Readme For Project 4 Team Members :

1)Suraj Thite (suraj.thite@colorado.edu)

2)Atharv Desai (atharv.desai@colorado.edu)

This is a readme file for the fourth project assignment in the Principles of Embedded Software Course for FALL '19.

The below enumerated files are contained in the repository

1) PES_PROJECT4

i) Project Settings

ii) Build Targets

iii) Binaries

iv) Includes

v) CMSIS

vi) Board

vii) Drivers

viii) Source a) main.c & main.h

b) logger.c & logger.h

c) RGBled.c & RGBled.h

d) i2c.c & i2c.h

e) unittest.c & unittest.h

f) statemachine.c & statemachine.h

ix) Startup

x) Utilities

xi) Debug

xii) PESProject4 PE Debug.launch

2) Readme File (Readme Markdown File)

3) I2C Transactions Screenshot

INSTALLATION & EXECUTION NOTES

The code is tested on the enviornment below:

1. MCUXpresso IDE which is an easy to use Eclipse-based development environment for NXP® MCUs based on Arm® Cortex®-M cores.

2. During this project, this IDE was used to code,execute state machines and interface TMP102 with FRDM board using I2C protocol on FRDM-KL25z and print their output on the IDE's serial terminal.

3. Also, we were able to check and verify the workability of the state machines using the unittest cases in the project.

4. Using Debug Port Logic Analyser, the SCL and SDA pin data in the form of waveform was used to capture I2C transactions

5. The hardware used in this project was FRDM-KL25Z board which has been designed by NXP and built on ARM® Cortex™-M0+ processor.

6. The editor used to build the code is gedit version 2.3.8.1 on Linux Mint Machine.

7. To execute the executable file simply type ./(filename) in linux gcc environment while click on debug (bug icon) and then resume button to execute the file on MCUXpresso.

8. Kindly use notepad++ for viewing .out files ,particularly for first output since they have been misaligned due to character "Space or Tab" encoding.

9. Set #define (MODE) to 1 or 0 for test cases or state machine modes respectively.

10. Set modes to Test, Debug or Status mode by setting the vale for variable 'a' in logger.c file accordingly.

DIFFICULTIES & OBSERVATIONS:

1. While capturing the I2C transactions output on Logic Analyser, faced issues in capturing the SCL, SDA waveforms in timing mode.

2. While designing the disconnected state, we were not able to swap between Connected and Disconnected state using software control. Many times, reset button needed to be pressed to check whether the current state has changed from connected to disconnected state and viceversa.

3. In state-oriented state machine, we were able to define all states like Alert, read temperature in a single function . However, in table driven, we had to create standalone functions for each and every state to make sure about optimal working of the table driven machine.

4. While implementing logger earlier, the enum values were being passed as arguments in integer format only. But on accessing them in other .c files using extern keyword, the issue was resolved.

/**
 * @file   PES-PROJECT4.c
 * @brief   Application entry point.
 */

/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

/*
 * @brief   Application entry point.
 */

#include "main.h"
#include "RGBled.h"
#include "unitTest.h"
#include "uCUnit-v1.0.h"

#define MODE (0)

//MODE 0 FOR STATE MACHINE AND MODE 1 FOR TEST CASES

int16_t temp_val; //GLobal Variable to Store current temperature value
int32_t average_temp;//Global variable to store average temperature value
uint32_t total_count; //Global variable to store total no of iterations
states current_state1; //Global variable to store current state
events event1 = INIT;//Global variable to store current event in the state machine
modes select;
bool disconnected; //Check on disconnected state
uint8_t conn_check;//Check on connection check
uint8_t terminate_flag;

