```
/*****************************************************************************
                                    PES PROJECT 4
                    AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                    Cross Platform IDE: MCUXpresso IDE v11
                    Cross-Compiler: ARM GCC
                    PROJECT.C
*****************************************************************************/
/**************Header files**************/
#include <stdint.h>
#include <stdio.h>
#include "TMP102.h"
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_device_registers.h"
#include <time.h>
#include "led.h"
#include "logger.h"
#include <state_driven.h>
#include "UCunit.h"
#include <string.h>
#include "unittest.h"

/***********global variables**********/
uint8_t post;
uint8_t disc_buffer;
uint8_t disc_value;

/**********alert pin function***********/
void set_ptrreg(void)
{
        write_byte(0x90,0x01,0x62);
        write_byte(0x90,0x01,0xA0);
        write_byte(0x90,0x02,0x28);
        write_byte(0x90,0x02,0x00);

}
/****function and mode enumerations*******/
enum functions{
        temp_reading=0,
        temp_alert,
        temp_average,
        temp_disconnected
}f;
```

```
typedef enum {
        test=0,
        debug,
        status
}mode;
mode mo=0;

char ch_arr[40][40]={    "Temperature Reading mode",
                         "Temperature average",
                          "Temperature alert mode",
                           "Temperature Disconnect",
                            "LED is initialized",
                            "Switches to Other State Machine"
                        };
int main(void)
{

    BOARD_InitBootPins();
   BOARD_InitBootClocks();
   BOARD_InitBootPeripherals();
   BOARD_InitDebugConsole();

   I2C_Master_Init();

   if(mo==0)
   {
        log_string("******TEST MODE ON *******");
              /***uCunit testing*******/
        unit_test();
    }
   else if(mo==1)
   {
        log_string("******DEBUG MODE ON **********");
   }
   else if(mo==2)
   {
        log_string("******NORMAL/STATUS MODE ON *******");
   }

/*****check post*****/
   post = post_condition();

   if(post == 0x60 || post == 0x62)
   {
        log_string("\n \r POST SUCCESSFUL");
        //set_ptrreg();
        State_Driven();                    /****State driven state machine**/
   }
```

```
  else
  {
        log_string("\n \r POST UNSUCCESSFUL");
        handle_disconnect();
  }

  return 0 ;
}

/*************************************************************************
                                PES PROJECT 4
              AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                      Cross Platform IDE: MCUXpresso IDE v11
                       Cross-Compiler: ARM GCC (FB version)
                                state_driven.c
*************************************************************************/
/**************Header files**************/

#include <stdint.h>
#include <stdlib.h>
#include "state_driven.h"
#include "TMP102.h"
#include <stdio.h>
#include "tabledriven.h"
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "led.h"
#include "logger.h"
#include "fsl_device_registers.h"

/*****global Declaration******/
uint8_t element=0;
uint8_t timeout_var=0;
uint8_t disp_ele=0;
uint8_t arr[100];
uint8_t var_print=1;
uint8_t average_read=0,total_read=0;
uint8_t neg_temp;
uint8_t timeout_table;
extern uint8_t post;
extern uint8_t disc_value;
extern uint8_t disc_buffer;
uint8_t time_var=0;


typedef enum
      {
            temp_reading , temp_average, temp_alert, temp_disconnected,LED_on
```

```
        }tstate;


        tstate r = temp_reading;

        enum modes{
                test=0,
                debug,
                status
        }m;

/***Delay for 15sec***/
        void delay(uint32_t d)
        {
                uint32_t count = d*3450;   /***7000****** As clock is 8MHz *****/
                while(count!=0)
                {
        count--;
                }
        }

enum modes m=0;
/******Power on Self Test**************/
int post_condition(void)
{

        uint8_t post_buffer;
        post_buffer = read_byte(0x90,0x01);
        extern uint8_t actual;
        actual=post_buffer;
        return post_buffer;

}
/************Disconnect Condition*********/
//int disc_condition(void)
//{
//      disc_buffer = read_byte(0x90,0x01);
//      return disc_buffer;
//}


/*****Temperature read function*********/
uint8_t handle_timeout(void)
{

        log_level(m,0);             //Log_level
        init_LED();                 //Initialize LED
        led_wait();
        log_level(m,4);
        disc_value=read_byte(0x90,0x01);
        log_string("checks the disconnect condition");
        if (disc_value == 0x00)    /****disconnect condition***/
                        {
                        return temp_disconnected;
                        }
```

```
	else
	{
		if(timeout_var<4)
		{
			arr[element] = read_byte(0x90,0x00);    //taking 1st temp reading

			log_string("the temp value is:");
			log_integer(arr[element]);

			if(m==1||m==2){
				log_string("\t \t \t \t Timeout ");
				log_integer(var_print);
			}
		var_print++;
		timeout_var++;
		//disc_value=disc_condition();
		/**for temperature greater than 0***/
		if(arr[element]>0)
		{
			return temp_average;
		}
		else if(arr[element]<=0)  /**for temperature less than 0***/
		{
		return temp_alert;
		}

		}
	}
}
/***********Temperature Average***/
uint8_t handle_complete(void)
{
	log_level(m,1);
	init_LED();
	led_wait();
	log_level(m,4);

disc_value=read_byte(0x90,0x01);
log_string("checks the disconnect condition");
if (disc_value == 0x00)   /****disconnect condition***/
{
	return temp_disconnected;
}
else
{

	if(arr[element]>0)
	{
		//disc_value = disc_condition();
		if(timeout_var == 4)
			{
					timeout_var=0;
				Table_Driven();
			}
		else{
```

```
if(m==1||m==2)
{
log_string("The last temperature reading  ");
log_integer(disp_ele);
log_string(" is: ");
log_integer(arr[disp_ele]);
}
total_read=0;
for(uint8_t i=0;i<timeout_var;i++)                              // for average
{
  total_read = total_read + arr[i];
}
average_read= total_read/(timeout_var);
if(m==1||m==2)
{
    log_string("The current average temperature reading is:");
    log_integer(average_read);
    log_string("\n");
   }
element++;
  delay(10000);
disp_ele++;

