLK) | (1<<MOSI) | (1<<PWM_BRT);
    DDRD |= (1<<SHLD_ENC) | (1<<OE_N_BG);

    // SPI Control Register
    SPCR |= (1<<SPE) | (1<<MSTR) | (0<<SPR1)|(1<<SPR0);

    // SPI Status Register
    SPSR |= (1<<SPI2X);
```

```
    // SPI Data Register
    PORTB &= ~(1<<OE_N_BG);
}



//*****************************************************************************
//      -- Transmits and Receives to/from SPI --
//*****************************************************************************
void spiTxRx() {
    static uint8_t textLine = 0;

    //clear_display();
    if (textLine) {
        cursor_home();
        string2lcd((char *)lcdText1);  //write upper half
        textLine = 0;
    } else {
        home_line2();
        string2lcd((char *)lcdText2);  //write upper half
        textLine = 1;
    }
    //_delay_us(500);

    // Toggle Encoder Shift/Load
    PORTD &= ~(1<<SHLD_ENC);
    PORTD |= (1<<SHLD_ENC);

    // SPI write from global variable
    SPDR = barNum;

    // Wait for 8 clock cycles
    while(bit_is_clear(SPSR, SPIF)) {}

    // Save the most recent serial reading into global variable
    encoderState = SPDR;

    // Toggle Bargraph Register Clock
    PORTB |= (1<<RCLK);
    PORTB &= ~(1<<RCLK);
}



//*****************************************************************************
//      -- Timer 0 Compare Interrupt --i
// Using the internal 32.768 KHz oscillator to implement a clock
//*****************************************************************************
ISR(TIMER0_OVF_vect) {
    static uint8_t clock=0;
    clock++;

    if (clock == 128){
                timeToCheckForRX = 1;
        clock_s++;
                freqTime++;
        clock = 0;
    }
    if (clock_s >= 60) {
        clock_s = 0;
        clock_m++;
```



```
    }
    if (clock_m >= 60) {
        clock_m = 0;
        clock_h++;
    }
    if (clock_h > hours12_24) {
        clock_h -= hours12_24;
        if (am_pm)
            am_pm = 0;
        else
            am_pm = 1;
    }

    if (clock % 32 == 0){
        alarmBeep = !alarmBeep;
    }
}//ISR TIMER0_OVF_vect



//*****************************************************************************
//      -- Timer 1 Compare Interrupt: Alarm signal generation --
//*****************************************************************************
ISR(TIMER1_COMPA_vect) {
        PORTD ^= (1<<D_BP);
}//ISR



//*****************************************************************************
//      -- Timer 2 Compare Interrupt: 7 Segment Brightness PWM --
//*****************************************************************************
// NO ISR NEEDED; PWM GOES STRAIGHT TO OUTPUT PIN



//*****************************************************************************
//      -- Timer 3 Compare Interrupt: Audio Amp Volume to DAC --
//      -- Also use this as a slower interrupt for various functionality --
//*****************************************************************************
ISR(TIMER3_OVF_vect) {
        //static char buffer[17];

        // -- 7 SEG BRIGHTNESS --
        // Read the value of the photo resistor
        readADC();
        // Set the brightness of the LCD
        OCR2 = 255- (adc_result)/4;
}



//*****************************************************************************
//      -- TIMER Initialization --
//*****************************************************************************
void timer_init() {

    // Timer counter 0 setup, running off i/o clock

    // Asynchronous Status Register, pg107
    //    Run off of external clock
    ASSR  |= (1<<AS0);
```

```
    // Timer/Counter Interrupt Mask, pg109
    //    Timer 0: overflow interrupt enable
    TIMSK |= (1<<TOIE0) | (1<<OCIE1A) | (1<<OCIE3A);
    ETIMSK |= (1<<TOIE3);

    // Timer/Counter Control Register, pg104
    // Timer 0: 32kHz osc. for internal clock
    //      Normal mode, OC0 disconnected, clkTOS with no prescalar
    TCCR0 = (1<<CS00);