int main(void) {

        /* Init board hardware. */

```
        BOARD_InitBootPins();
        BOARD_InitBootClocks();
        BOARD_InitBootPeripherals();
        /* Init FSL debug console. */
        BOARD_InitDebugConsole();
        RGB_init(); //Initialize the RGB LEDs
        RGB_OFF();      //TURN Them off for now
        i2c_init(); //Initialize the i2c
        Init_SysTick(); //Initialize the Systick timer for 15 seconds delay

#if MODE == 0
        while(terminate_flag==0)
        {
                state_machine_1();// start state machine
        }
#else
        {
                UCUNIT_Init();
                UCUNIT_WriteString("\n*************************************");
                UCUNIT_WriteString("uCUnit demo application");
                UCUNIT_WriteString(__DATE__);
                UCUNIT_WriteString("\nTime:     ");
                UCUNIT_WriteString(__TIME__);
                printf("\n \r Running Test Cases");
                Testsuite_RunTests();
        }
#endif

        printf("\n \r **************PROGRAM TERMINATED*******************");

        return 0 ;
}
```

```c
/*
 * state_machine.c
 *
 *  Created on: Nov 2, 2019
 *      Author: SURAJ THITE
 */
#include "state_machine.h"
#include "RGBled.h"

//extern global variables
extern int32_t average_temp;
extern uint32_t total_count;
extern int16_t temp_val;
extern states current_state1 ;
extern events event1;
extern uint8_t _15_seconds_counter;
extern modes a;
extern fnnames fn_name;
extern uint8_t averge_15_wait;
extern uint8_t terminate_flag;
uint8_t averge_15_wait = 0;
int switch1 =0;
extern uint8_t _15seconds_passed ;

////////////////////////////////////////////////////////////////////////////////STATE MACHINE -1
////////////////////////////////////////////////////////////////////////////////////////////////
/*****************************************************************************************************
 * Function Name:void state_machine_1()
 * Description :This is switch case event driven state machine.
 * @input: void
 * @Return :void
 *****************************************************************************************************/
void state_machine_1()
{

        fn_name=state_machine1;
        switch(current_state1)
        {
        //POST TESTS
        case POST_STATES:
                if(event1 == INIT )
                {
                        Log_String(a,state_machine1,"\n \r <<<<<<<<<<<<<<<<<<<<<<<<<<<<<STATE MACHINE
1>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
                        //printf("\n \r <POST_STATES>"); //Use Log_String("<POST_STATES>");
                        if (a==Test || a== Status)
                                Log_String(a,fn_name,"\n \r  ********************POST_STATES: Running
Power on Startup test **********************");
                        else
                                Log_String(a,fn_name,"\n \r POST_STATES: LED set to green");
                        event1 = null_event;
                        current_state1 =I2C_READ_TEMPERATURE;
                        POST_TEST_read_bytes(0x90, 0x01);        //Set PASS and Fail Event
                }

                break;
                //READ TEMPERATURE VALUE FROM TMP102 IMPLEMENTING I2C
        case I2C_READ_TEMPERATURE:
                if(event1 == PASS || event1 == TIME_OUT_1_2_3)
                {

                        uint8_t status =0;
                        status = check_connection(0x90,0x00);
                        if(status==1)
                        {
                                printf("\n \r DISCONNECTED");
                                current_state1=DISCONNECT;
                                event1 =DISCONNECT_EVENT;
                                break;
```

```
                        }
                        led_switch(2);
                        event1 = null_event;
                        total_count= total_count +1;
                        temp_val = i2c_read_temperature(0x90, 0x00);     //Read Current Temperature Value
                        if (a==Test || a== Status)
                                Log_String(a,fn_name,"\n \r <<<<<<<<<<<<<<<<<<<<<<<<<<<< STATE MACHINE 1
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> \n \r  ******************** ENTERED <STATE I2C_READ_TEMPERATURE>
********************\n \r  ********************CURRENT TEMP VALUE :");
                        else
                                Log_String(a,fn_name,"\n \r  ********************ENTERED <STATE
I2C_READ_TEMPERATURE>  ******************** \n \r Temp Reading: LED set to green");
                        Log_integer(a,temp_val);
                        event1 = COMPLETE;       //Set Complete event
                        current_state1 =AVERAGE_WAIT;
                        if(temp_val<=0)
                        {
                                event1 = ALERT;
                                current_state1 = TEMP_ALERT;     //Over write current_state1 if alert
                        }
                }
                break;
                //ALERT STATE
        case TEMP_ALERT:

                if(event1==ALERT)
                {

                        uint8_t status =0;
                        status = check_connection(0x90,0x00);
                        if(status==1)
                        {
                                printf("\n \r DISCONNECTED");
                                current_state1=DISCONNECT;
                                event1 =DISCONNECT_EVENT;
                                break;
                        }
                                RGB_OFF();
                                led_switch(0);
                                //printf(" \n \r <TEMP_ALERT>");
                                event1 = null_event;
                                if (a==Test || a== Status)
                                        Log_String(a,fn_name,"\n \r \n  ********************ALERT :
TEMPERTURE BELOW 0 DEGREE C  ********************");
                                else
                                        Log_String(a,fn_name,"\n \r Temp Alert: LED set to blue ");
                                current_state1 =AVERAGE_WAIT; //Change to AVERAGE_WAIT STATE
                                event1 = COMPLETE;       //SET EVENT as complete
                }

                break;
                //CALCULATE AVERAGE and set current state to I2C READ TEMPERATURE OR  STATE MACHINE 2
DEPENDING UPON total_count
        case AVERAGE_WAIT:

                if ( event1 == COMPLETE)
                {
