PROJECT REPORT
5400(COURSE)

BODDU MOURYA CHANDRA
1002022108

Contents

INTRODUCTION:

The aim of this project is to build a device capable of handling chemical samples in turret-mounted test tubes while performing chemical analysis using colorimetry.

REQUIRED HARDWAE:

PART	QUANTITY
3D-printed turret	1
17PM-K374BN01CN stepper motor with cable	1
3D-printed motor base and optical mount	1
WP154A4SEJ3VBDZGC-CA RGB LED	1
TEPT5600 ambient light sensor	1
SN754410 motor driver	1
2N3904 transistor	3
4.7 Kohm resistor	6
10 kohm resistor	1
0.01µF capacitor	1
TM4C123GXL board	1
2X10 pin header ,2.54mm pitch	2
2-position terminal strip(motor)	2
3-position terminal strip(LED/sensor)	2
Test tube	6

REQUIRED SOFTWARE:

1) Code composer studio
2) Putty terminal

Theory of Operation:

This project supports two user interfaces

1) User command interface

2) IR remote interface.

User can measure the pH value of a liquid using this project .Whenever this system is switched on it undergoes calibration state which means it returns to the home position (Reference tube will be aligned to the light and sensor) and fixes the RGB light intensities .After the calibration user has to select the tube in which required liquid is present and use **"goto<Required tube>"** function so that the tube will be present in between the light and sensor .User can measure the RAW values by using **"measure <tube number>"** command and pH values using "**measurepH <tube number>**" .User can also use these features by using specific buttons in the IR remote .

All the commands and their description is described below.

Instruction	Description
home	Moves turret to reference position and stores home position
goto TUBE	Rotates the turret to position 1, 2, 3, 4, 5, or R (reference)
calibrate	Rotates the turret to reference position and calibrates the light path
measure TUBE	Rotates the turret to position 1, 2, 3, 4, 5, or R (reference) and displays the raw RGB data values from the A/D converte
measurepH TUBE	Rotates the turret to position 1, 2, 3, 4, 5, or R (reference) and displays the pH or chlorine values

