Table of Contents

Abstract :

in this project we will be controlling the temperature of an oven by using the embedded system and the free Rtos and the LCD and GPIO Libraries as well as the UART buffer

Our Temperature sensor is The LM35 temperature sensor but we will use a potentiometer instead because the LM35 doesn’t give accurate temperature and it is hard using it and getting expected setpoint

We will also be using a Buzzer as an alarm in order to be informed if the temperature increases more than a certain value for safety measures

We will be using a Relay as a switch to turn on and off the heater

And finally we will be using the LCD in order to see the setpoint and the current value of the temperature

Here Are the connection of each components :

PF1 buzzer
PE3 temperature potentiometer
PF3 relay
______
LCD:
PC4 D4
PC5 D5
PC6 D6
PC7 D7
PA3 R/W pin
PA2 Enable
PA4 R/S pin
Potentiometer V0 pin
3.3v Anode
Gnd Cathode

Main.c :

Description :

In the main we identify the task that we want to use in the project , there are four task in this project which are the Main task that controls the oven temperature. This task reads the analog temperature from the LM35 temperature sensor chip which is connected to PE5 of the microcontroller. the alarm value is set to 70°C. New setpoint values are received through the xUARTQueue from Uart Task. The relay at port pin PF3 is configured as an output and also the LED at port pin PF2 is configured as an output.

Code :

```
#include "Trial2.h"

int main()
{

PROJECT_Init();
xTaskCreate(Main_Task,"Main Controller",400, NULL,7, NULL);
xTaskCreate(UART_Task,"UART Controller",400, NULL,7, NULL);
xTaskCreate(LCD_Task,"LCD Controller",400, NULL,7, NULL);
xTaskCreate(Buzzer_Task,"Buzzer Controller",400, NULL,7, NULL);

vTaskStartScheduler();// Start the RTOS scheduler

while(1);
return 0;
}
```

Trial2.c :

Description :

Here we have the functions that are called in the main and the initialization of the tiva

1. Initialization :

Description :

We initialize PortE and PortF and the clock of the ports and the UART along with the transmitter and the receiver and the buffer of the UART

Code :

```
void PORTE_init(void)
{

  SYSCTL_RCGCGPIO_R |= 0x00000016;                  //enables clock to PE
  //while((SYSCTL_PRGPIO_R&0x00000016) == 0)           //Wait for clock stability
  //{};
  GPIO_PORTE_CR_R |= 0x37;                           //Determine that we are using  pins
0,1,2,4, and 5
  GPIO_PORTE_DEN_R |= 0x37;                          //Digital Enable pins 0,1,2,4, and 5
  GPIO_PORTE_DIR_R |= 0x37;                         //Set pins 0,1,2,4,5 as outputs      //0111
111

}

void PORTF_init(void)
{

SYSCTL_RCGCGPIO_R |= 0x00000020;                    //enables clock to PF
//  while((SYSCTL_PRGPIO_R&0x00000016) == 0)            //Wait for clock stability {};
  GPIO_PORTF_LOCK_R = 0x4C4F434B;                   //Unlock PF0
  GPIO_PORTF_CR_R  |= 0x0e;                          // Determine that we are using PF0 and
PF4
  GPIO_PORTF_DIR_R |= 0x0e;                          //Set pins 1,2,3 as outputs (LED pins)
  GPIO_PORTF_DEN_R |= 0x0e;                          //Digital Enable pins 1,2,3

}
```

```
/////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////
void UART_Init(void)//UART initialization
{
  SYSCTL_RCGCUART_R |=  0x1;     // enable clock for UART0
  SYSCTL_RCGCGPIO_R |=  0x1;     // enable clock for portA

  GPIO_PORTA_AFSEL_R = 0x3;         //Use PA0,PA1 alternate function
  GPIO_PORTA_PCTL_R = (1<<0)|(1<<4); // PA0 and PA1 configure for UART module
  GPIO_PORTA_DEN_R = 0x3; // Make PA0 and PA1 as digital

  UART0_CTL_R &= ~0x1; //disable uart module during configuration
  UART0_IBRD_R = 104; //16MHz/16=1MHz, 1MHz/104=9600 baud rate
  UART0_FBRD_R = 11; //fraction part of baud generator register
  UART0_LCRH_R = (0x3 << 5); //8-bit , no parity , 1 stop bit
  UART0_CC_R = 0x05; //use system clock
  UART0_CTL_R = (1<<0)|(1<<8)|(1<<9); // enable uart module
}
char UART0_Receiver(void) //Receives data entered by user
{
    char data;
  while((UART0->FR & (1<<4)) != 0); /* wait until Rx buffer is not full */
    data = UART0->DR ;            /* before giving it another byte */
    return (unsigned char) data;

}

void UART0_Transmitter(unsigned char data) //Transmits data
{
    while((UART0->FR & (1<<5)) != 0); /* wait until Tx buffer not full */
    UART0->DR = data;                /* before giving it another byte */
}

void printstring(char *str) //print data on PC
{
  while(*str)
{
          UART0_Transmitter(*(str++));
}
}
```