return temp_reading;
        }
  }

  else if (arr[element]<=0)
  {
        //disc_value = disc_condition();
        log_level(m,1);
        init_LED();
        led_wait();
        log_level(m,4);
        if(m==1||m==2)
        {
        log_string("The last temperature reading is:\t ");
        log_integer(disp_ele);
        log_integer(arr[disp_ele]);
        }
        timeout_var++;
        total_read=0;
        for(uint8_t i=0;i<timeout_var;i++)
        {
          total_read = total_read + arr[i];
        }
        //  total_read = total_read + (neg_temp);
          average_read= total_read/(timeout_var);
          if(m==1||m==2)
         {
     log_string("The current average temperature reading is:\t");
     log_integer(average_read);
         }
          element++;
          delay(10000);
```

```
                disp_ele++;
                if(timeout_var == 4)
                {
                        timeout_var=0;
                }
                return temp_disconnected;
        }
}
}
/**********Temperature alert condition*******/
uint8_t handle_alert(void)
{
        //disc_value = disc_condition();
        log_level(m,2);
        init_LED();
        led_alert();
        log_level(m,4);
        disc_value=read_byte(0x90,0x01);
        log_string("checks the disconnect condition");
        if(disc_value==0x00)
        {
                return temp_disconnected;
        }
        else
        {
        log_level(m,2);
        init_LED();
        led_alert();
        log_level(m,4);
        neg_temp = arr[element];
        log_integer(arr[element]);
        log_string(" is the negative temperature received");

        return temp_average;
        }
}

uint8_t handle_disconnect(void)
{
        log_level(m,3);
        init_LED();
        led_error();
        log_level(m,4);
        disc_value = read_byte(0x90,0x00);
        log_string("Connection has been disconnected");
        exit(0);
}

void State_Driven(void)
{
        tstate r = temp_reading;
        timeout_var = 0;
        element = 0;
        disp_ele = 0;
        average_read = 0;
```

```
	var_print = 1;
	log_string("*************STATE DRIVEN STATE MACHINE BEGINS***********");

while(1)
  {
	switch(r)
	{

	case temp_reading:
                                        r =  handle_timeout();
                                        break;
	case temp_average:
                                    r = handle_complete();
                                    break;

	case temp_alert:
                                    r = handle_alert();
                                    break;

	default:
                                          r = handle_disconnect();
                                          break;

	}
}

}


/*********************************************************************************
                                  PES PROJECT 4
               AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                      Cross Platform IDE: MCUXpresso IDE v11
                       Cross-Compiler: ARM GCC (FB version)
                                  state_driven.h
*********************************************************************************/

#include <stdint.h>

#ifndef STATE_DRIVEN_H_
#define STATE_DRIVEN_H_

uint8_t handle_timeout(void);
uint8_t handle_complete(void);
uint8_t handle_alert(void);
uint8_t handle_disconnect(void);
void State_Driven(void);

int post_condition(void);

int disc_condition(void);



#endif
```

```c
/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                    tabledriven.c
*****************************************************************************/
/**************Header files**************/

#include "tabledriven.h"
#include <stdint.h>
#include <stdlib.h>
#include "state_driven.h"
#include "TMP102.h"
#include <stdio.h>
#include "tabledriven.h"
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "led.h"
#include "logger.h"
#include "fsl_device_registers.h"

/**************Global declaration**************/
uint8_t telement=0;
uint8_t ttimeout_var=0;
uint8_t tdisp_ele=0;
uint8_t tarr[100];
uint8_t tvar_print=1;
uint8_t taverage_read=0,ttotal_read=0;
uint8_t tneg_temp;
uint8_t ttimeout_table;
uint8_t ttime_var=0;
//extern uint8_t arr[100];
extern uint8_t post;
extern uint8_t disc_value;

int Sm_tempread(void);
int Sm_tempavg(void);
int Sm_tempalert(void);

enum module{
      test=0,
      debug,
      status
}md;
enum module md=0;

/*******Disconnect*********/
int Sm_tempdisc(void)
{
```

```
	//log_string("Connection has been disconnected");
	log_level(md,3);
		init_LED();
		led_error();
		log_level(md,4);
	_exit(0);
	return 0;
}


typedef char tevent;

typedef enum state
{
	temp_reading , temp_average, temp_alert , temp_disconnected

}tstate;



typedef struct
{
	tstate state;
	tevent trigger_event[30];
	int (*func_ptr)(void);

}StateMachinetype;

StateMachinetype StateMachine[10] =
{
		{temp_reading, "completeevent",&Sm_tempavg},
		{temp_average, "timeoutevent", &Sm_tempread},
		{temp_reading, "alertevent", &Sm_tempalert},
		{temp_alert, "completeevent", &Sm_tempavg},
		{temp_alert, "disconnectevent", &Sm_tempdisc},
		{temp_reading, "disconnectevent", &Sm_tempdisc},
		{temp_average, "disconnectevent", &Sm_tempdisc},
		{temp_disconnected,"disconnectevent", &Sm_tempdisc}
};

StateMachinetype *s_p = &StateMachine[0];

void Table_Driven()
{
	telement=0;
	ttimeout_var=0;
	tdisp_ele=0;
	tvar_print=1;
	taverage_read=0;
	ttotal_read=0;
	ttime_var=0;
	log_string("/*************************TABLE driven state Machine
Begins**********************/");
	Sm_tempread();
}
```

```
/*****Temperature read function*********/
int Sm_tempread(void)
{

disc_value=read_byte(0x90,0x01);
log_string("checks the disconnect condition");
if (disc_value == 0x00)   /****disconnect condition***/
{
            Sm_tempdisc();
}
else
{
//uint8_t i=0;
      if(ttimeout_var<4)
/************ timeout 0,1,2,3*****************************************/
      {
            log_level(md,0);
            init_LED();
            led_wait();
            log_level(md,4);
            //     disc_value = disc_condition();
                  tarr[telement] = read_byte(0x90,0x00); // repeat taking 1st temp
reading
            //     arr[telement]=arr[telement]+arr[telement]*0.0625;
                  log_string("The temperature value is");
                  log_integer(tarr[telement]);