    // Timer 1 init
    // Clear on compare, prescale and value set to give ~5kHz signal
    TCCR1B |= (1<<WGM12) | (1<<CS11) | (1<<CS10);
    OCR1A = 0X00EF;

    // Timer 2: Phase-corrected PWM for 7 seg brightness, no prescale
    TCCR2 |= (1<<WGM20) | (1<<COM21) | (1<<COM20) | (1<<CS20);

    // Timer 3: PWM for audio volume, but also correct speed for SPI reading
    // CTC mode, Clear on match
    DDRE = 0xFF;
    PORTE = 0xFF;
    TCCR3A = (1<<WGM31)|(0<<WGM30) | (1<<COM3A1)|(0<<COM3A0);
    TCCR3B = (0<<WGM33)|(1<<WGM32) | (1<<CS32)|(0<<CS31)|(0<<CS30);
    OCR3A = 70;

}



//*****************************************************************************
//      -- Initializes the analog->digital converter --
//*****************************************************************************
void adc_init(uint8_t pin) {

    //Initialize ADC and its ports
    DDRF  &= ~(_BV(pin)); //make port F bit 7 is ADC input
    PORTF &= ~(_BV(pin));  //port F bit 7 pull-ups must be off

    //single-ended, input PORTF bit 7, right adjusted, 10 bits
    // ADC Multiplexer Selection Register
    // Reference Selection = 01: Internal VRef
    // MUX = 00111: ADC7
    ADMUX |= (1<<REFS0) | (1<<MUX2)|(1<<MUX1)|(1<<MUX0);

    //ADC enabled, don't start yet, single shot mode
    // factor is 128 (125khz)
    // ADC Control and Status Register A, ADC Enable, ADC Prescalar Selection =
128;
    ADCSRA |= (1<<ADEN) | (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}



//*****************************************************************************
//      -- Update 7 seg with the most recent data --
//*****************************************************************************
void update7Seg() {

    //make PORTA an output
    DDRA = 0xFF;
```



```
    //bound a counter (0-4) to keep track of digit to display
    for (i=0; i<5; i++)
    {
        // Clear digit select
        PORTB &= SELCL;

        //update digit to display
        PORTB |= digitSelect[i];

        //send 7 segment code to LED segments
        PORTA = segment_data[i];


        //dimming/flicker correction
        //_delay_ms(10);
        _delay_us(600);
    }//for

    PORTB &= SELCL;
    PORTB |= SELBN;
    PORTB |= digitSelect[2];
    DDRA = 0x00;
} // end update7seg



//*****************************************************************************
//      -- Read ADC
//*****************************************************************************
void readADC() {


    //poke ADSC and start conversion
    // ADC Control and Status Register A, ADC Start Conversion
    ADCSRA |= (1<<ADSC);

    //spin while interrupt flag not set
    // ADC Control and Status Register A, ADC Interrupt Flag
    while(bit_is_clear(ADCSRA, ADIF)) {}

    //its done, clear flag by writing a one
    // ADC Interrupt Flag
    ADCSRA |= (1<<ADIF);

    //read the ADC output as 16 bits
    adc_result = ADC;
} // end readADC



//*****************************************************************************
//      -- Handles State Logic
//*****************************************************************************
void stateSwitcher() {
    //volatile static char buffer[4] = "    ";
    //uint8_t temp = volume;

    switch (buttonState){
        case 0b01:
            STATE = SET_TIME;
            break;
```

```
        case 0b10:
            STATE = SET_ALARM;
            break;
        case 0b100:
            if ((STATE == SNOOZE) || (STATE == ALARM)) {
                STATE = DISP_TIME;
                buttonState = 0;
                snuze_s = alarm_s;
                snuze_m = alarm_m;
                snuze_h = alarm_h;
            } else
                STATE = ALARM;
            break;
        default:
            if (STATE == ALARM) {
                if (buttonState != 0)
                    STATE = SNOOZE;
                snuze_s = clock_s + 10;
                snuze_m = clock_m;
                snuze_h = clock_h;
            } else {
                if (STATE != SNOOZE)
                    STATE = DISP_TIME;
                buttonState = 0;
            }
            break;
    }