2. Main_Task :

Description :

this is the main task that controls the oven temperature , If the measured temperature is less than the setpoint then the relay is activated to turn ON the heater. If on the other hand the measured temperature is greater than the setpoint then the relay is de-activated to turn OFF the heater. The measured temperature is compared with the pre-defined alarm value and if it is higher than the alarm value, then a (1) is sent to Task 4 (Buzzer Controller) using the xBuzzerQueue so that Buzzer Task activates the buzzer. If the measured temperature is less than the alarm value, then a (0) is sent to Buzzer Task to stop the buzzer. The measured and the setpoint temperature values are sent to the LCD via the xLCDQueue so that they can be displayed on the LCD. A structure is created to store the measured and the setpoint temperature values in character arrays Txt1 an Txt2 respectively

Code :

```
void Main_Task(void *pvParameters) //Main Controller Task
{
//character arrays to store the measured and setpoint values
typedef struct Message
{
        unsigned char Txt1[4];
        unsigned char Txt2[4];
} AMessage;

AMessage msg;




unsigned char setpoint;   // the setpoint entered by user
unsigned char Temperature; //the actual measured temperature
float mV;
float adc_value;
unsigned char AlarmValue=70; //the alarm value which controls the buzzer
int state;
state=0;
```

```
int on=1;   //to indicate that buzzer should be turned ON
int off=0;  //to indicate that buzzer should be turned OFF

while(1)
 {
    xQueueReceive(xUARTQueue, &setpoint,0);  //Receive the setpoint entered by
user through uart

            ADC0->PSSI |= (1<<3);       /* Enable SS3 conversion or start
sampling data from AN0 */
  while((ADC0->RIS & 8) == 0) ;   /* Wait untill sample conversion completed*/
  adc_value = ADC0->SSFIFO3; /* read adc coversion result from SS3 FIFO*/
  ADC0->ISC = 8;         /* clear coversion clear flag bit*/
            mV= ( (adc_value/4096.0)*3300.0 ) ; //mV value
            Temperature=(int) mV/10.0; //Temp as integer


        if (Temperature > (setpoint+2) ){
            state=1;
            GPIO_PORTF_DATA_R &=~ 0x08;                 // turn off heater on
PF3
            GPIO_PORTF_DATA_R &=~ 0x04;    // turn off led on PF2

        }
        else if(Temperature<(setpoint-1)){
            state=0;
            GPIO_PORTF_DATA_R |= 0x08;                  //heater on PF3
            GPIO_PORTF_DATA_R |= 0x04;    // led on PF2

        }
        if(Temperature>=(setpoint-1) && Temperature <= (setpoint+2) &&
state==0)
        {
            GPIO_PORTF_DATA_R |= 0x08;                  //heater on PF3
            GPIO_PORTF_DATA_R |= 0x04;    //led on PF2
        }
        else if (Temperature>=(setpoint-1) && Temperature <= (setpoint+2) &&
state==1)
        {
            GPIO_PORTF_DATA_R &=~ 0x08;                 // turn off heater on
PF3
```

```
            GPIO_PORTF_DATA_R &=~ 0x04;    // turn off led on PF2
        }



        itoa(Temperature, msg.Txt1,10); //Measured Value
        itoa(setpoint, msg.Txt2,10); //setpoint Value
        xQueueSend(xLCDQueue, &msg,0);

if(Temperature> AlarmValue) //Alarm??
{
    xQueueSend(xBuzzerQueue, &on,0); //Buzzer On
}
else{
    xQueueSend(xBuzzerQueue, &off,0); //Buzzer Off
}

  }
}
```

3. Buzzer_Task :

Description :

This is the Buzzer Controller task. At the beginning of this task, port pin PF1 is configured as output. The task receives the buzzer state through the queue xBuzzerQueue. If (1) is received, then an alarm condition is assumed and the buzzer is activated. If on the other hand (0) is received, then the buzzer is de-activated

Code :

```
void Buzzer_Task(void *pvParameters)//Buzzer Task //PF1
{
char BuzzerState;
BuzzerState=0;

while(1)
{
        xQueueReceive(xBuzzerQueue,&BuzzerState,0); //Get Data from main task

        if(BuzzerState==1)//Alarm?
        {
                GPIO_PORTF_DATA_R |= 0x02; //Buzzer is turned ON
        }
        else{
                GPIO_PORTF_DATA_R &=~ 0x02; //Buzzer is turned OFF
        }

}
}
```

4. LCD_Task :

Description :

This is the LCD Controller task. At the beginning of this task the LCD is initialized, display is cleared, and the cursor is set to be OFF so that it is not visible. The task receives the measured and the setpoint temperature values through queue xLCDQueue in the form of a structure Character arrays Txt1 and Txt2 store the measured and the setpoint temperature values respectively. The LCD display is refreshed every second