Code:

```
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include "clock.h"
#include "tm4c123gh6pm.h"
#include "wait.h"
#include "uart0.h"
#include "Stepper_motor.h"
#include "adc0.h"
#include "rgb_led.h"

// PortB masks
#define AIN11_MASK 32

//PORT D bit band definition
#define PD0_IR_DATA  (*((volatile uint32_t *)(0x42000000 + (0x400073FC-0x40000000)*32
+ 0*4)))

// PortD masks
#define IR_DATA_IN_MASK 1

// port F Bitband aliases
#define RED_LED      (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32
+ 1*4)))
#define GREEN_LED    (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32
+ 3*4)))
#define BLUE_LED     (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32
+ 2*4)))

// PortF masks
#define BLUE_LED_MASK               4
#define GREEN_LED_MASK              8
#define PUSH_BUTTON_MASK            16

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
//IR sensor

bool     valid      = false ;
uint32_t time_diff  = 0 ,ADDR_DATA_REG=0    ;
uint32_t time[50]   = {0}   ;
uint8_t  count      = 0     ;
uint8_t  code       = 0     ;
uint8_t  ADDR       = 0     ;
uint8_t  NEG_ADDR   = 0     ;
uint8_t  DATA       = 0     ;
uint8_t  NEG_DATA   = 0     ;

//Ambient sensor
uint8_t Tube_value      =   0   ;
```

```
uint16_t ii             =   0   ;
uint16_t raw            =   0   ;
uint16_t pwm_r          =   0   ;
uint16_t pwm_g          =   0   ;
uint16_t pwm_b          =   0   ;
uint16_t analog_r       =   0   ;
uint16_t analog_g       =   0   ;
uint16_t analog_b       =   0   ;
float analog_r_ref   =   0   ;
float analog_g_ref   =   0   ;
float analog_b_ref   =   0   ;
char str[100];

uint16_t RAW_R[5]={3149,2737,2997,2905,2846}    ;
uint16_t RAW_G[5]={2663,1163,763,232,1082}      ;
uint16_t RAW_B[5]={887,715,683,605,460}        ;

float d_sqr[5]    =   {0}                       ;
float pH_HC[5]    =   {6.8,7.5,7.8,8.7,7.2}     ;
float fin_pH      = 0                           ;

//--------------------------------------------------------------------------------
// Subroutines
//--------------------------------------------------------------------------------

void enableWideTimer()
{
    // Configure Wide Timer 2 as counter of external events on CCP0 pin
    WTIMER2_CTL_R   &= ~TIMER_CTL_TAEN
;// turn-off counter before reconfiguring
    WTIMER2_CFG_R   = 4
;// configure as 32-bit counter (A only)
    WTIMER2_TAMR_R  = TIMER_TAMR_TAMR_CAP | TIMER_TAMR_TACDIR |TIMER_TAMR_TACMR
;// configure for edge time mode, count up
    WTIMER2_CTL_R   = 0
;// Disable the timer
    WTIMER2_IMR_R   = 0
;// turn-off interrupts
    WTIMER2_TAV_R   = 0
;// zero counter for first period
    WTIMER2_CTL_R   |= TIMER_CTL_TAEN
;// turn-on counter

}

void calibrate(void)
{

    pwm_r      =   0   ;
    pwm_g      =   0   ;
    pwm_b      =   0   ;
    //RED TEST
    raw = 0;
    for (ii = 0; raw < 3072 && ii < 1024; ii++) {
        setRgbColor(ii, 0, 0);
```

```
        waitMicrosecond(30000);
        raw = readAdc0Ss3();
    }
     pwm_r = ii;
     analog_r = raw;
    sprintf(str, "red_pwm:          %4u\n", pwm_r);
    putsUart0(str);
    sprintf(str, "red_analog:          %4u\n", analog_r);
    putsUart0(str);
    raw = 0;

    //GREEN TEST
    for (ii = 0; raw < 3072 && ii < 1024; ii++) {
        setRgbColor(0, ii, 0);
        waitMicrosecond(30000);
        raw = readAdc0Ss3();
    }
     pwm_g = ii;
     analog_g = raw;

    sprintf(str, "green_pwm:          %4u\n", pwm_g);
    putsUart0(str);
    sprintf(str, "green_analog:          %4u\n", analog_g);
    putsUart0(str);
    raw = 0;

    //BLUE TEST
    for (ii = 0; raw < 3072 && ii < 1024; ii++) {
        setRgbColor(0, 0, ii);
        waitMicrosecond(30000);
        raw = readAdc0Ss3();
    }
     pwm_b = ii;
     analog_b = raw;

    sprintf(str, "blue_pwm:          %4u\n", pwm_b);
    putsUart0(str);
    sprintf(str, "blue_analog:          %4u\n", analog_b);
    putsUart0(str);

    analog_r_ref = analog_r;
    analog_g_ref = analog_g;
    analog_b_ref = analog_b;

    raw = 0;
    setRgbColor(0, 0, 0);
}

void measure(uint8_t tube,uint16_t *r,uint16_t *g,uint16_t *b)
{
    goto_tube(tube);
    setRgbColor(0, 0, 0);
    waitMicrosecond(10000); //This wait is to make tube settled
    //Set red LED
    uint16_t i = 0;
```

```
    for (i = 0; i <= pwm_r; i++) {