    switch (STATE) {
        case DISP_TIME: // Display Time
            lcdText1 = "Signal      ";
            lcdText2 = lcdTextTemp;
                        //lcdText2 = "hello           ";
            //strcat(lcdText2, itoa(temp, buffer, 10));
            //lcdText2[11] = 0;
            break;
        case SET_TIME:
            lcdText1 = "Use dials to   ";
            lcdText2 = "change time    ";
            break;
        case ALARM:
            lcdText1 = "    ALARM!    ";
            lcdText2 = "  (Snooze?)   ";
            buttonState = 0;
            break;
        case SET_ALARM:
            lcdText1 = "Use dials to   ";
            lcdText2 = "change alarm   ";
            break;
        case SNOOZE:
            lcdText1 = "Snoozing.....";
            lcdText2 = "zzZzzZZZzzzZzZZz";
            break;
        default:
            lcdText1 = "   Welcome    ";
            lcdText2 = " (State error) ";
            break;
    } // end switch
} // end stateSwitcher
```



```
//*****************************************************************************
//      -- Transmits and Receives to/from SPI --
//*****************************************************************************
void twiRx() {

        uint16_t lm73_temp_local = 0; //  local temperature
        //char tempCharRemote[3] = "??"; //  local temperature
        char tempCharLocal[3] = "??";
        static char buffer[17];

        if (timeToCheckForRX) {
                timeToCheckForRX = 0;


                // -- TWI Read --
                twi_start_rd(lm73_address_local, lm73_rd_buf, 2); //read tempera
ture data from LM73 (2 bytes)
                _delay_ms(2);    //wait for it to finish
                lm73_temp_local = lm73_rd_buf[0]; //save high temperature byte i
nto lm73_temp
                lm73_temp_local = lm73_temp_local << 8; //shift it into upper by
te
                lm73_temp_local |= lm73_rd_buf[1]; //"OR" in the low temp byte t
o lm73_temp
                lm73_temp_local = lm73_temp_convert(tempCharLocal, lm73_temp_loc
al, 1); //convert to string in array with itoa() from avr-libc


                //uart_putc('t');
                //_delay_ms(30);    //wait for it to finish
                //tempCharRemote[0] = uart_getc();
                //tempCharRemote[0] = 'n';
                //_delay_ms(2);    //wait for it to finish
                //tempCharRemote[1] = uart_getc();
                //_delay_ms(2);    //wait for it to finish
                //tempCharRemote[2] = '\0';

                sprintf(buffer, "I=%d O=%s",lm73_temp_local, uartString);


                // -- VOLUME --
                OCR3A = (volume<<1);
                current_volume = volume;
                sprintf(buffer,"%s V=%d%%",lcdTextTemp, (int)(volume/2.50));
                //sprintf(buffer,tempCharRemote);
        }

        sprintf(buffer,"%-16s",buffer);
        lcdTextTemp = buffer;
}



//*****************************************************************************
//              -- Tells UART to start transmitting and then reads two character
s --
//*****************************************************************************
void uartTxRx() {
        static char buffer[16] = "no!";
        //static uint8_t count = 0;

        uart_putc('t');
```

```
        _delay_ms(3);    //wait for it to finish

        // When nothing to receive, uart_getc returns 0, which is also null char
        buffer[0] = uart_getc();
        buffer[1] = uart_getc();
        buffer[2] = 0;

        //sprintf(buffer,"%-16s",buffer);
        uartString = buffer;
}



//*****************************************************************************
//              -- Checks if alarm should be going off --
//*****************************************************************************
void checkIfAlarm() {
        if (STATE != SET_ALARM) {
                if ((alarm_s + 100*alarm_m + 10000*alarm_h == clock_s + 100*cloc
k_m + 10000*clock_h)
                || (snuze_s + 100*snuze_m + 10000*snuze_h == clock_s + 100*clock
_m + 10000*clock_h))
                STATE = ALARM;