                  if(md==1||md==2){
                  log_string("\t \t \t timeout");
                  log_integer(tvar_print);        //variable to print 1st, 2nd,
3rd timeout
                  }
                  tvar_print++;
                  ttimeout_var++;

                  if(tarr[telement]>0)
/******************if temp is greater than 0*************************/
                  {
                  tstate state = temp_reading;
                  tevent trigger_event[30] = "completeevent";

                  for(uint8_t j=0;j<8;j++)
                  {
                                          if (StateMachine[j].state==state &&
trigger_event)
                                                {


                                                      s_p = &StateMachine[j];
                                                      (*(s_p->func_ptr))();
                                                }

                                    }
                  }
```

```
                    else if(tarr[telement]<=0)
/*************for alert event*******************************/
                    {
                    tstate state = temp_reading;
                    tevent trigger_event[30] = "alertevent";
                          for(uint8_t i=0;i<8;i++)
                                {
                                if(StateMachine[i].state==state && trigger_event)
                                            {
                                            s_p = &StateMachine[i];
                                                  (*(s_p->func_ptr))();
                                            }
                                }
                    }

             }
else
/**********if timeout var is equal to 4******************/
{
                          tstate state = temp_average;
                          tevent trigger_event[30] = "completeevent";
                          for(uint8_t i=0;i<8;i++)
                                {
                                if( StateMachine[i].state == state && trigger_event)
                                      {
                                      s_p = &StateMachine[i];
                                      (*(s_p->func_ptr))();
                                      }
                                }
}
}
      return 0;
}

/*****Temperature average function*********/
int Sm_tempavg(void)
{
                    log_level(md,1);
                    init_LED();
                    led_wait();
                    log_level(md,4);
//disc_value = disc_condition();
disc_value=read_byte(0x90,0x01);
log_string("checks the disconnect condition");
if (disc_value == 0x00)   /****disconnect condition***/
{
      Sm_tempdisc();
}
else
{
      if(tarr[telement]>0)
      /***************when temp is greater than zero********************/
      {
                    if(ttimeout_var == 4)
      /******************checks if 4th timeout has occured*************/
```

```
                    {
                    log_string("last timeout var value");
                    log_integer(ttimeout_var);
                    post = post_condition();
/***********calling POST condition func***************/
                    log_string("value is");
                    log_integer(post);

                          if(post == 0x60 || post == 0x62)
                                {
                                  log_string("\n \r POST SUCCESSFUL");
                                  //set_ptrreg();
                                 // Table_Driven();
                                  State_Driven();
                                  log_level(md,5);
                                 }
                          else
                                {
                                  log_string("\n \r POST UNSUCCESSFUL");
                                  handle_disconnect();
                                }

                    }
                    else
/********* if timeout is not equal to 4************/
                    {
                          log_string("the last temperture reading");
                          log_integer(tdisp_ele);
                          log_string("is:");
                          log_integer(tarr[tdisp_ele]);

                          ttotal_read=0;
                          for(uint8_t i=0;i<ttimeout_var;i++)
                          {
                          ttotal_read = ttotal_read + tarr[i];
                          //printf("\n \r %d is total read",ttotal_read);
                          }
                          taverage_read= ttotal_read/(ttimeout_var);
                          log_string("the current average temperature reading is:");
                          log_integer(taverage_read);
                          telement++;
                          delay(10000);
                          tdisp_ele++;
//variable to display last temperature reading
                    tstate state = temp_average;
                    tevent trigger_event[30] = "timeoutevent";
                    for(uint8_t c=0;c<8;c++)
                    {
                          if( StateMachine[c].state == state && trigger_event)
                          {
                                s_p = &StateMachine[c];
                                (*(s_p->func_ptr))();
                          }
                    }
   // return temp_reading;
```

```
                }
        }
        else if (tarr[telement]<=0)
/**************condition for when temperature is less than zero******************/
        {               if(md==1||md==2)
                {
                log_string("the last temperture reading");
                log_integer(tdisp_ele);
                log_string("is:");
                log_integer(tarr[tdisp_ele]);
                }
                ttimeout_var++;
                ttotal_read=0;
                for(uint8_t m=0;m<ttimeout_var;m++)
                {
                  ttotal_read = ttotal_read + tarr[m];
                }
                //  ttotal_read = ttotal_read + (tneg_temp);
                  taverage_read= ttotal_read/(ttimeout_var);
                  if(md==1||md==2){
                  log_string("the current average temperature reading is:");
                  log_integer(taverage_read);
                  }
                  telement++;
                  delay(10000);