        setRgbColor(i, 0, 0);
        waitMicrosecond(1000);
        *r=readAdc0Ss3();
    }
    setRgbColor(0, 0, 0);
    waitMicrosecond(10000);
    //Set Green LED
    for (i = 0; i <= pwm_g; i++) {
        setRgbColor(0, i, 0);
        waitMicrosecond(1000);
        *g=readAdc0Ss3();
    }
    setRgbColor(0, 0, 0);
    waitMicrosecond(10000);
    //Set Green LED
    for (i = 0; i <= pwm_b; i++) {
        setRgbColor(0, 0, i);
        waitMicrosecond(1000);
        *b=readAdc0Ss3();
    }
    setRgbColor(0, 0, 0);
}

void measurepH(uint8_t tube)
{
    float d_first_min = 0 , d_second_min = 0 ,temp = 0 ,diff_r = 0,diff_g = 0,diff_b
= 0,diff_r_div = 0,diff_g_div = 0,diff_b_div = 0,diff_r_sqr = 0,diff_g_sqr =
0,diff_b_sqr = 0  ;
    uint8_t first_min_index = 0,second_min_index = 0    ;
    measure(tube,&analog_r,&analog_g,&analog_b) ;


    for(ii=0;ii<5;ii++)
    {
        diff_r = (float)(analog_r - RAW_R[ii])  ;
        diff_g = (float)(analog_g - RAW_G[ii])  ;
        diff_b = (float)(analog_b - RAW_B[ii])  ;

        diff_r_div = diff_r/3072    ;
        diff_g_div = diff_g/3072    ;
        diff_b_div = diff_b/3072    ;

        diff_r_sqr = diff_r_div * diff_r_div    ;
        diff_g_sqr = diff_g_div * diff_g_div    ;
        diff_b_sqr = diff_b_div * diff_b_div    ;

       // d_sqr[ii] = (((analog_r - RAW_R[ii])/analog_r_ref)*((analog_r -
RAW_R[ii])/analog_r_ref)) + (((analog_g - RAW_G[ii])/analog_g_ref)*((analog_g -
RAW_G[ii])/analog_g_ref)) + (((analog_b - RAW_B[ii])/analog_b_ref)*((analog_b -
RAW_B[ii])/analog_b_ref))  ;

        d_sqr[ii]   = diff_r_sqr + diff_g_sqr + diff_b_sqr  ;
    }
```

```
    //Finding minimum
    d_first_min = d_sqr[0];

     for ( ii = 1 ; ii < 5 ; ii++ )
     {
         if ( d_sqr[ii] < d_first_min )
         {
             d_first_min = d_sqr[ii];
             first_min_index = ii+1;
         }
     }

     //finding maximum
     for ( ii = 0 ; ii < 5 ; ii++ )
     {
         if ( d_sqr[ii] > d_second_min )
         {
             d_second_min = d_sqr[ii];
         }
     }

     //finding second minimum

     for(ii=0 ;ii <5 ;ii++)
     {
        temp = d_sqr[ii]    ;
        if(temp > d_first_min && temp < d_second_min)
        {
            d_second_min = temp ;
            second_min_index = ii   ;
        }
     }

     //pH formula

     fin_pH = pH_HC[first_min_index] + ((pH_HC[second_min_index] -
pH_HC[first_min_index])*(d_first_min/(d_first_min+d_second_min)))    ;

     sprintf(str, "pH:          %4f\n", fin_pH);
     putsUart0(str);

}

// Initialize Hardware
void initHw(void)
{
    // Initialize system clock to 40 MHz
    initSystemClockTo40Mhz();

    // Enable clocks for LED's and PUSH BUTTONS
    SYSCTL_RCGCWTIMER_R |= SYSCTL_RCGCWTIMER_R2
;
    SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1 | SYSCTL_RCGCGPIO_R5 |SYSCTL_RCGCGPIO_R2
|SYSCTL_RCGCGPIO_R3 |SYSCTL_RCGCGPIO_R4 |SYSCTL_RCGCGPIO_R0;
    _delay_cycles(3);
```

```
    // Configure SIGNAL_IN for frequency and time measurements
    GPIO_PORTD_AFSEL_R  |= IR_DATA_IN_MASK                                  ;//
select alternative functions for SIGNAL_IN pin
    GPIO_PORTD_PCTL_R   &= ~GPIO_PCTL_PD0_M                                 ;//
map alt fns to SIGNAL_IN
    GPIO_PORTD_PCTL_R   |= GPIO_PCTL_PD0_WT2CCP0
;//writing encoding value in port control register
    GPIO_PORTD_DEN_R    |= IR_DATA_IN_MASK                                  ;//
enable bit 1 for digital input

    //Enable interrupt for PORT D(To call the GPDIsr)
    NVIC_EN0_R          |= 1 << (INT_GPIOD-16)                              ;//
turn-on interrupt 3 (GPIO port D)
    GPIO_PORTD_IM_R     &= ~IR_DATA_IN_MASK                                 ;//
disable the PD0 interrupt
    GPIO_PORTD_IS_R     &= ~IR_DATA_IN_MASK                                 ;//
clearing the 1st bit of interrupt sense register to make it edge sensitive
    GPIO_PORTD_IEV_R    &= ~IR_DATA_IN_MASK                                 ;//
clearing the 1st bit to make it negative edge trigger
    GPIO_PORTD_IM_R     |= IR_DATA_IN_MASK                                  ;//
enable the PD0 interrupt

    // Configure AIN11 as an analog input
    GPIO_PORTB_AFSEL_R |= AIN11_MASK;                 // select alternative functions
for AIN11 (PB5)
