5:53 AM 18-Dec-21/\*

\* File: main.c

\* Author: moaz said

\*

\*/

#include "../MCAL/Registers.h"

#include "../ServiceLayer/std\_types.h"

#include "../ServiceLayer/Config.h"

#include "../HAL/7SEG/7seg.h"

#include "../MCAL/DIO/DIO.h"

#include "../HAL/7SEG/unitTest/7seg\_test.h"

#include "../HAL/BTN/unit test/btn\_test.h"

#include "../HAL/BTN/btn.h"

#include "../HAL/eeprom\_ext/eeprom\_ext.h"

#include "../MCAL/ADC/adc.h"

#include "../MCAL/ISR/isr.h"

#include "../MCAL/TIMER/timer.h"

#include "../ServiceLayer/SOS/SOS.h"

#include "../MCAL/I2C/i2c.h"

#include <xc.h>

#include <pic16f877a.h>

#define LEDPIN 0

#define ON\_OFF\_BTN 0

#define UP\_BTN 1

#define DOWN\_BTN 2

#define USED\_PATTERN (0x3D)

#define TEMP\_VLAUES 10

#define SSD\_FLASHING\_CYCLE 20

#define TEMP\_STEP 5

#define MAX\_TEMP (75)

#define MIN\_TEMP (35)

typedef enum{TEMP\_SET\_MODE,NORM\_MODE}operationMode\_t;

static uint8\_t tempValues[TEMP\_VLAUES];

static uint8\_t currentTemp;

static operationMode\_t gu8\_mode;

static uint8\_t gu8\_settedTemp;

static uint8\_t gu8\_TempControlFlag;

static uint8\_t gu8\_tempToAchive;

static uint8\_t gu8\_deviceDisableFlag;

void tempControlTask()

{

/\*\*

\* if the desired temp have been set

\* get average temp from last 10 readings

\* control the cooler/heater element based on themp

\*/

if(gu8\_TempControlFlag)

{

static uint8\_t u8\_counter = 0;

static uint8\_t u8\_heaterOn = FALSE;

static uint8\_t u8\_coolerOn = FALSE;

uint8\_t u8\_i = 0;

uint16\_t u8\_averageTempValue = 0;

/\*\*

\* display the average of the avaliable readins

\* before getting ten readings the average = average of the last x reading

\* where x is <=10

\*/

for(; u8\_i< TEMP\_VLAUES && tempValues[u8\_i] ;u8\_i++)

u8\_averageTempValue += tempValues[u8\_i];

u8\_averageTempValue/=u8\_i;

/\*\*

\* if both heater and cooler off

\* if temp blow critical turn on heater

\* else \_temp above critical\_ turn on cooler

\* else if heater on and temp above critical

\* turn heater off and turn cooler on and turn led on

\* else if cooler on and temp below critical

\* turn cooler off and turn heater on

\*/

if(u8\_heaterOn == FALSE && u8\_coolerOn == FALSE)

{

if(u8\_averageTempValue <= gu8\_tempToAchive-TEMP\_STEP)

{

DIO\_Write(HEATER\_ELEMENT,HIGHL);///turn on the heater

u8\_heaterOn = TRUE;

}

else

{

DIO\_Write(COOLER\_ELEMENT,HIGHL);///turn on the cooler

u8\_coolerOn = TRUE;

}

}

else if(u8\_heaterOn && u8\_averageTempValue >= gu8\_tempToAchive+TEMP\_STEP)

{

//turn off heater and turn on cooler

DIO\_Write(HEATER\_ELEMENT,LOWL);

DIO\_Write(COOLER\_ELEMENT,HIGHL);

DIO\_Write(TEMP\_CONTROL\_LED,HIGHL);

u8\_heaterOn = FALSE;

u8\_coolerOn = TRUE;

}

else if(u8\_coolerOn && u8\_averageTempValue <= gu8\_tempToAchive-TEMP\_STEP)

{

//turn on heater and turn off cooler

DIO\_Write(HEATER\_ELEMENT,HIGHL);

DIO\_Write(COOLER\_ELEMENT,LOWL);

u8\_heaterOn = TRUE;

u8\_coolerOn = FALSE;

}

/\*\*

\* if the heater on toggle the led state every 1000 ms

\* we enter the function every 100ms so the counter count to 10 times befor

\* toggle

\*/

if(u8\_heaterOn)

{

u8\_counter++;

if(u8\_counter == 10)

{

DIO\_Toggle(TEMP\_CONTROL\_LED);

u8\_counter = 0;

}

}

}

}

void buttonTask()

{

BtnStateType u8\_onOffBtnStatus;

