```
/*****************************************************************
 * File:        fsm_tasks.c
 * Author:      Bryant Gonzaga
 * Created:     4/12/2018
 * Modified:    4/12/2018
 *
 * Notes:
 *  Processor specific, libraries need
 *
 * Description:
 *  A full description of what can be found in this file
 *
 * How To:
 *  If necessary add some instructions on how to use the file.
 *****************************************************************/

#include "fsm_tasks.h"

/* Global Variables */
static unsigned char temp_time_data[7]; // temp data for time
static unsigned char temp_alarm_data[4]; // temp data for alarm

/* Global Variables */
unsigned char key_code;         // stores the last pressed key value
unsigned char rtc_time_data[7]; // stores full date and time
unsigned char rtc_alarm_data[4]; // stores full date and time for
alarm

extern state present_state;

static unsigned char time_input_state;
static unsigned char alarm_input_state;

//TODO: maybe find a better way to check if confirm key is a valid
input
unsigned char confirm_valid;

void alarm_input_init()
{
    alarm_input_state = 0;
    dsp_alarm_setter_fn();
}

void time_input_init()
{
    time_input_state = 0;
    dsp_time_setter_fn();
}

void alarm_input_handler_fn()
{
    /* Check Key is in range */
    if (key_code > 9) {
        return;
    }
```

```
    unsigned char temp = 0xFF;

    // TODO: only minumul checks are done. Create a more robust check
of imputs.

    /* Handle Key Input */
    switch (alarm_input_state) {
        case 0:     // Hours
            if (key_code < 3) {
                temp = 2;
            }
            break;
        case 1:
            if (key_code < 10) {
                temp = 2;
            }
            break;
        case 2:     // Minutes
            if (key_code < 6) {
                temp = 1;
            }
            break;
        case 3:
            if (key_code < 10) {
                temp = 1;
            }
            break;
        case 4:    // Seconds
            if (key_code < 6) {
                temp = 0;
            }
            break;
        case 5:
            if (key_code < 10) {
                temp = 0;
                confirm_valid = 1;
            }
            break;
        case 6:
            if (key_code < 4) {
                temp_alarm_data[key_code] |= _BV(7);
            }
        default:
            alarm_input_state = 0;
            break;
    }

    /* On Good Key Input */
    if (temp != 0xFF) {
        /* Update temp Time Array */
        if ( (alarm_input_state % 2) == 0) {
            temp_alarm_data[temp] = (key_code << 4);
        } else {
            temp_alarm_data[temp] |= (key_code & 0x0F);
        }
        /* Increment Handler State */
        alarm_input_state = (alarm_input_state + 1) % 7;
```

```
        /* Display Prompt */
        dsp_alarm_setter_fn();

        temp_alarm_data[3] |= _BV(7);
    }
}

void time_input_handler_fn()
{
    /* Check Key is in range */
    if (key_code > 9) {
        return;
    }

    unsigned char temp = 0xFF;

    // TODO: only minumul checks are done. Create a more robust check
of imputs.