    GPIO_PORTB_DEN_R &= ~AIN11_MASK;                  // turn off digital operation
on pin PB5
    GPIO_PORTB_AMSEL_R |= AIN11_MASK;                 // turn on analog operation on
pin PB5

    // Configure LED and pushbutton pins
    GPIO_PORTF_DIR_R    |= GREEN_LED_MASK | BLUE_LED_MASK                    ;//
bits 1 and 2 are outputs, other pins are inputs
    GPIO_PORTF_DIR_R    &= ~PUSH_BUTTON_MASK                                ;//
bit 4 is an input
    GPIO_PORTF_DR2R_R   |= GREEN_LED_MASK | BLUE_LED_MASK                   ;//
set drive strength to 2mA (not needed since default configuration -- for clarity)
    GPIO_PORTF_DEN_R    |= PUSH_BUTTON_MASK | GREEN_LED_MASK | BLUE_LED_MASK    ;//
enable LEDs and pushbuttons
    GPIO_PORTF_PUR_R    |= PUSH_BUTTON_MASK                                 ;//
enable internal pull-up for push button

}


//GPIO Port D ISR
void GPDIsr(void)
{
    //putsUart0("\n Interrupt handler\r\n ");
    if(count == 0)
    {
        WTIMER2_TAV_R = 0           ;
        time[count] = WTIMER2_TAV_R ;
```

```
        count = count + 1           ;
    }
    else if(count == 1)
    {
        time[count] = WTIMER2_TAV_R ;
        //check whether this edge is the valid start edge of the IR command
        if((time[1]-time[0] >= 520000) && (time[1]-time[0] <= 560000))
        {
            count = count + 1   ;
        }
        else
        {
            count = 0   ;
           // putsUart0("\n invalid start command \n");
        }
    }

    else if(count > 1 && count <34)
    {
        time[count] = WTIMER2_TAV_R                             ;
        time_diff   = time[count] - time[count-1]               ;
        //checking whether these bits are valid IR Addr and Data
        if((time_diff >= 33750 && time_diff <= 56250)||(time_diff >= 78750 &&
time_diff <= 101250))
        {
            count = count + 1   ;
        }
        else
        {
            //putsUart0("\n invalid data \n");
            count = 0   ;
        }
    }


    if(count > 33)
    {
        count = 0   ;
        GREEN_LED=1;


        ADDR_DATA_REG = 0;
        for(ii=1;ii<33;ii++)
        {
            time_diff = time[ii+1] - time[ii] ;
           if(time_diff > 78750 && time_diff < 101250)
               ADDR_DATA_REG  |= 1 << ii-1  ;
        }

        ADDR        = ADDR_DATA_REG          ;
        NEG_ADDR    = ADDR_DATA_REG >> 8     ;
        DATA        = ADDR_DATA_REG >> 16    ;
        NEG_DATA    = ADDR_DATA_REG >> 24    ;
```

```
if(ADDR == (uint8_t)~NEG_ADDR && DATA == (uint8_t)~NEG_DATA)
{
    valid = true    ;
    code  = DATA    ;
    //putsUart0("\n True \n");
}
else
{
    valid = false   ;
    putsUart0("\n Fail \n");
}

if(code == 0x58) //R
{
    goto_tube(0)    ;
}
else if(code == 0x54) //L30
{
    goto_tube(1)    ;
}
else if(code == 0x50) //L30
{
    goto_tube(2)    ;
}
else if(code == 0x1C) //L30
{
    goto_tube(3)    ;
}
else if(code == 0x18) //L30
{
    goto_tube(4)    ;
}
else if(code == 0x14) //L30
{
    goto_tube(5)    ;
}
else if(code == 0x59) //L30
{
    measure(0,&analog_r,&analog_g,&analog_b)    ;
    sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
    putsUart0(str);
}
else if(code == 0x55) //L30
{
    measure(1,&analog_r,&analog_g,&analog_b)    ;
    sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
    putsUart0(str);
}
else if(code == 0x51) //L30
{
    measure(2,&analog_r,&analog_g,&analog_b)    ;
    sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
    putsUart0(str);
}
else if(code == 0x1D) //L30
```

```
        {
            measure(3,&analog_r,&analog_g,&analog_b)    ;
            sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
            putsUart0(str);
        }
        else if(code == 0x19) //L30
        {
            measure(4,&analog_r,&analog_g,&analog_b)    ;
            sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
            putsUart0(str);
        }
        else if(code == 0x15) //L30
        {
            measure(5,&analog_r,&analog_g,&analog_b)    ;
            sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
            putsUart0(str);
        }
        else if(code == 0x45) //L30
        {
            measurepH(0)    ;
        }
        else if(code == 0x49) //L30
        {
            measurepH(1)    ;
        }
        else if(code == 0x4D) //L30
        {