                        uint8_t status =0;
                        status = check_connection(0x90,0x00);
                        if(status==1)
                        {
                                printf("\n \r DISCONNECTED");
                                current_state1=DISCONNECT;
                                event1 =DISCONNECT_EVENT;
                                break;
                        }
                        led_switch(2);   //Change LED
                        //printf("\n \r ENTERED <AVERAGE WAIT>"); // Log_String("ENTERED <AVERAGE WAIT>
");
```

```
                        if (a==Test || a== Status)
                                Log_String(a,fn_name,"\n \r  ********************ENTERED <AVERAGE WAIT>
******************** \n \r  ********************AVERAGE TEMPERATURE VALUE FOR ITERATION:");
                        else
                                Log_String(a,fn_name,"\n \r ENTERED <AVERAGE WAIT> : LED set to green");
                        event1 = null_event;
                        average_temp = ((average_temp * (total_count-1)) +
(int32_t)temp_val)/(total_count); //Calculate average value
                        Log_integer(a,total_count);
                        Log_integer(a,average_temp);

                        if (_15_seconds_counter <4)
                        {
                                current_state1 =I2C_READ_TEMPERATURE;
                                Log_String(a,fn_name," ********************STATE CHANGED TO
I2C_READ_TEMPERATURE");
                                Log_integer(a,_15_seconds_counter);
                        }
                        //switch to next state machine if counter value is equal to 4
                        else
                        {       _15_seconds_counter=0;   //Reinitialize the counter to zero
                        current_state1 = STATE_MACHINE_SWITCH;
                        }
                }
                break;
                // STATE MACHINE SWITCH STATE
        case STATE_MACHINE_SWITCH:
                if(event1 == TIME_OUT_4)
                {
                        event1 = null_event;
                        current_state1 = STATE_MACHINE_SWITCH;
                        while(1)
                        {
                                state_machine_2(I2C_READ_TEMPERATURE);   //Handle next state machine
                                break;
                        }
                        event1 = INIT;
                }
                break;
                //DISCONNECT STATE
        case DISCONNECT:
                if (event1==DISCONNECT_EVENT)
                {
                        printf(" ********************  DISCONNECTED ********************");
                        event1 = null_event;
                        led_switch(1);   //Change LED to RED
                        current_state1=END;
                        event1=EXIT;
                }
                break;
        case END:
                if(event1==EXIT)
                {
                        Log_String(a, 1,"END STATE");
                        event1 = null_event;
                        terminate_flag =1;
                }
        }
}
////////////////////////////////////////////////////////////////////////////////STATE MACHINE -2
////////////////////////////////////////////////////////////////////////////////////////////
//STATE TRANSITION TABLE FOR STATE MACHINE 2
struct transition state_table [] =
{                               //CURRENT STATE  -- EVENT --   NEXT STATE//
                { POST_STATES, PASS,  I2C_READ_TEMPERATURE},
                { POST_STATES, FAIL,  END},
                { I2C_READ_TEMPERATURE, COMPLETE,  AVERAGE_WAIT},
                { I2C_READ_TEMPERATURE, ALERT,  TEMP_ALERT},
                { I2C_READ_TEMPERATURE, DISCONNECT_EVENT,  DISCONNECT},
```

```
                { AVERAGE_WAIT, TIME_OUT_1_2_3,  I2C_READ_TEMPERATURE},
                { AVERAGE_WAIT, TIME_OUT_4,  POST_STATES},
                { AVERAGE_WAIT, DISCONNECT_EVENT,  DISCONNECT},
                { TEMP_ALERT, DISCONNECT_EVENT,  DISCONNECT},
                { TEMP_ALERT, COMPLETE,  AVERAGE_WAIT},
                { DISCONNECT, EXIT,  END},

};

// arrays to pointer to functions
events (*state[])(void)=
{
                POST_STATE,
                I2C_READ_TEMPERATURE_STATE,
                AVERAGE_WAIT_STATE,
                TMEP_ALERT_STATE,
                DISCONNECTED_STATE,
                END_STATE
};

/*****************************************************************************************************
 * Function Name:void state_machine_2()
 * Description :This is a table driven state machine.
 * @input: CURRENT STATE
 * @Return :void
 *****************************************************************************************************/


void state_machine_2(states current_state2)
{
        Log_String(a,state_machine2,"\n \r <<<<<<<<<<<<<<<<<<<<<<<<<<<<<STATE MACHINE
2>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
        events (* state_function_ptr)(void);
        //state current_state = Temp_Reading;
        events event_code;

        while(current_state1 != I2C_READ_TEMPERATURE )   //  Run Loop until current_state 1 is set to
POST_STATES
        {
                state_function_ptr = state[current_state2];      // Store the function pointer  from the
array
                event_code = state_function_ptr();  // Execute function and store returned event
                current_state2 = state_transition__in_table(current_state2, event_code); //switch to next
state depending upon the state table

        }
        current_state1 = I2C_READ_TEMPERATURE; //CURRENT STATE = POST_STATES
        event1 = PASS;
}

/*****************************************************************************************************
 * Function Name:state_transition__in_table( states CR, events EC)
 * Description :This function looks for state and events in the state table
 * @input: state and returned event
 * @Return :void
 *****************************************************************************************************/
states state_transition__in_table( states CR, events EC)
{
        uint16_t max = 0;
        states next_State_TB;
        max = sizeof(state_table) / sizeof(state_table[0]);
        for(int i = 0; i < max ; i++)
        {

                if ((state_table[i].current_state2 == CR) && (state_table[i].event2 == EC))       //Value
found
                {
                        next_State_TB = state_table[i].next_state_2;
                        break;
```

```