    /* Handle Key Input */
    switch (time_input_state) {
        case 0:     // Month
            if (key_code < 2) {
                temp = 5;
            }
            break;
        case 1:
            if ( (temp_time_data[5] >> 4) == 0) {
                if (key_code != 0) {
                    temp = 5;
                }
            } else if ( (temp_time_data[5] >> 4) == 1) {
                if (key_code < 3) {
                    temp = 5;
                }
            }
            break;
        case 2:     // Day
            if (key_code < 4) {
                temp = 4;
            }
            break;
        case 3:
            temp = 4;
            break;
        case 4:     // Year
            temp = 6;
            break;
        case 5:
            temp = 6;
            break;
        case 6:     // Day of the week
            temp = 3;
            break;
        case 7:     // 12 or 24 hr choice
            if (key_code == 1) {
                temp = 12;
```

```
            } else if (key_code == 2) {
                temp = 24;
            }
            break;
        case 8:     // Hours
            if (key_code < 3) {
                temp = 2;
            }
            break;
        case 9:
                temp = 2;
            break;
        case 10:     // Minutes
            if (key_code < 6) {
                temp = 1;
            }
            break;
        case 11:
                temp = 1;
            break;
        case 12:    // Seconds
            if (key_code < 6) {
                temp = 0;
            }
            break;
        case 13:
            temp = 0;
            confirm_valid = 1;
            break;
        default:
            time_input_state = 0;
            break;
    }

    /* On Good Key Input */
    if (temp != 0xFF) {
        /* Update temp Time Array */
        if (time_input_state == 6) {

        } else if (time_input_state == 7) {

        } else if ( (time_input_state % 2) == 0) {
            temp_time_data[temp] = (key_code << 4);
        } else {
            temp_time_data[temp] |= (key_code & 0x0F);
        }
        /* Increment Handler State */
        time_input_state = (time_input_state + 1) % 15;
        /* Display Prompt */
        dsp_time_setter_fn();
    }
}

void dsp_alarm_setter_fn()
{
    /* Init LCD */
    init_lcd_dog();     // setup spi configuration
```

```
    clear_dsp();        // clear ram buffer

    /* save what is to be displayed */
    switch (alarm_input_state) {
        case 0:
        case 1:
            printf("Input hour:\n");
            printf("%d%d",
                ( (temp_alarm_data[2] >> 4) & 0x03 ) ,    // Hour
                (  temp_alarm_data[2]       & 0x0F )      // Hour
            );
            break;
        case 2:
        case 3:
            printf("Input minutes:\n");
            printf("%d%d",
                ( (temp_alarm_data[1] >> 4) & 0x0F ) ,    // Minute
                (  temp_alarm_data[1]       & 0x0F )      // Minute
            );
            break;
        case 4:
        case 5:
            printf("Input seconds:\n");
            printf("%d%d",
                ( (temp_alarm_data[0] >> 4) & 0x0F ) ,    // Second
                (  temp_alarm_data[0]       & 0x0F )      // Second
            );
            break;
        case 6:
            printf("Is this correct?");
            printf("Time: %d%d:%d%d:%d%d",
                ( (temp_alarm_data[2] >> 4) & 0x03 ) ,    // Hour
                (  temp_alarm_data[2]       & 0x0F ) ,    // Hour
                ( (temp_alarm_data[1] >> 4) & 0x0F ) ,    // Minute
                (  temp_alarm_data[1]       & 0x0F ) ,    // Minute
                ( (temp_alarm_data[0] >> 4) & 0x0F ) ,    // Second
                (  temp_alarm_data[0]       & 0x0F )      // Second
            );
            break;
        default:
            break;
    }
    /* Print Chosen Message */
    update_lcd_dog();
}

void dsp_time_setter_fn()
{
    /* Init LCD */
    init_lcd_dog();     // setup spi configuration
    clear_dsp();        // clear ram buffer

    /* save what is to be displayed */
    switch (time_input_state) {
        case 0:
        case 1:
            printf("Input month:\n");
```

```
    printf("%d%d",
        ( (temp_time_data[5] >> 4) & 0x03 ) ,    // Month
        (  temp_time_data[5]       & 0x0F )      // Month
    );
    break;
case 2:
case 3:
    printf("Input day:\n");
    printf("%d%d",
        ( (temp_time_data[4] >> 4) & 0x03 ) ,    // Day