                  tdisp_ele++;
                  tstate state = temp_disconnected;
                  tevent trigger_event[30] = "disconnectevent";
                  for(uint8_t b=0;b<8;b++)
                  {
                       if( StateMachine[b].state == state && trigger_event[30])
                       {
                       s_p = &StateMachine[b];
                       (*(s_p->func_ptr))();
                       }
                  }
        }

}
        return 0;
}

/*****Temperature alert function*********/
int Sm_tempalert(void)
{       log_level(md,2);
        init_LED();
        led_wait();
        log_level(md,4);

disc_value=read_byte(0x90,0x01);
log_string("checks the disconnect condition");
if (disc_value == 0x00)   /****disconnect condition***/
{
        Sm_tempdisc();
}
```

```
else
{
	tneg_temp = tarr[telement];
	if(md==1||md==2){
	log_string("Negative temperature value is");
	log_integer(tarr[telement]);
	}
}
	return 0;
}

/*********************************References*************************************
*
 1.https://kjarvel.wordpress.com/2011/10/26/table-driven-state-machine-using-
function-pointers-in-c/
 ********************************************************************************/

/*********************************************************************************
                                PES PROJECT 4
            AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                    Cross Platform IDE: MCUXpresso IDE v11
                     Cross-Compiler: ARM GCC (FB version)
                                tabledriven.h
*********************************************************************************/
#ifndef TABLE_DRIVEN_H_
#define TABLE_DRIVEN_H_

int Sm_tempdisc(void);
void Table_Driven();

#endif
/*********************************************************************************
                                PES PROJECT 4
            AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                    Cross Platform IDE: MCUXpresso IDE v11
                     Cross-Compiler: ARM GCC (FB version)
                                  TMP102.c
*********************************************************************************/
/*************Header files*************/

#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include <led.h>
#include "TMP102.h"
#include "fsl_device_registers.h"

void I2C_Master_Init(void)
```

```
{
	/* Enable clock for I2C0 module */
	SIM->SCGC4 |= SIM_SCGC4_I2C0_MASK;

	/* Enable clock for Port C */
	SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;


	/* Port C MUX configuration */
	PORTC->PCR[8] |=  PORT_PCR_MUX(2);
	PORTC->PCR[9] |=  PORT_PCR_MUX(2);


	/* Configure Divider Register */
	I2C0->F |= I2C_F_ICR(0x11) | I2C_F_MULT(0);

	/* Enable I2C module and interrupt */
	I2C0->C1 |= (I2C_C1_IICEN_MASK);

	/*select high drive mode*/
	I2C0->C2 |= (I2C_C2_HDRS_MASK);

	/* Enable TX mode */
	//I2C0_C1 |= I2C_C1_TX_MASK;

	/* Enable I2C0 NVIC interrupt */
	//Enable_irq(INT_I2C0 - 16);
}



void write_byte(uint8_t dev,uint8_t reg,uint8_t data)
{
	I2C_TRAN;
	I2C_M_START;
	I2C0->D = dev;
	I2C_WAIT;

	I2C0->D = reg;
	I2C_WAIT;

	I2C0->D = data;
	I2C_WAIT;
	I2C_M_STOP;
}


uint8_t read_byte(uint8_t dev,uint8_t reg)
{
	uint8_t data=0;
	//uint8_t i;
	I2C_TRAN;                // set to transmit mode in Control 1 register
	I2C_M_START;             // master mode select
```

```
//     for(i=0;i<3;i++)
//     {
//     I2C0->D = buffer[i] ;
//     I2C_WAIT;
//     }
       I2C0->D = dev;
       I2C_WAIT;

       I2C0->D = reg ;           //
       I2C_WAIT;

//     I2C0->D = buffer[i];
//     I2C_WAIT;

       I2C_M_RSTART;
       I2C0->D = (dev | 0x1);
       I2C_WAIT;

       I2C_REC;
       NACK;

       data = I2C0->D;
       I2C_WAIT;

       I2C_M_STOP;
       data = I2C0->D;

       return data;
}


/*********************************References*************************************
*
 1.ARM CORTEX M BASED MICROCONTROLLERS,DEAN 2017,ARM EDUCATION MEDIA
 ****************************************************************************/
/*****************************************************************************
                                  PES PROJECT 4
               AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                       Cross Platform IDE: MCUXpresso IDE v11
                        Cross-Compiler: ARM GCC (FB version)
                                    master.h
*****************************************************************************/

#ifndef TMP102_H_
#define TMP102_H_
#include "stdint.h"

#define SLAVE_ADDRESS 0x91

#define BUFFER_SIZE 10

#define WRITE0x00
#define      READ   0x01

#define I2C_M_START     I2C0->C1 |= I2C_C1_MST_MASK
```

```
#define I2C_M_STOP      I2C0->C1 &= ~I2C_C1_MST_MASK
#define I2C_M_RSTART    I2C0->C1 |= I2C_C1_RSTA_MASK
#define I2C_TRAN          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 NACK              I2C0->C1 |= I2C_C1_TXAK_MASK
#define ACK               I2C0->C1 &= ~I2C_C1_TXAK_MASK



void I2C_Master_Init(void);
void write_byte(uint8_t dev, uint8_t reg, uint8_t data);

uint8_t read_byte(uint8_t dev, uint8_t reg);

#endif

/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                      UCunit.h
*****************************************************************************/
/*****************************************************************************
 *                                    REFERENCE
*
 *  uCUnit - A unit testing framework for microcontrollers                   *
 *  (C) 2007 - 2008 Sven Stefan Krauss                                       *
 *                  https://www.ucunit.org                                   *
 *****************************************************************************/

#ifndef UCUNIT_0101_H_
#define UCUNIT_0101_H_

/*****************************************************************************/
/****** Customizing area ******/
/*****************************************************************************/
/**
 * @Macro:       UCUNIT_WriteString(msg)
 *
 * @Description: Encapsulates a function which is called for
 *               writing a message string to the host computer.
 *
 * @param msg:   Message which shall be written.
 *
 * @Remarks:     Implement a function to write an integer to a host
 *               computer.