                }
        }

        return next_State_TB;
}

/*****************************************************************************************************
 * Function Name:events POST_STATE(void)
 * Description :This is the state function for POST TESTS
 * @input: CURRENT STATE
 * @Return :events
 *****************************************************************************************************/

events POST_STATE(void)
{
        fn_name =POSTSTATE;
        if (a==Test || a== Status)
                Log_String(a,POSTSTATE,"\n \r ******************POST_STATES: Running Power on Startup
test");
        else
                Log_String(a,POSTSTATE,"\n \r POST_STATES: LED set to green");
        POST_TEST_read_bytes(0x90, 0x01);        //RUN POST
        return PASS;
}

/*****************************************************************************************************
 * Function Name:events I2C_READ_TEMPERATURE_STATE(void)
 * Description :This is the READS the TEMPERATURE and returns the events
 * @input: void
 * @Return :ALERT if Temperature is  less than zero else complete
 *****************************************************************************************************/
events I2C_READ_TEMPERATURE_STATE(void)
{
        //fn_name =I2C_READ_TEMPERATURESTATE;
        //current_state1 = AVERAGE_WAIT;

        uint8_t status =0;
        status = check_connection(0x90,0x00);
        if(status==1)
        {
                printf("\n \r DISCONNECTED");
                return DISCONNECT_EVENT;
        }
        led_switch(2);
        total_count= total_count +1;
        temp_val = i2c_read_temperature(0x90, 0x00); //REad temperature values and store it in a temporary
variable
        if (a==Test || a== Status)
                Log_String(a,I2C_READ_TEMPERATURESTATE,"\n \r ENTERED <STATE I2C_READ_TEMPERATURE> \n \r
******************CURRENT TEMP VALUE :");
        else
                Log_String(a,I2C_READ_TEMPERATURESTATE,"\n \r Temp Reading: LED set to green");
        //Log_String(a,fn_name,"\n \r ENTERED <STATE I2C_READ_TEMPERATURE>:");

        //Log_String(a,fn_name,"CURRENT TEMP VALUE :");
        Log_integer(a,temp_val);
        if(temp_val <=0)//CHECK FOR ALTERT EVENT
                return ALERT;
        else
                return COMPLETE;//Else change it to COMPLETE state
}

/*****************************************************************************************************
 * Function Name:events AVERAGE_WAIT_STATE(void)
 * Description :This function calculates average and waits for 15 seconds to pass thus returning the
events.
 * @input: void
 * @Return :TIMEOUT1_2_3 if counter value is less than 4.
 *****************************************************************************************************/
```

```
events AVERAGE_WAIT_STATE(void)
{
        uint8_t status =0;
        status = check_connection(0x90,0x00);
        if(status==1)
        {
                printf("\n \r DISCONNECTED");
                return DISCONNECT_EVENT;
        }
        led_switch(2);
        fn_name =AVERAGE_WAITSTATE;
        average_temp = ((average_temp * (total_count-1)) + (int32_t)temp_val)/(total_count);
        //printf("\n \r>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Entered Average State");
        if (a==Test || a== Status)
                Log_String(a,AVERAGE_WAITSTATE,"ENTERED <AVERAGE WAIT> \n \r ********************AVERAGE
TEMPERATURE VALUE FOR ITERATION:");
        else
                Log_String(a,AVERAGE_WAITSTATE,"\n \r Average Wait: LED set to green");
        Log_integer(a,total_count);
        Log_integer(a,average_temp);
        int current_cnt_temp = _15_seconds_counter;      //Store current count in temporary variable
        while(1)
        {
                if(current_cnt_temp != _15_seconds_counter)      //Wait for Systick timer to timeout
                {
                        break;
                }
        }

        if (_15_seconds_counter <3)
                return TIME_OUT_1_2_3;
        else if (_15_seconds_counter ==3)
        {
                _15_seconds_counter =0;
                current_state1 = I2C_READ_TEMPERATURE;
                event1 = PASS;
        }
}

/*****************************************************************************************************
 * Function Name:events TMEP_ALERT_STATE(void)
 * Description :This function handles the temperature alert state
 * @input: void
 * @Return :COMPLETE event
 *****************************************************************************************************/
events TMEP_ALERT_STATE(void)
{

        uint8_t status;
        status = check_connection(0x90,0x00);
        if(status==1)
        {
                printf("\n \r DISCONNECTED");
                return DISCONNECT_EVENT;
        }
        RGB_OFF();
        led_switch(0);
        fn_name =TMEP_ALERTSTATE;
        if (a==Test || a== Status)
                Log_String(a,TMEP_ALERTSTATE,"\n \r \n ********************ALERT : TEMPERTURE BELOW 0
DEGREE C********************");
        return COMPLETE;

}

/*****************************************************************************************************
 * Function Name:events DISCONNECTED_STATE(void)
 * Description :This function handles the disconnection state.
```

```
 * @input: void
 * @Return :EXIT EVENT
 ********************************************************************************************************/
events DISCONNECTED_STATE(void)
{
        led_switch(1);
        fn_name=DISCONNECTEDSTATE;
        if (a==Test || a== Status)
                Log_String(a,DISCONNECTEDSTATE,"\n \r \n ****************** Disconnected State
**********************");
        else
                Log_String(a,DISCONNECTEDSTATE,"\n \r Disconnected: LED set to Red ");
        return EXIT;
}
/********************************************************************************************************
 * Function Name:events END_STATE(void)
 * Description :This function terminates the program upon disconnection
 * @input: void
 * @Return :void
 ********************************************************************************************************/

events END_STATE(void)
{
        printf("\n \r ****************** END STATE ******************");
        terminate_flag =1;
}
```

```c
/*
 * RGBled.c
 *
 *  Created on: Sep 28, 2019
 *      Author:SURAJ THITE , ATHARV DESAI
 */



#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_gpio.h"

#include "clock_config.h"
#include "pin_mux.h"

/*******************************************************************************/