Code :

```
void LCD_Task(void *pvParameters)
{
typedef struct Message
    {
        char Txt1[4];
        char Txt2[4];
    }Amessage;

    Amessage msg;
LCD LCD;
LCD=LCD_create();
LCD_setup(&LCD);
LCD_init(&LCD);

    while(1)
    {
        xQueueReceive(xLCDQueue, &msg,0); //Receive Data from main task
        LCD_clear(&LCD);
        LCD_setPosition(&LCD,1,1);
        LCD_sendString(&LCD, "Measured: "); //Heading
        LCD_sendString(&LCD,msg.Txt1); //display temperature
        LCD_sendByte(&LCD,1,(char)223);
        LCD_sendString(&LCD, "C"); //degree unit
        LCD_setPosition(&LCD,2,1);
        LCD_sendString(&LCD, "Setpoint: "); //Heading
        LCD_sendString(&LCD,msg.Txt2); //display setpoint
        LCD_sendByte(&LCD,1,(char)223);
        LCD_sendString(&LCD, "C"); //degree unit
        vTaskDelay(pdMS_TO_TICKS(1000)); //wait one second
    }
}
```

5. UART_Task :

Description :

This is the UART Controller task. At the beginning of this task, UART is initialized to operate at 9600 baud. The message (Enter Temperature Setpoint (Degrees):) is displayed on the PC screen using the Putty terminal emulation software. The required setpoint value (in integer) is read from the keyboard and is sent to Task 1 through xUARTQueue. Then the message (Temperature setpoint changed…) is displayed on the PC screen.

Code :

```
void UART_Task(void *pvParameters)
{
 unsigned char N;
 //unsigned AdcValue;
 unsigned char Total;
 printstring(" Hello!! ");
 while (1)
 {

        printstring("\n\r\n\r Please Enter Temperature Setpoint (Degrees): ");
        N=0;
        Total=0;
        while(1)
        {
                N= UART0_Receiver();      //get a character from UART0
          UART0_Transmitter(N);      // echo that character

                if (N == '\n') //if Enter break from this while loop
                        {
                        break;
                        }
    N= N-'0';   //Pure number
                Total = 10*Total + N; //Total number

        }

  xQueueSend(xUARTQueue, &Total, pdMS_TO_TICKS(30)); //send via queue
        printstring("\n\r Temperature Setpoint changed.. ");
}
```

```
}


// Function to swap two numbers
void swap(char *x, char *y) {
    char t = *x; *x = *y; *y = t;
}

// Function to reverse `buffer[i…j]`
char* reverse(char *buffer, int i, int j)
{
    while (i < j) {
        swap(&buffer[i++], &buffer[j--]);
    }

    return buffer;
}

// Iterative function to implement `itoa()` function in C
char* itoa(int value, char* buffer, int base)
{
    // invalid input
    if (base < 2 || base > 32) {
        return buffer;
    }

    // consider the absolute value of the number
    int n = abs(value);

    int i = 0;
    while (n)
    {
        int r = n % base;

        if (r >= 10) {
            buffer[i++] = 65 + (r - 10);
        }
        else {
            buffer[i++] = 48 + r;
        }
        n = n / base;
    }

    // if the number is 0
    if (i == 0) {
        buffer[i++] = '0';
    }

    // If the base is 10 and the value is negative, the resulting string
    // is preceded with a minus sign (-)
    // With any other base, value is always considered unsigned
    if (value < 0 && base == 10) {
        buffer[i++] = '-';
    }

    buffer[i] = '\0'; // null terminate string

    // reverse the string and return it
    return reverse(buffer, 0, i - 1);
}

void ADC_Init(void){
    /* Enable Clock to ADC0 and GPIO pins*/
    SYSCTL->RCGCGPIO |= (1<<4);   /* Enable Clock to GPIOE or PE3/AN0 */
    SYSCTL->RCGCADC |= (1<<0);    /* AD0 clock enable*/

    /* initialize PE3 for AIN0 input  */
    GPIOE->AFSEL |= (1<<3);       /* enable alternate function */
    GPIOE->DEN &= ~(1<<3);        /* disable digital function */
    GPIOE->AMSEL |= (1<<3);       /* enable analog function */

    /* initialize sample sequencer3 */
    ADC0->ACTSS &= ~(1<<3);        /* disable SS3 during configuration */
    ADC0->EMUX &= ~0xF000;    /* software trigger conversion */
    ADC0->SSMUX3 = 0;         /* get input from channel 0 */
    ADC0->SSCTL3 |= (1<<1)|(1<<2);        /* take one sample at a time, set flag at 1st
sample */
    ADC0->ACTSS |= (1<<3);         /* enable ADC0 sequencer 3 */
}
```