        (  temp_time_data[4]       & 0x0F )      // Day
    );
    break;
case 4:
case 5:
    printf("Input year:\n");
    printf("%d%d",
        ( (temp_time_data[6] >> 4) & 0x0F ) ,    // Year
        (  temp_time_data[6]       & 0x0F )      // Year
    );
    break;
case 6:
    printf("Select day:");                    // Day of week
    break;
case 7:
    printf("Select mode:\n");
    printf("1: 12 hr\n2: 24 hr");
    break;
case 8:
case 9:
    printf("Input hour:\n");
    printf("%d%d",
        ( (temp_time_data[2] >> 4) & 0x03 ) ,    // Hour
        (  temp_time_data[2]       & 0x0F )      // Hour
    );
    break;
case 10:
case 11:
    printf("Input minutes:\n");
    printf("%d%d",
        ( (temp_time_data[1] >> 4) & 0x0F ) ,    // Minute
        (  temp_time_data[1]       & 0x0F )      // Minute
    );
    break;
case 12:
case 13:
    printf("Input seconds:\n");
    printf("%d%d",
        ( (temp_time_data[0] >> 4) & 0x0F ) ,    // Second
        (  temp_time_data[0]       & 0x0F )      // Second
    );
    break;
case 14:
    printf("Is this correct?");
    printf("Date: %d%d/%d%d/%d%d\n",
        ( (temp_time_data[5] >> 4) & 0x03 ) ,    // Month
        (  temp_time_data[5]       & 0x0F ) ,    // Month
```

```
                ( (temp_time_data[4] >> 4) & 0x03 ) ,    // Day
                (  temp_time_data[4]       & 0x0F ) ,    // Day
                ( (temp_time_data[6] >> 4) & 0x0F ) ,    // Year
                (  temp_time_data[6]       & 0x0F )      // Year
            );
            printf("Time: %d%d:%d%d:%d%d",
                ( (temp_time_data[2] >> 4) & 0x03 ) ,    // Hour
                (  temp_time_data[2]       & 0x0F ) ,    // Hour
                ( (temp_time_data[1] >> 4) & 0x0F ) ,    // Minute
                (  temp_time_data[1]       & 0x0F ) ,    // Minute
                ( (temp_time_data[0] >> 4) & 0x0F ) ,    // Second
                (  temp_time_data[0]       & 0x0F )      // Second
            );
            break;
        default:
            break;
    }
    /* Print Chosen Message */
    update_lcd_dog();
}

void confirm_alarm_fn()
{
        if (confirm_valid) {
        /* Transfer Temp Data to Permenant Array */
        for (unsigned char i = 0; i < 4; i++) {
            rtc_alarm_data[i] = temp_alarm_data[i];
        }
        /* Initialize ds1306 IC */
        spi_rtc_ds1306_config();
        /* unlock mem */
        unlock_rtc();
        /* Send Time to DS1306 */
        block_write_rtc(rtc_alarm_data, 0x87, 4);
        /* lock mem */
        lock_rtc();
        /* confirm no longer valid */
        confirm_valid = 0;
        /* display data */
        dsp_all_fn();
    }

}

void confirm_time_fn()
{
    if (confirm_valid) {
        /* Transfer Temp Data to Permenant Array */
        for (unsigned char i = 0; i < 7; i++) {
            rtc_time_data[i] = temp_time_data[i];
        }
        /* Initialize ds1306 IC */
        spi_rtc_ds1306_config();
        /* unlock mem */
        unlock_rtc();
        /* Send Time to DS1306 */
        block_write_rtc(rtc_time_data, 0x80, 7);
```

```
        /* lock mem */
        lock_rtc();
        /* confirm no longer valid */
        confirm_valid = 0;
        /* display data */
        dsp_all_fn();
    }
}

void dsp_all_fn()
{
     /* Get Data */
    block_read_rtc(rtc_time_data, 0x00, 7); // get the time data from
rtc

    /* Init LCD */
    init_lcd_dog();     // setup spi configuration
    clear_dsp();        // clear ram buffer

    /* save what is to be displayed */
    printf("Date: %d%d/%d%d/%d%d\n",
            ( (rtc_time_data[5] >> 4) & 0x03 ) ,    // Month
            (  rtc_time_data[5]       & 0x0F ) ,    // Month
            ( (rtc_time_data[4] >> 4) & 0x03 ) ,    // Day
            (  rtc_time_data[4]       & 0x0F ) ,    // Day
            ( (rtc_time_data[6] >> 4) & 0x0F ) ,    // Year
            (  rtc_time_data[6]       & 0x0F )      // Year
    );

    printf("Time: %d%d:%d%d:%d%d\n",
            ( (rtc_time_data[2] >> 4) & 0x03 ) ,    // Hour
            (  rtc_time_data[2]       & 0x0F ) ,    // Hour
            ( (rtc_time_data[1] >> 4) & 0x0F ) ,    // Minute
            (  rtc_time_data[1]       & 0x0F ) ,    // Minute
            ( (rtc_time_data[0] >> 4) & 0x0F ) ,    // Second
            (  rtc_time_data[0]       & 0x0F )      // Second
    );

    // TODO: Check for 12 HR or 24 HR setting

    printf("Alarm 0:");

    /* Actually send data */
    update_lcd_dog();
}

void error_fn()
{
}

void nop_fn()           // not sure if needed yet
{

}
```