/* Function name:RGB_init
 * Parameters: void
 * Return : void
 * Description: Function to initialize the GPIO RGB Led Pins . */
/*******************************************************************************/
void RGB_init()
{
                gpio_pin_config_t led_blue_config = {
                kGPIO_DigitalOutput, 1,
            };  //Config the pin  for BLUE LED to Digital Output
                GPIO_PinInit(BOARD_LED_BLUE_GPIO,BOARD_LED_BLUE_GPIO_PIN, &led_blue_config);
            gpio_pin_config_t led_red_config = {
                kGPIO_DigitalOutput, 1,
            };  //Config the pin  for RED LED to Digital Output
            GPIO_PinInit(BOARD_LED_RED_GPIO,BOARD_LED_RED_GPIO_PIN, &led_red_config);
            gpio_pin_config_t led_green_config = {
                kGPIO_DigitalOutput, 1,
            };  //Config the pin  for GREEN LED to Digital Output
            GPIO_PinInit(BOARD_LED_GREEN_GPIO,BOARD_LED_GREEN_GPIO_PIN, &led_green_config);
        //Initialize the GPIO Pins
}


/*******************************************************************************/
/* Function name:led_switch(int n )
 * Parameters: current state n
 * Return : void
 * Description: Function to initialize the GPIO RGB Led Pins . */
/*******************************************************************************/
void led_switch(int n)
 {
        GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN); //Clear the Pins
        GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
        GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);

        switch (n)
        {
        // Switch LED BLUE ON and TURN OTHER LEDs OFF
        case 0:
                {
                GPIO_ClearPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
                GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
                GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
                delay(1000);
                }
                break;
                // Switch LED RED ON and TURN OTHER LEDs OFF
        case 1:
        {

                        GPIO_ClearPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
                        GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
```

```
                        GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
                        delay(1000);
        }
                        break;
                        // Switch LED GREEN ON and TURN OTHER LEDs OFF
        case 2:
        {
                        GPIO_ClearPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
                        GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
                        GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
                        delay(1000);
        }
                        break;
        case 3:
                {
                        // Switch LED BLUE ON and TURN OTHER LEDs OFF
                GPIO_ClearPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
                GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
                GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
                }
                break;
        }
 }


/*************************************************************************************************
 * Function Name:int delay(int time_ms)
 * Description : this function provides delay in milliseconds according to input parameters
 * @input:time in milliseconds
 * @Return : NULL
 ************************************************************************************************/

void delay(int time_ms)
{
        volatile uint32_t i = 0;
        for (i = 0; i < 2400*time_ms; ++i)
        {
                __asm("NOP"); /* No operation */
        }
}

/*************************************************************************/
/* Function name:RGB_off
 * Parameters: void
 * Return : void
 * Description: Function to turn off the RGB Led Pins . */
/*************************************************************************/

void RGB_OFF()
{       // Clear all the  LEDs.
        GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
        GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
        GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
}
```

```c
/*
 * i2c_interrupt.c
 *
 *  Created on: Oct 29, 2019
 *      Author: SURAJ THITE
 *
 */


#include "i2c.h"
#include "logger.h"

extern events event1;
extern states current_state1;
extern int _15seconds_counter;
extern uint8_t averge_15_wait;

#define I2C_M_START      I2C0->C1 |= I2C_C1_MST_MASK
#define I2C_M_STOP       I2C0->C1 &= ~I2C_C1_MST_MASK
#define I2C_M_RESTART    I2C0->C1 |= I2C_C1_RSTA_MASK


#define I2C_TRANS                I2C0-> C1 |= I2C_C1_TX_MASK
#define I2C_REC          I2C0-> C1 &= ~I2C_C1_TX_MASK

#define I2C_WAIT             while((I2C0->S & I2C_S_IICIF_MASK) ==0)      {} \
                I2C0->S |= I2C_S_IICIF_MASK;

#define ACK I2C0->C1 &= ~I2C_C1_TXAK_MASK
#define NACK I2C0->C1 |= I2C_C1_TXAK_MASK



#define POST_CONFIG_REGISTER_VALUE0 96

#define POST_CONFIG_REGISTER_VALUE1 160


const uint8_t TMP102_addr = 0x90U; //TMP102 Address

bool Negative_Temp_Alert;        //Flag for negative temperature alert
uint8_t _15_seconds_counter;     //Flag for 15 seconds counter
bool mutex;
extern modes a;
extern fnnames fn_name;

uint8_t _15seconds_passed =0;
/*****************************************************************************************************
 * Function Name:void i2c_init(void)
 * Description :This Function Initializes the I2C bus
 * @input: void
 * @Return : Void
 *****************************************************************************************************/
void i2c_init(void)
{

        //ENable Clock Gating
        SIM->SCGC4 |= SIM_SCGC4_I2C0_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;

        //Set Pins for I2C functionality
        PORTC->PCR[8] |=  PORT_PCR_MUX(2);
        PORTC->PCR[9] |=  PORT_PCR_MUX(2);


        //Frequency Divider
        I2C0->F = I2C_F_ICR(0x11) | I2C_F_MULT(0);
        I2C0->C1 |= I2C_C1_IICEN_MASK;
```

```
        //Select High Drive Mode by setting HDRS bit in Control Register 2
        I2C0->C2 |= (I2C_C2_HDRS_MASK);

}

/*****************************************************************************************************
 * Function Name:void i2c_write(uint8_t addr, uint8_t reg,uint8_t data)
 * Description :This function writes data to the register pointed by addr address
 * @input: device address, register address, data
 * @Return : Void