            measurepH(2)    ;
        }
        else if(code == 0x1E) //L30
        {
            measurepH(3)    ;
        }
        else if(code == 0x1A) //L30
        {
            measurepH(4)    ;
        }
        else if(code == 0x16) //L30
        {
            measurepH(5)    ;
        }
        else if(code == 0x5C) //Brightup
        {
            home()  ;
        }
        else if(code == 0x5D) //Bright down
        {
            calibrate()  ;
        }
    }

    GPIO_PORTD_ICR_R |= IR_DATA_IN_MASK;           // clear interrupt flag

}
```

```
//-----------------------------------------------------------------------------
// Main
//-----------------------------------------------------------------------------

int main(void)
{
    // Initialize hardware
    initHw();
    //Initialize Uart
    initUart0();
    //Initialize ADC
    initAdc0Ss3();
    //Initialize RGB
    initRgb();


    // Setup UART0 baud rate
    setUart0BaudRate(115200, 40e6);
    enableWideTimer()   ;

    //Blink the Green LED to ensure Program is running
    GREEN_LED        = 1    ;
    waitMicrosecond(2000000);
    GREEN_LED        = 0    ;

    //Initialize the stepper motor
    initStepperMotor()  ;

    // Use AIN11 input with N=4 hardware sampling
    setAdc0Ss3Mux(11);
    setAdc0Ss3Log2AverageCount(2);//(Refer 13.3.3 in data sheet)

    calibrate() ;

    USER_DATA data ;

    while (1)
    {
        //Get the String from user
        getsUart0(&data);

#ifdef DEBUG
        //print the string
        putsUart0(data.buffer);
        putsUart0("\n");
#endif
        //Parse fields
        parseFields(&data);

#ifdef DEBUG
        for( ii = 0;ii < data.fieldCount ;ii++){
            putcUart0(data.fieldType[ii]);
            putsUart0("\t");
            putsUart0(&(data.buffer[data.fieldPosition[ii]]));
            putsUart0("\n");
```

```
        }
#endif

        if (isCommand(&data, "calibrate", 0))
            calibrate();
        else if (isCommand(&data, "tube", 2))
        {
            //de referencing the return address to check whether it is character or
not
            if (*(getFieldString(&data, 1)) == 'R')
            {
                goto_tube(0)    ;
            }
            else
            {
                Tube_value = (uint8_t) getFieldInteger(&data, 1);

                if (Tube_value < 6){
                    goto_tube(Tube_value)    ;
                }
                else
                    putsUart0("\n invalid Tube Selection ");
            }

        }
        else if (isCommand(&data, "measurepH", 2))
        {
            //de referencing the return address to check whether it is character or
not
            if (*(getFieldString(&data, 1)) == 'R')
            {
                measurepH(0)    ;
            }
            else
            {
                Tube_value = (uint8_t) getFieldInteger(&data, 1);

                if (Tube_value < 6){
                    measurepH(Tube_value)    ;
                }
                else
                    putsUart0("\n invalid Tube Selection ");
            }

        }
        else if (isCommand(&data, "measure", 2))
        {
            //de referencing the return address to check whether it is character or
not
            if (*(getFieldString(&data, 1)) == 'R')
            {
               measure(0,&analog_r,&analog_g,&analog_b)    ;
               sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
               putsUart0(str);
                //measurepH(0)    ;
```

```
            }
            else
            {
                Tube_value = (uint8_t) getFieldInteger(&data, 1);

                if (Tube_value < 6){
                     measure(Tube_value,&analog_r,&analog_g,&analog_b)    ;
                     sprintf(str, "(%4u,%4u,%4u)\n", analog_r,analog_g,analog_b);
                     putsUart0(str);
                    //measurepH(Tube_value)    ;
                }
                else
                    putsUart0("\n invalid Tube Selection ");
            }

        }
        else if (isCommand(&data, "home", 0))
        home();
    }

}
```

Conclusion:

This system measures the pH of a liquid effectively with a limited resource i.e an ambient sensor and RGB LED and also provides a user friendly interface to use the features of this system .