 *
 *               For most microcontrollers a special implementation of
 *               PRINTF is available for writing to a serial
 *               device or network. In some cases you will have
 *               also to implement a putch(char c) function.
 */
```

```
#define UCUNIT_WriteString(msg)    System_WriteString(msg)

/**
 * @Macro:       UCUNIT_WriteInt(n)
 *
 * @Description: Encapsulates a function which is called for
 *               writing an integer to the host computer.
 *
 * @param n:     Integer number which shall be written
 *
 * @Remarks:     Implement a function to write an integer to a host
 *               computer.
 *
 *               For most microcontrollers a special implementation of
 *               PRINTF is available for writing to a serial
 *               device or network. In some cases you will have
 *               also to implement a putch(char c) function.
 */
#define UCUNIT_WriteInt(n)    System_WriteInt(n)

/**
 * @Macro:       UCUNIT_Safestate()
 *
 * @Description: Encapsulates a function which is called for
 *               putting the hardware to a safe state.
 *
 * @Remarks:     Implement a function to put your hardware into
 *               a safe state.
 *
 *               For example, imagine a motor controller
 *               application:
 *                 1. Stop the motor
 *                 2. Power brake
 *                 3. Hold the brake
 *                 4. Switch warning lamp on
 *                 5. Wait for acknowledge
 *                    ...
 *
 */
#define UCUNIT_Safestate()         System_Safestate()

/**
 * @Macro:       UCUNIT_Recover()
 *
 * @Description: Encapsulates a function which is called for
 *               recovering the hardware from a safe state.
 *
 * @Remarks:     Implement a function to recover your hardware from
 *               a safe state.
 *
 *               For example, imagine our motor controller
 *               application:
 *                 1. Acknowledge the error with a key switch
 *                 2. Switch warning lamp off
 *                 3. Reboot
```

```
 *                    ...
 *
 */
#define UCUNIT_Recover()           System_Reset()

/**
 * @Macro:       UCUNIT_Init()
 *
 * @Description: Encapsulates a function which is called for
 *               initializing the hardware.
 *
 * @Remarks:     Implement a function to initialize your microcontroller
 *               hardware. You need at least to initialize the
 *               communication device for transmitting your results to
 *               a host computer.
 *
 */
#define UCUNIT_Init()              System_Init()

/**
 * @Macro:       UCUNIT_Shutdown()
 *
 * @Description: Encapsulates a function which is called to
 *               stop the tests if a checklist fails.
 *
 * @Remarks:     Implement a function to stop the execution of the
 *               tests.
 *
 */
#define UCUNIT_Shutdown()          System_Shutdown()

/**
 * Verbose Mode.
 * UCUNIT_MODE_SILENT: Checks are performed silently.
 * UCUNIT_MODE_NORMAL: Only checks that fail are displayes
 * UCUNIT_MODE_VERBOSE: Passed and failed checks are displayed
 */
//#define UCUNIT_MODE_NORMAL
#define UCUNIT_MODE_VERBOSE

/**
 * Max. number of checkpoints. This may depend on your application
 * or limited by your RAM.
 */
#define UCUNIT_MAX_TRACEPOINTS 16

/*****************************************************************************/
/* **** End of customizing area *****                                        */
/*****************************************************************************/

/*****************************************************************************/
/* Some useful constants                                                     */
/*****************************************************************************/

#define UCUNIT_VERSION "v1.0" /* Version info */
```

```c
#ifndef NULL
#define NULL (void *)0
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

/* Action to take if check fails */
#define UCUNIT_ACTION_WARNING   0 /* Goes through the checks
                                     with message depending on level */
#define UCUNIT_ACTION_SHUTDOWN  1 /* Stops on the end of the checklist
                                     if any check has failed */
#define UCUNIT_ACTION_SAFESTATE 2 /* Goes in safe state if check fails */

/*****************************************************************************/
/* Variables */
/*****************************************************************************/

/* Variables for simple statistics */
static int ucunit_checks_failed = 0; /* Numer of failed checks */
static int ucunit_checks_passed = 0; /* Number of passed checks */
static int ucunit_testcases_failed = 0; /* Number of failed test cases */
static int ucunit_testcases_passed = 0; /* Number of passed test cases */
static int ucunit_testcases_failed_checks = 0; /* Number of failed checks in a
testcase */
static int ucunit_checklist_failed_checks = 0; /* Number of failed checks in a
checklist */
static int ucunit_action = UCUNIT_ACTION_WARNING; /* Action to take if a check fails
*/
static int ucunit_checkpoints[UCUNIT_MAX_TRACEPOINTS]; /* Max. number of tracepoints
*/
static int ucunit_index = 0; /* Tracepoint index */

/*****************************************************************************/
/* Internal (private) Macros                                                 */
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_DefineToStringHelper(x)
 *
 * @Description: Helper macro for converting a define constant into
 *               a string.
 *
 * @Param x:     Define value to convert.
 *
 * @Remarks:     This macro is used by UCUNIT_DefineToString().
 *
 */
#define UCUNIT_DefineToStringHelper(x)    #x
```

```
/**
 * @Macro:       UCUNIT_DefineToString(x)
 *
 * @Description: Converts a define constant into a string.
 *
 * @Param x:     Define value to convert.
 *
 * @Remarks:     This macro uses UCUNIT_DefineToStringHelper().