 *****************************************************************************************************/

void i2c_write(uint8_t addr, uint8_t reg,uint8_t data)
{
        //Start transmission
        I2C_TRANS;
        I2C_M_START;

        //Send the device address to the i2c bus
        I2C0->D = addr;

        //wait for ack from slave
        I2C_WAIT

        //Write register address to i2c bus
        I2C0->D =reg;
        I2C_WAIT

        //Write data to the i2c bus
        I2C0->D = data;
        I2C_WAIT

        //Stop transaction
        I2C_M_STOP;
}

/*****************************************************************************************************
 * Function Name:int16_t i2c_read_temperature(uint8_t dev, uint8_t reg)
 * Description :This function returns the temperature value recieved from TMP102 module via I2C
Transactions
 * @input: device address , register address
 * @Return : Void
 *****************************************************************************************************/
int16_t i2c_read_temperature(uint8_t dev, uint8_t reg)
{
        fn_name = i2creadtemperature;
        int16_t tempc;
        uint8_t data[2];

        //Start I2C write
        I2C_TRANS;
        I2C_M_START;
        I2C0->D = dev;
        i2c_dealy1();

        I2C0->D =reg;
        i2c_dealy1();

        I2C_M_RESTART;
        I2C0->D =(dev |0x01);
        i2c_dealy1();

        //Start I2C read
        I2C_REC;
        ACK;

        data[0]=I2C0->D; //Store data in a variable
        i2c_dealy1();
```

```
	data[0]=I2C0->D;
	i2c_dealy1();    //Dummy read

	NACK;

	data[1] = I2C0->D;       //store data in a uint8 variable
	i2c_dealy1();

	//IF MSB is SET the negative value and set ALert flag
	if ((data[0] >>7 ) ==1)
	{
		Negative_Temp_Alert =1;
	}

	//Stop transaction
	I2C_M_STOP;
	//printf("/n /r %d ......%d",data[0],data[1]);


	data[1] = data[1] >> 4; // Right shift by 4 bits

	tempc = ((data[0] << 4) | data[1]);      //Store the result in a temporary variable
	event1 = COMPLETE;       //set read complete event

	//Return Negtive temperature value
	if(Negative_Temp_Alert)
	{
		Negative_Temp_Alert =0;
		return ((257-(tempc*0.0625))*(-1));

	}

	//Return Positive temperature value
	return (tempc*0.0625);
}
/*****************************************************************************************************
 * Function Name:Init_SysTick(void)
 * Description :This function Initializes the SysTick Timer for 15 seconds interrupts.
 * @input: void
 * @Return : Void
 *****************************************************************************************************/
void Init_SysTick(void)
{
	SysTick->LOAD = (48000000L/5);
	NVIC_SetPriority(SysTick_IRQn,3);               //Enable NVIC Interrupt with priority 3
	SysTick->VAL=0;
	SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;       //Enable interrupt
}

/*****************************************************************************************************
 * Function Name:void POST_TEST_read_bytes(uint8_t dev, uint8_t reg)
 * Description :This function RUNS POST TESTS by reading value in the CONFIGURATION REGISTER VALUES
 * @input: device address,register address
 * @Return : Void
 *****************************************************************************************************/
void POST_TEST_read_bytes(uint8_t dev, uint8_t reg)
{

	fn_name = POST_TESTread_bytes;
	uint8_t data[2];
	if(a!=Debug)
		Log_String(a,fn_name,"POST-----");

	I2C_TRANS;

	I2C_M_START;

	I2C0->D = dev;
```

```
        i2c_dealy1();

        I2C0->D =reg;

        i2c_dealy1();

        I2C_M_RESTART;

        I2C0->D =(dev |0x01);

        i2c_dealy1();

        I2C_REC;
        ACK;

        data[0]=I2C0->D;
        i2c_dealy1();

        data[0]=I2C0->D;
        i2c_dealy1();

        NACK;

        data[1] = I2C0->D;
        i2c_dealy1();

        I2C_M_STOP;

        if(data[0]==POST_CONFIG_REGISTER_VALUE0 && data[1] ==POST_CONFIG_REGISTER_VALUE1)  // Check for
POST Default values in the config register
        {
                if(a!=Debug)
                {
                        Log_String(a,fn_name,"POST SUCCESSFULL ------POST CONFIG REGISTER VALUES---C0 ---
C1 are: ");
                        Log_integer(a, data[0]); //Print Config register values
                        Log_integer(a, data[1]); //Print COnfig register value
                }
                else
                        Log_String(1,fn_name,"POST PASS : LED set to green");
                if (event1 != TIME_OUT_4)
                        event1=PASS;    //Set event to PASS

        }
        else
        {
                Log_String(a,fn_name,"POST FAILED");
                current_state1 = END;    //Set state to END
                if (event1 != TIME_OUT_4)
                        event1=FAIL;    //  event to fail
                else
                        return FAIL;
        }
}

/*****************************************************************************************************
 * Function Name:uint8_t check_connection(uint8_t dev, uint8_t reg)
 * Description :This function checks whether TMP102 device is active on I2C bus and returns connection
status
 * @input: device address,register address
 * @Return : Connection Status
 *****************************************************************************************************/

uint8_t check_connection(uint8_t dev, uint8_t reg)
{

        //int16_t tempc;
        uint8_t data[2];
        //bool disconnect_flag;
```

```
        uint8_t temp;
        uint8_t total=0;
        //Start I2C write
        I2C_TRANS;
        I2C_M_START;
        I2C0->D = dev;

        temp = i2c_dealy1();
        if (temp==1)
        total = total + temp;
        I2C0->D =reg;

        temp = i2c_dealy1();
        total = total + temp;

        I2C_M_RESTART;
        I2C0->D =(dev |0x01);

        temp = i2c_dealy1();
        total = total + temp;

        //Start I2C read
        I2C_REC;
        ACK;

        data[0]=I2C0->D; //Store data in a variable

        temp = i2c_dealy1();
        total = total + temp;

        data[0]=I2C0->D;

        temp = i2c_dealy1();
        total = total + temp;

        NACK;

        data[1] = I2C0->D;

        temp = i2c_dealy1();
        total = total + temp;

        I2C_M_STOP;
        printf("\n \r ***********CONNECTION HEALTH*********: %d",total);
        if (total >1)
        {
                return 1;
        }
        else return 0;
}


/*****************************************************************************************************
 * Function Name:void SysTick_Handler()