BtnStateType u8\_upBtnStatus;

BtnStateType u8\_downBtnStatus;

BTN\_Manager();

BTN\_GetState(&u8\_onOffBtnStatus,ON\_OFF\_BTN);

BTN\_GetState(&u8\_upBtnStatus,UP\_BTN);

BTN\_GetState(&u8\_downBtnStatus,DOWN\_BTN);

/\*\*

\* if on/off button pressed rise disable system flag

\* we check on this flag on the main loop and disable the system

\* if the flag is up

\*/

if(u8\_onOffBtnStatus == BUT\_ON)

gu8\_deviceDisableFlag = !gu8\_deviceDisableFlag;

/\*\*

\* functionality of the button depend on the current system mode {NORM\_MODE,TEMP\_SET\_MODE}

\* in normal mode: if the up or down buttons pressed

\* get temp from the external e2Prom

\* change mode to TEMP\_SET\_MIDE

\* in TEMP\_SET\_MODE: if up or down buttons pressed

\* change temp by 5 with max 75 and min 35 degree

\*

\*/

if(gu8\_mode == NORM\_MODE)

{

if(u8\_upBtnStatus == BUT\_ON || u8\_downBtnStatus == BUT\_ON)

{

gu8\_mode = TEMP\_SET\_MODE;

gu8\_settedTemp = e2pext\_r(0x00);

}

}

else if(gu8\_mode == TEMP\_SET\_MODE)

{

static uint8\_t u8\_btnPressCntr = 0;

if(u8\_upBtnStatus == BUT\_ON)

{

gu8\_settedTemp+=(gu8\_settedTemp == MAX\_TEMP)? 0 :TEMP\_STEP;

}

else if(u8\_downBtnStatus == BUT\_ON)

{

gu8\_settedTemp-=(gu8\_settedTemp == MIN\_TEMP)? 0 :TEMP\_STEP;

}

else if(u8\_downBtnStatus == BUT\_OFF || u8\_upBtnStatus == BUT\_OFF)

{

/\*

\* count the time no button has been pressed to implement a time out

\* to exit TEMP\_SET\_MODE

\*/

u8\_btnPressCntr++;

}

/\*

\* if either up or down buttons pressed reset timeout counter

\*/

if(u8\_downBtnStatus == BUT\_ON || u8\_upBtnStatus == BUT\_ON

|| u8\_upBtnStatus == BUT\_PRSSED || u8\_downBtnStatus == BUT\_PRSSED)

u8\_btnPressCntr = 0;

/\*\*

\* if time out counter reached time out counts \_button task called every 50ms

\* so we need 50000/50 = 100 enterance befor change the state to normal state\_

\*

\* load setted value to e2prom

\* set temp for temp control task

\* fire temp control task flag

\* reset timeout counter

\* change state to normal state

\*/

if(u8\_btnPressCntr >= 100)

{

e2pext\_w(0x00,gu8\_settedTemp);

gu8\_tempToAchive = gu8\_settedTemp;

gu8\_TempControlFlag = TRUE;

u8\_btnPressCntr = 0;

gu8\_mode = NORM\_MODE;

}

}

}

void sevenSegTask()

{

/\*\*

\* two modes for lcd {NORMAL MODE , TEMP SET MODE}

\* if NORMAL\_MODE:

\* display current temp to the SSD

\* else:

\* blink SSD every 1 sec

\* display the stored temp in EXT\_EEPROM

\* update temp based on user input up-down

\*

\*/

if(gu8\_mode == NORM\_MODE)

{

static uint8\_t u8\_selectedSevenSeg = SEVEN\_SEG\_TWO;

if(u8\_selectedSevenSeg == SEVEN\_SEG\_TWO )

sevenSegSendChar(currentTemp%10,SEVEN\_SEG\_TWO);

else

sevenSegSendChar(currentTemp/10,SEVEN\_SEG\_ONE);

/\*

\* update u8\_selectedSevenSeg variable to alternate between the two seven seg.

\*/

u8\_selectedSevenSeg = (u8\_selectedSevenSeg == SEVEN\_SEG\_ONE)? SEVEN\_SEG\_TWO : SEVEN\_SEG\_ONE;

}

else if( gu8\_mode == TEMP\_SET\_MODE)

{

static uint8\_t u8\_ssdDisableFalg = FALSE;

static uint8\_t u8\_flashCounter = 0;

static uint8\_t u8\_selectedSevenSeg = SEVEN\_SEG\_TWO;

if(u8\_selectedSevenSeg == SEVEN\_SEG\_TWO)

sevenSegSendChar(gu8\_settedTemp%10,SEVEN\_SEG\_TWO);

else

sevenSegSendChar(gu8\_settedTemp/10,SEVEN\_SEG\_ONE);

/\*

\* update u8\_selectedSevenSeg variable to alternate between the two seven seg.