 *
 */
#define UCUNIT_DefineToString(x)   UCUNIT_DefineToStringHelper(x)

#ifdef UCUNIT_MODE_VERBOSE
/**
 * @Macro:       UCUNIT_WritePassedMsg(msg, args)
 *
 * @Description: Writes a message that check has passed.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_WritePassedMsg(msg, args)                        \
    do                                                          \
    {                                                           \
        UCUNIT_WriteString( __FILE__);                           \
        UCUNIT_WriteString(":");                                \
        UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
        UCUNIT_WriteString(" : passed:");                        \
        UCUNIT_WriteString(msg);                                \
        UCUNIT_WriteString("(");                                \
        UCUNIT_WriteString(args);                               \
        UCUNIT_WriteString(")\n");                              \
    } while(0)
#else
#define UCUNIT_WritePassedMsg(msg, args)
#endif

#ifdef UCUNIT_MODE_SILENT
#define UCUNIT_WriteFailedMsg(msg, args)
#else
/**
 * @Macro:       UCUNIT_WriteFailedMsg(msg, args)
 *
 * @Description: Writes a message that check has failed.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
```

```
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_WriteFailedMsg(msg, args)                        \
    do                                                          \
    {                                                           \
        UCUNIT_WriteString(__FILE__);                           \
        UCUNIT_WriteString(":");                                \
        UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
        UCUNIT_WriteString(": failed:");                        \
        UCUNIT_WriteString(msg);                                \
        UCUNIT_WriteString("(");                                \
        UCUNIT_WriteString(args);                               \
        UCUNIT_WriteString(")\n");                              \
    } while(0)
#endif

/**
 * @Macro:       UCUNIT_FailCheck(msg, args)
 *
 * @Description: Fails a check.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_FailCheck(msg, args)                 \
    do                                              \
    {                                               \
        if (UCUNIT_ACTION_SAFESTATE==ucunit_action) \
        {                                           \
            UCUNIT_Safestate();                     \
        }                                           \
        UCUNIT_WriteFailedMsg(msg, args);           \
        ucunit_checks_failed++;                     \
        ucunit_checklist_failed_checks++;           \
    } while(0)

/**
 * @Macro:       UCUNIT_PassCheck(msg, args)
 *
 * @Description: Passes a check.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
```

```
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_PassCheck(message, args)             \
    do                                              \
    {                                               \
        UCUNIT_WritePassedMsg(message, args);       \
        ucunit_checks_passed++;                     \
    } while(0)



/*************************************************************************/
/* Check Macros                                                          */
/*************************************************************************/

/**
 * @Macro:       UCUNIT_Check(condition, msg, args)
 *
 * @Description: Checks a condition and prints a message.
 *
 * @Param msg:   Message to write.
 * @Param args:  Argument list as string
 *
 * @Remarks:     Basic check. This macro is used by all higher level checks.
 *
 */
#define UCUNIT_Check(condition, msg, args)            \
    if ( (condition) ) { UCUNIT_PassCheck(msg, args); } else { UCUNIT_FailCheck(msg,
args); }

/**
 * @Macro:       UCUNIT_CheckIsEqual(expected,actual)
 *
 * @Description: Checks that actual value equals the expected value.
 *
 * @Param expected: Expected value.
 * @Param actual: Actual value.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsBitSet(value, bitno) \
    UCUNIT_Check( (1==(((value)>>(bitno)) & 0x01) ), "IsBitSet", #value "," #bitno)

#define UCUNIT_CheckIsBitClear(value, bitno) \
    UCUNIT_Check( (0==(((value)>>(bitno)) & 0x01) ), "IsBitClear", #value "," #bitno)

#define UCUNIT_CheckIsEqual(expected,actual)         \
    UCUNIT_Check( (expected) == (actual), "\n \r IsEqual", #expected "," #actual )

/**
 * @Macro:       UCUNIT_CheckIsNull(pointer)
```

```
 *
 * @Description: Checks that a pointer is NULL.
 *
 * @Param pointer: Pointer to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsNull(pointer)                   \
    UCUNIT_Check( (pointer) == NULL, "\n \r IsNull", #pointer)

/**
 * @Macro:       UCUNIT_CheckIsNotNull(pointer)
 *
 * @Description: Checks that a pointer is not NULL.
 *
 * @Param pointer: Pointer to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsNotNull(pointer)                \
    UCUNIT_Check( (pointer) != NULL, "\n \r IsNotNull", #pointer)

/**
 * @Macro:       UCUNIT_CheckIsInRange(value, lower, upper)
 *
 * @Description: Checks if a value is between lower and upper bounds (inclusive)
 *               Mathematical: lower <= value <= upper
 *
 * @Param value: Value to check.
 * @Param lower: Lower bound.
 * @Param upper: Upper bound.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsInRange(value, lower, upper)    \
    UCUNIT_Check( ( (value>=lower) && (value<=upper) ), "\n \r IsInRange", #value ","
#lower "," #upper)

/**
 * @Macro:       UCUNIT_CheckIs8Bit(value)
 *
 * @Description: Checks if a value fits into 8-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs8Bit(value)                      \
    UCUNIT_Check( value==(value & 0xFF), "\n \r Is8Bit", #value )

/**
```

```
 * @Macro:       UCUNIT_CheckIs16Bit(value)
 *
 * @Description: Checks if a value fits into 16-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs16Bit(value)                      \
    UCUNIT_Check( value==(value & 0xFFFF), "Is16Bit", #value )

/**
 * @Macro:       UCUNIT_CheckIs32Bit(value)
 *
 * @Description: Checks if a value fits into 32-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs32Bit(value)                      \
    UCUNIT_Check( value==(value & 0xFFFFFFFF), "\n \r Is32Bit", #value )

/**
 * Checks if bit is set
 */
/**
 * @Macro:       UCUNIT_CheckIsBitSet(value, bitno)
 *
 * @Description: Checks if a bit is set in value.
 *
 * @Param value: Value to check.