 * Description :This function is the IRQ Handler
 * This function generates event for TIMEOUT_1_2_3 and TIMEOUT4 for changing states to READ TEMPERATURE
AND
 * SWITCH STATE MACHINE
 * @input: void
 * @Return : void
 *****************************************************************************************************/
//Event handler for SystickTimer for 15 seconds delay
void SysTick_Handler()
{
        _15seconds_passed = _15seconds_passed +1;
        fn_name = SysTickHandler;
        _15_seconds_counter = _15_seconds_counter +1 ;
        if (_15_seconds_counter < 4)
        {
```

```
                event1 =TIME_OUT_1_2_3;
        }
        else
        {
                event1 = TIME_OUT_4;
                current_state1=STATE_MACHINE_SWITCH;
                _15_seconds_counter =0;
        }
        //reset in else condition for second state machine
}

/*****************************************************************************************************
 * Function Name:uint16_t CONFIG_REGISTER_VALUE()
 * Description :This function returns current config register value
 * @input: void
 * @Return : void
 *****************************************************************************************************/
uint16_t CONFIG_REGISTER_VALUE()
{
        uint8_t dev =0x90;
        uint8_t reg =0x01;
        uint8_t data[2];
        uint16_t temp;
        I2C_TRANS;
        I2C_M_START;
        I2C0->D = dev;
        I2C_WAIT;

        I2C0->D =reg;
        I2C_WAIT;

        I2C_M_RESTART;
        I2C0->D =(dev |0x01);
        I2C_WAIT;

        I2C_REC;
        ACK;

        data[0]=I2C0->D;
        I2C_WAIT;

        data[0]=I2C0->D;
        I2C_WAIT;

        NACK;

        data[1] = I2C0->D;
        I2C_WAIT;

        I2C_M_STOP;

        temp = (data[0]*256 + data[1]);

        return temp;
}

/*****************************************************************************************************
 * Function Name:uint8_t i2c_dealy1()
 * Description :This function returns the connection status of the i2c connection
 * @input: void
 * @Return : connection status
 *****************************************************************************************************/
uint8_t i2c_dealy1()
{
        uint16_t temp=0;
        while((I2C0->S & I2C_S_IICIF_MASK) ==0)
        {
                temp++;
                if(temp > 250 )
```

```
                {
                        break;

                }

        }
        I2C0->S = I2C0->S | I2C_S_IICIF_MASK;
        if(temp>250)
        {
                //printf("\n \r CONNECTION FAILED : %d",temp);
                return 1;
        }
        else
        {
                //printf("\n \r CONNECTION PASSED :%d",temp);
                return 0;

        }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "logger.h"
#include "main.h"

modes a = Test;

fnnames fn_name;

////////TABLE for modes and current states//////////////////////////////

uint8_t flag;
////// Logger for integer ///////////

/*********************************************************************************************
 * Function Name: Log_integer(uint32_t intval)
 * Description : This function prints the integer value to the serial terminal
 * @input: integer to be printed
 * @Return : void
 *********************************************************************************************/
void Log_integer(modes current_mode,int16_t intval)
{
        if(current_mode != 1)      // since no integers to print in normal status mode
        printf("%d ",intval); // Print the  data
}


/*********************************************************************************************
 * Function Name: Log_string(char* str)
 * Description : This function prints the string pointed by the input argument
 * @input:  pointer from which string  to be printed
 * @Return : void
 *********************************************************************************************/
////// Logger for string ///////////
void Log_String(uint8_t current_mode,fnnames mycurrent_function, char* str)
{
        if (current_mode ==0)
        {
            printf("\n \r -->Test Mode: \t");
            //call Unittest();
        }
         if (current_mode ==1)
        {
                printf("\n \r -->Debug Mode: \t");
                /*if(mycurrent_state1==0)
                {
                        printf("POST_STATES: \t Running Power on Startup test \n");
                }
                if(mycurrent_state1==1)
            {
                        printf("I2C_READ_TEMPERATURE:\t ENTERED <STATE I2C_READ_TEMPERATURE> \n CURRENT
TEMP VALUE :");
                }
                if(mycurrent_state1==2)
                {
                        printf("AVERAGE_WAIT:\t ENTERED <AVERAGE WAIT> \n \r AVERAGE TEMPERATURE VALUE
FOR ITERATION: ");
                }
                if(mycurrent_state1==3)
                {
                        printf("TEMP_ALERT:\t Led set to red \n ***********ALERT : TEMPERTURE BELOW 0
DEGREE C ************** ");
                }*/

        }
         if (current_mode ==2)
        {
            printf("\n \r -->Normal Mode: \t");
```

```
            /*if(mycurrent_state1==3)
            {
                printf("TEMP_ALERT:\t Led set to red \n ************ALERT : TEMPERTURE BELOW 0 DEGREE C
************** ");
            }*/
        }
        if (mycurrent_function==state_machine1)
        {
                printf(" -->Function name: state_machine_1 \n");
        }
        else if (mycurrent_function==i2cwrite)
        {
                printf(" -->Function name: i2c_write \n");
        }
        else if (mycurrent_function==i2creadtemperature)
        {
            printf("-->Function name: i2c_read_temperature \n");
        }
        else if (mycurrent_function==POST_TESTread_bytes)
        {
                printf("Function name: POST_TEST_read_bytes \n");
        }
        else if (mycurrent_function==checkconnection)
        {
                printf("Function name: check connection \n");
        }
        else if (mycurrent_function==SysTickHandler)
        {
                printf("Function name: SysTick_Handler \n");
        }
        else if (mycurrent_function==InitSystick)
        {
                printf("Function name: InitSystick \n");
        }
        else if (mycurrent_function==state_machine2)
        {
                printf("Function name: state_machine2 \n");
        }
        else if (mycurrent_function==state_transition__intable)
        {
                printf("Function name: state_transition__intable \n");