\*/

u8\_selectedSevenSeg = (u8\_selectedSevenSeg == SEVEN\_SEG\_ONE)? SEVEN\_SEG\_TWO : SEVEN\_SEG\_ONE;

/\*\*

\* control the ssd on/off time as 1 sec for each

\*/

if(u8\_flashCounter == SSD\_FLASHING\_CYCLE)

{

u8\_ssdDisableFalg = !u8\_ssdDisableFalg;

u8\_flashCounter = 0 ;

}

if(u8\_ssdDisableFalg)

disableSevenSeg();

u8\_flashCounter++;

}

}

void tempTask()

{

static uint8\_t u8\_TempPos = 0;

float64\_t clsius;

uint16\_t u16\_ADC\_value = 0;

/\*\*

\* trigger ADC on specific chanal

\* get temp from adc value

\* set adc value in it's pos in temp array

\*/

ADC\_trigger(TEMP\_ADC\_CHNL);

getADC\_value(&u16\_ADC\_value);

//this eq from the data sheet of the LM25 module

clsius = (((float64\_t)u16\_ADC\_value\*4.88)/(10.00));

currentTemp = (uint32\_t)clsius; //this is for SSD

tempValues[u8\_TempPos] = (uint32\_t)clsius; //this is for temp control task

u8\_TempPos++;

if(u8\_TempPos == TEMP\_VLAUES) //reach array boundry

u8\_TempPos = 0;

}

void appInit()

{

///init temp control element to zero

/\*\*

\* call MCAL functions from application as it is necessary to

\* initialize the heater/cooler and led

\* and there is no module implemented for them case their functionality

\* are very simple and don't need to be structured necessary overhead\_

\*/

DIO\_Init();

DIO\_Write(HEATER\_ELEMENT,LOWL);

DIO\_Write(COOLER\_ELEMENT,LOWL);

//init temp control LED AS 0

DIO\_Write(TEMP\_CONTROL\_LED,LOWL);

BTN\_Init();

interruptsInit();

sevenSegInit();

disableSevenSeg();

ADC\_Init();

i2c\_init();

SOS\_Init();

uint8\_t u8\_i = 0;

for(; u8\_i< TEMP\_VLAUES ;u8\_i++)

tempValues[u8\_i] = 0;

/\*\*

\* this block of code used to set the temp to 60

\* in the first run

\* this can be simplified by checking for 0xff pattern

\* but this is work for now

\*/

uint8\_t u8\_usedPattern = e2pext\_r(0x02);

if(u8\_usedPattern != USED\_PATTERN)

{

e2pext\_w(0x00,60);

e2pext\_w(0x02,USED\_PATTERN);

}

///set initial values for global variables

gu8\_mode = NORM\_MODE;

currentTemp = 0;

gu8\_settedTemp = 60;

gu8\_TempControlFlag = 0 ;

gu8\_deviceDisableFlag = TRUE;

}

void checkONBtnStatus()

{

/\*

\* this function used to check on the on/off btn

\* if the system on the off state

\*/

BtnStateType u8\_onOffBtnStatus;

BTN\_Manager();

BTN\_GetState(&u8\_onOffBtnStatus,ON\_OFF\_BTN);

if(u8\_onOffBtnStatus == BUT\_ON)

gu8\_deviceDisableFlag = !gu8\_deviceDisableFlag;

}

void OS\_startFunction()

{

tempTask();

}

void main(void){

appInit();

/\*\*

\* task priority is set by thier order of call not thier

\* priority number as the priority feature is off

\* from the small os.

\*/

SOS\_createTask(1,tempTask,100,PERIODIC,1);

SOS\_createTask(3,buttonTask,50,PERIODIC,2);

SOS\_createTask(4,tempControlTask,100,PERIODIC,3);

SOS\_createTask(2,sevenSegTask,50,PERIODIC,4);

//set the CBF to call at the start of the os

SOS\_StartProc(OS\_startFunction);

while(1)

{

/\*\*

\* if the on off btn is pressed and the system is on disable system

\* disable system by call init function of the system

\*/

if(gu8\_deviceDisableFlag == TRUE)

{

appInit();

/\*\*

\* halt system and only check for the

\* on btn

\*/

while(gu8\_deviceDisableFlag == TRUE)

{

checkONBtnStatus();

}

}

SOS\_run();

}

return;

}