 * @Param bitno: Bit number. The least significant bit is 0.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */

#define UCUNIT_TestcaseBegin(name)                                      \
    do                                                                  \
    {                                                                   \
        UCUNIT_WriteString("\n \r====================================\n"); \
        UCUNIT_WriteString(name);                                       \
        UCUNIT_WriteString("\n \r====================================\n\r"); \
        ucunit_testcases_failed_checks = ucunit_checks_failed;          \
    }                                                                   \
    while(0)

/**
 * @Macro:       UCUNIT_TestcaseEnd()
 *
 * @Description: Marks the end of a test case and calculates
 *               the test case statistics.
```

```
 *
 * @Remarks:     This macro uses UCUNIT_WriteString(msg) to print the result.
 *
 */
#define UCUNIT_TestcaseEnd()                                                \
    do                                                                      \
    {                                                                       \
        UCUNIT_WriteString("\n \r=====================================\n");  \
        if( 0==(ucunit_testcases_failed_checks - ucunit_checks_failed) ) \
        {                                                                   \
            UCUNIT_WriteString("\n \rTestcase passed.\n");                  \
            ucunit_testcases_passed++;                                      \
        }                                                                   \
        else                                                                \
        {                                                                   \
            UCUNIT_WriteFailedMsg("\n \rEndTestcase","");                   \
            ucunit_testcases_failed++;                                      \
        }                                                                   \
        UCUNIT_WriteString("\n \r=====================================\n"); \
    }                                                                       \
    while(0)



#define UCUNIT_WriteSummary()                                               \
{                                                                           \
    UCUNIT_WriteString("\n \r ************************************");        \
    UCUNIT_WriteString("\n \r Testcases: failed: ");                        \
    UCUNIT_WriteInt(ucunit_testcases_failed);                               \
    UCUNIT_WriteString("          passed:");                          \
    UCUNIT_WriteInt(ucunit_testcases_passed);                               \
    UCUNIT_WriteString("\n\r Checks:    failed:");                            \
    UCUNIT_WriteInt(ucunit_checks_failed);                                  \
    UCUNIT_WriteString("           passed: ");                          \
    UCUNIT_WriteInt(ucunit_checks_passed);                                  \
    UCUNIT_WriteString("\n\r ************************************\n"); \
}

/**
 * @Macro:       UCUNIT_Tracepoint(index)
 *
 * @Description: Marks a trace point.
 *               If a trace point is executed, its coverage state switches
 *               from 0 to the line number.
  *              If a trace point was never executed, the state
 *               remains 0.
 *
 * @Param index: Index of the tracepoint.
 *
 * @Remarks:     This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_Tracepoint(index)                                \
    if(index<UCUNIT_MAX_TRACEPOINTS)                            \
    {                                                           \
```

```
        ucunit_checkpoints[index] = __LINE__;            \
    }                                                    \
    else                                                 \
    {                                                    \
        UCUNIT_WriteFailedMsg("Tracepoint index", #index);     \
    }
#endif /*UCUNIT_H_*/

/********************* Reference*********************************************
 https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers
/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                    Unittest.c

https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers/
*****************************************************************************/
#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include <led.h>
#include "UCunit.h"
#include "state_driven.h"
#include "TMP102.h"
#include <string.h>
uint8_t actual;
void unit_test(void){
      UCUNIT_Init();
      UCUNIT_TestcaseBegin("UNIT TESTING BEGINS");
      UCUNIT_CheckIsEqual(0x60,actual);
      UCUNIT_CheckIsInRange(actual,0,80);
      UCUNIT_CheckIs8Bit(actual);
      UCUNIT_CheckIs16Bit(actual);
      UCUNIT_CheckIs32Bit(actual);
      UCUNIT_CheckIsBitSet(actual, 5);
      UCUNIT_CheckIsBitClear(actual,5);
      UCUNIT_WriteSummary();
      UCUNIT_TestcaseEnd();
}
/*********************************Reference*************************************
 https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers/
*****************************************************************************
*******/
```

```
/*************************************************************************
                                PES PROJECT 4
            AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                    Cross Platform IDE: MCUXpresso IDE v11
                     Cross-Compiler: ARM GCC (FB version)
                                 Unittest.h
/********************************** References *******************************
https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers/
*************************************************************************/

#ifndef UNIT_TEST_

#define UNIT_TEST_

#endif
   /*************************************************************************
                                PES PROJECT 4
            AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                    Cross Platform IDE: MCUXpresso IDE v11
                     Cross-Compiler: ARM GCC (FB version)
                                  system.c
*************************************************************************/
/********************************** References *******************************
 https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers/
*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include "system.h"
#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include <led.h>
#include "UCunit.h"
#include <string.h>
#include "logger.h"

/* Stub: Initialize your hardware here */
void System_Init(void)
{

      log_string("\n \r Initialization of system done");
}

/* Stub: Shutdown your hardware here */
void System_Shutdown(void)
{
```

```
	PRINTF("\n \r Shutdowns the system");
	exit(0);
}

/* Stub: Recover your system from a safe state. */
void System_Recover(void)
{
	/* Stub: Recover the hardware */
	/* asm("\tRESET"); */
	PRINTF("\n \r System recovers.\n");
	exit(0);
}

/* Stub: Put system in a safe state */
void System_Safestate(void)
{
	PRINTF("\n \r Safe state of system\n");
	exit(0);
}

/* Stub: Write a string to the host/debugger/simulator */