        }
        else if (mycurrent_function==POSTSTATE)
        {
                printf("Function name: POSTSTATE \n");
        }
        else if (mycurrent_function==I2C_READ_TEMPERATURESTATE)
        {
                printf("Function name: I2C_READ_TEMPERATURESTATE \n");
        }
        else if (mycurrent_function==    AVERAGE_WAITSTATE)
        {
                printf("Function name:   AVERAGE_WAITSTATE \n");
        }
        else if (mycurrent_function==    TMEP_ALERTSTATE)
        {
                printf("Function name:   TMEP_ALERTSTATE \n");
        }
        else if (mycurrent_function==    DISCONNECTEDSTATE)
        {
                printf("Function name:   DISCONNECTEDSTATE \n");
        }
        else
        {
                printf("Function name:   ENDSTATE \n");
        }

           printf("\n \r %s ",str);
```

```
}


/*****************************************************************************************************
 * Function Name: Log_data(uint32_t * loc,uint32_t length)
 * Description : This function  is used to log the address and variable value stored at it.
 * @input:  pointer to the memory location and the length
 * @Return : void
 *****************************************************************************************************/
////// Logger for data ///////////
void Log_data(uint32_t * loc,uint32_t length)
{
        if(flag==1) {
                int i=0;
                printf("\n\rThe Address is %p  ",loc);   //Print the address of the memory block pointed
by the pointer
                if (length !=0)
                {
                        printf(" and data is");
                }
                for(i=0;i<length;i++)
                {
                        printf(" %x",*(loc+i));  //Print the data stored at the memory location
                }

        }
}


////// Logger test, debug and normal ///////////

/*****************************************************************************************************
 * Function Name:void Log_test()
 * Description : This function .enables the logging for the system
 * @input:  void
 * @Return : void
 *****************************************************************************************************/
/*
////// Logger set flag for test mode  ///////////
void Log_test()
{
        flag =0;
}

*****************************************************************************************************
 * Function Name: Log_debug()
 * Description : This function .disables the logging for the system
 * @input:  void
 * @Return : void
 *****************************************************************************************************
////// Logger set flag for debug mode ///////////
void Log_debug()
{
        flag =1;
}

////// Logger set flag for normal mode ///////////
void Log_normal()
{
        flag =2;
}
*/

/*****************************************************************************************************
 * Function Name:uint8_t Log_status()
 * Description : This function  enables whether log is enabled or disabled for the system
 * @input:  void
 * @Return : flag of uint8_t
 *****************************************************************************************************/
```

```
/*////// Logger for data ///////////
uint8_t Log_status()
{
        return flag;
}*/

///////////////Log level//////////////////////////
uint8_t Log_level()
{

        return a;
}
```

```
/*
```

```
 * i2c.h
 *
 *  Created on: Nov 2, 2019
 *      Author: SURAJ THITE
 */

#ifndef I2C_H_
#define I2C_H_

#include "state_machine.h"
#include "main.h"


extern int16_t temp_val;



void i2c_init(void);
void i2c_write(uint8_t addr, uint8_t reg,uint8_t data);
int16_t i2c_read_temperature(uint8_t dev, uint8_t reg);
void POST_TEST_read_bytes(uint8_t dev, uint8_t reg);
void Init_SysTick(void);
uint8_t check_connection(uint8_t dev, uint8_t reg);
uint16_t CONFIG_REGISTER_VALUE();
uint8_t i2c_dealy1();



#endif /* I2C_H_ */
```

```h
#ifndef LOGGER_H_
#define LOGGER_H_

void Log_enable();
void Log_disable();
uint8_t Log_status();
void Log_data (uint32_t *, uint32_t );
void Log_String(uint8_t ,uint8_t, char *);
void Log_integer(uint8_t ,int16_t);
uint8_t Log_level();

#endif
```

```c
/*
 * main.h
 *
 *  Created on: Nov 2, 2019
 *      Author: SURAJ THITE
 */

#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "logger.h"
#include "i2c.h"
#include "state_machine.h"


///////////////ENUM for test, debug and status//////////////////
typedef enum
{
        Test,
        Debug,
        Status
}modes ;
/////////////////////////////////////////////////////////////

typedef enum
{
        state_machine1,
        i2cwrite,
        i2creadtemperature,
        POST_TESTread_bytes,
        checkconnection,
        SysTickHandler,
        InitSystick,
        state_machine2,
        state_transition__intable,
        POSTSTATE,
        I2C_READ_TEMPERATURESTATE,
        AVERAGE_WAITSTATE,
        TMEP_ALERTSTATE,
        DISCONNECTEDSTATE,
        ENDSTATE

}fnnames;

/////////////////////////////////////////////////////////////////

#endif /* MAIN_H_ */
```

```c
/*
 * RGBled.h
 *
 *  Created on: Sep 28, 2019
 *   Author:SURAJ THITE , ATHARV DESAI
 */

#ifndef RGBLED_H_
#define RGBLED_H_
void led_switch(int n);  //Function to switch the led_state
void RGB_init();//Function to initialize the RGB Leds
void RGB_OFF();  //Function to turn off the RGB led off
void delay(int time_ms); // Delay
#endif /* RGBLED_H_ */
```

```c
/*
 * state_machine.h
 *
 *  Created on: Nov 2, 2019
 *      Author:  SURAJ THITE
 */

#ifndef STATE_MACHINE_H_
#define STATE_MACHINE_H_

#include "main.h"
#include "i2c.h"

typedef enum
{
        POST_STATES,
        I2C_READ_TEMPERATURE,
        AVERAGE_WAIT,
        TEMP_ALERT,
        DISCONNECT,
        STATE_MACHINE_SWITCH,
        END
}states;


typedef enum
{
        null_event,
        INIT,
        PASS,
        FAIL,
        ALERT,
        COMPLETE,
        DISCONNECT_EVENT,
        TIME_OUT_1_2_3,
        TIME_OUT_4,
        EXIT
}events;

struct transition
{
        states current_state2;
        events event2;
        states next_state_2;
};



events POST_STATE(void);
events I2C_READ_TEMPERATURE_STATE(void);
events AVERAGE_WAIT_STATE(void);
events TMEP_ALERT_STATE(void);
events DISCONNECTED_STATE(void);
events END_STATE(void);


//current_state = POST_STATES;
void state_machine_1();
void state_machine_2(states current_state2);
states table_tansition( states CR, events EC);
states state_transition__in_table(states,events);


#endif /* STATE_MACHINE_H_ */
```