        }

        if (alarm_m >= 60)
        alarm_m = 0;
        if (alarm_h >= 60)
        alarm_h = 0;
}



//*****************************************************************************
//              -- Reads global variables and converts them to 7seg data --
//*****************************************************************************
void diplayTimeToSegment() {
        // -- TIME DISPLAY --

        // Display the button latch state on the bargraph
        barNum = clock_s;

        if (freqTime < 4) {
                segNum = current_fm_freq/10;
                segsum(segNum);
                segment_data[1] &= 0b01111111;
        } else {
                // Convert minutes and hours to a number for displaying
                if (STATE == SET_ALARM)
                        segNum = alarm_m + 100*alarm_h;
                else
                        segNum = clock_m + 100*clock_h;


                // Update number to digitSelect[i]
                segsum(segNum);
                if (am_pm)
                        segment_data[2] &= ~(1<<COLON3);
                else
                        segment_data[2] |= (1<<COLON3);
```



```
                // colon blink
                switch (clock_s % 2) {
                        case 0:
                                if (STATE == ALARM)
                                        segment_data[2] |= (1<<COLON1) | (1<<COLO
N2);
                                break;
                        case 1:
                                segment_data[2] &= ~((1<<COLON1) | (1<<COLON2));
                                break;
                        default:
                                break;
                }
        }
}



//*****************************************************************************
//              -- Initializes the radio --
//*****************************************************************************
void radio_init() {

        DDRE = 0xFF;            //PORTE output, low
        PORTE = 0x00;

        //DDRE  |= (1<<RAD_RST);
        PORTE |= (1<<RAD_RST); //Radio reset is on at powerup (active high)
        //Enable interrupt for radio
        EICRB |= (1<<ISC71) | (0<<ISC70); //GPIO is pull-up; detect falling edge
        EIMSK |= (1<<INT7);

        // Turn off pull up resistor
        PORTE &= ~(1<<RAD_INT);
        DDRE |= (1<<RAD_INT);

        // Toggle reset pin for at least 100us
        //DDRE |= (1<<RAD_RST);
        PORTE |= (1<<RAD_RST);
        _delay_us(200);
        PORTE &= ~(1<<RAD_RST);
        _delay_us(50);

        // Set radio interrupt back to input
        DDRE &= ~(1<<RAD_INT);
        //PORTE |= (1<<RAD_INT); // pull up
}



//*****************************************************************************
//              -- Response from radio --
//*****************************************************************************
ISR(INT7_vect) {STC_interrupt = TRUE;}



//*****************************************************************************
//                                  main
//  Does main stuff
//*****************************************************************************
int main(void) {
```

```
    digit_init();
    timer_init();
    spi_init();
    lcd_init();
        init_twi();
    adc_init(CDS);
    clear_display();
        uart_init();
        radio_init();

        sei();



        //set LM73 mode for reading temperature by loading pointer register
        lm73_wr_buf[0] = 0x00; //load lm73_wr_buf[0] with temperature pointer ad
dress

        // Tell TWI to start writing, number of bytes = 2
        twi_start_wr(lm73_address_local, lm73_wr_buf, 2);

        // Wait for the transfer to finish
        _delay_ms(2);


        fm_pwr_up();
        current_fm_freq = 10630;
        set_property(0x4001,0x0000);
        fm_tune_freq();



    while(1) {

                if (needToChangeStation) {
                        needToChangeStation = 0;
                        fm_tune_freq();
                }
        // State Machine Control!
        stateSwitcher();

        checkIfAlarm();


        // -- READ BUTTONS --
        toggle_button_bus();

                spiTxRx();
                interpret_encoders();

                // -- UART --
                uartTxRx();

                // -- TWI --
                twiRx();

                diplayTimeToSegment();

                update7Seg();

        // Alarm Beeping
```



```
        if (STATE == ALARM) {
            if (alarmBeep)
                DDRD |= (1<<D_BP);
            else
                DDRD &= ~(1<<D_BP);
        } else
            DDRD &= ~(1<<D_BP);

    }//while

    return 0;

}//main
```