void System_WriteString(char * string)
{
	PRINTF(string);
}

void System_WriteInt(int d)
{
	PRINTF(" %i", d);
}
/**************************************************************************
                                PES PROJECT 4
            AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                    Cross Platform IDE: MCUXpresso IDE v11
                     Cross-Compiler: ARM GCC (FB version)
                                system.h
**************************************************************************//*
/********************* References ******************************************
 https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers/
**************************************************************************/

#ifndef SYSTEM_H_
#define SYSTEM_H_

/* function prototypes */
void System_Init(void);
void System_Shutdown(void);
void System_Safestate(void);
void System_Recover(void);
void System_WriteString(char * string);
void System_WriteInt(int d);

#endif /* SYSTEM_H_ */
```

```c
/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                        LOGGER.C
*****************************************************************************/
#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include <led.h>
#include "TMP102.h"
#include "fsl_device_registers.h"

extern uint8_t var_flag;
 //char str[40];
uint32_t *data_ptr;
extern char ch_arr[40][40];

//enum functions{
//     temp_reading=0,
//     temp_alert,
//     temp_average,
//     temp_disconnected
//}f;



int log_level(m,f)
{
if(m==0 && f==0)
{
       PRINTF("\n \r ************************************************");
       PRINTF("\n \r TEST MODE:TEMP_READING FUNCTION ,%s",ch_arr[0]);
       PRINTF("\n \r************************************************");
}
else if(m==0 && f==1)
{
       PRINTF("\n \r************************************************");
       PRINTF("\n \r TEST MODE :Temp_Average, %s",ch_arr[1]);
       PRINTF("\n \r************************************************");
}
else if(m==0 && f==2)
{
       PRINTF("\n \r************************************************");
       PRINTF("\n \r TEST MODE :Temp_Alert ,%s",ch_arr[2]);
       PRINTF("\n \r************************************************");
}
```

```
else if(m==0 && f==3)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \rTEST MODE :Temp_Disconnected ,%s",ch_arr[3]);
	PRINTF("\n \r*************************************************");
}
else if(m==0 && f==3)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r TEST MODE:Temp_Disconnected ,%s",ch_arr[4]);
	PRINTF("\n \r*************************************************");
}
else if(m==0 && f==4)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r TEST MODE:LED_ON  %s ",ch_arr[5]);
	PRINTF("\n \r*************************************************");
}
else if(m==0 && f==5)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r TEST MODE: %s",ch_arr[6]);
	PRINTF("\n \r*************************************************");
}
else if(m==1 && f==0)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r DEBUG MODE:Temp_Reading ,%s",ch_arr[0]);
	PRINTF("\n \r*************************************************");
}
else if(m==1 && f==1)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r DEBUG MODE:Temp_Average %s",ch_arr[1]);
	PRINTF("\n \r*************************************************");
}
else if(m==1 && f==2)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r DEBUG MODE :Temp_Alert %s",ch_arr[2]);
	PRINTF("\n \r*************************************************");
}
else if(m==1 && f==3)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r DEBUG MODE:Temp_Disconnected %s",ch_arr[3]);
	PRINTF("\n \r*************************************************");
}
else if(m==1 && f==4)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r DEBUG MODE:LED_ON %s",ch_arr[4]);
	PRINTF("\n \r*************************************************");
}
else if(m==0 && f==5)
```

```
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r DEBUG MODE:SWITCH STATE MACHINES %s",ch_arr[5]);
	PRINTF("\n \r*************************************************");
}
else if(m==2 && f==0)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r STATUS MODE:Temp Reading %s",ch_arr[0]);
	PRINTF("\n \r*************************************************");
}
else if(m==2 && f==1)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r STATUS MODE:Temp_average %s",ch_arr[1]);
	PRINTF("\n \r*************************************************");
}
else if(m==2 && f==2)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r STATUS MODE:Temp_Alert %s",ch_arr[2]);
	PRINTF("\n \r*************************************************");
}
else if(m==2 && f==3)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r STATUS MODE:Temp_Disconnected %s",ch_arr[3]);
	PRINTF("\n \r*************************************************");
}
else if(m==2 && f==4)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r STATUS MODE:LED_ON %s",ch_arr[4]);
	PRINTF("\n \r*************************************************");
}
else if(m==0 && f==5)
{
	PRINTF("\n \r*************************************************");
	PRINTF("\n \r STATUS MODE:SWITCH STATE MACHINES %s",ch_arr[5]);
	PRINTF("\n \r*************************************************");
}
}

int log_integer(size_t x)
{
	PRINTF(" %d ",x);
	return 0;
}


void log_string(char *ptr_st)
{
	PRINTF("\n \r %s",ptr_st);
}
```

```
/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                      LOGGER.H
*****************************************************************************/
#ifndef LOGGER_H_
#define LOGGER_H_

int log_integer();
void log_string();
int log_level();

#endif /* LOGGER_H_ */


/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                       led.h
*****************************************************************************/

#ifndef LED_H_

#define LED_H_

void init_LED(void);
void led_alert(void);
void led_error(void);
void led_wait(void);
#endif
```

```c
/*****************************************************************************
                                    PES PROJECT 4
                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                        Cross Platform IDE: MCUXpresso IDE v11
                         Cross-Compiler: ARM GCC (FB version)
                                        led.c
*****************************************************************************/
#include "led.h"
#include <stdint.h>
#include <stdio.h>
#include "fsl_debug_console.h"
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"

void init_LED(void);
void led_alert(void);
void led_error(void);
void led_wait(void);

void init_LED(void)
{
      BOARD_InitBootPins();
      BOARD_InitBootClocks();
      BOARD_InitBootPeripherals();
      BOARD_InitDebugConsole();
      LED_BLUE_INIT(1);
      LED_RED_INIT(1);
      LED_GREEN_INIT(1);
}

void led_alert(void)            //temp alert state
{
       LED_RED_OFF();
       LED_GREEN_OFF();
       LED_BLUE_ON();
       //delay(10000);
}

void led_error(void)      //error or disconnected state
{
       LED_GREEN_OFF();
       LED_BLUE_OFF();
       LED_RED_ON();
       //delay(10000);
}

void led_wait(void) //temp reading state
{
       LED_RED_OFF();
       LED_BLUE_OFF();
       LED_GREEN_ON();
}
```

I2C READ AND WRITE CAPTURES