

# Principles of Embedded Software Project 4

---

## PDF Documentation

---

### README.md

Principles of Embedded Software Project 4

Readme

Name - Atharva Nandanwar

Folder Structure:

Source - contains all \*.c and \*.h files

main.c/h - main file which contains main subroutine

i2c - has i2c subroutines and tmp102 specific functions/macros

led\_control - has led specific subroutines

logger - has logger and errno files which are used as system wide

POST - has Power On Self Test subroutines

state\_machine - has state machine related implementations

test - contains a test suite to test various test cases

uctest - contains source files for unit testing

Leveraged\_Code\_Documentation - has list of links that I referred

Compilation instructions:

There are three targets

Debug - compiles the code with detailed logging

Run - compiles the code with lesser logging than debug

Test - compiles the test suite

Clean the target

Press on one of the build options

Debug the code

New Observations:

To make system wide logger, I borrowed the idea from one of the

For state machine implementation, I tried to make my code really

I found the idea of making colorful logging really innovative in

Problems Faced:

Major problem I faced during this project was getting my I2C to

Designing a common state machine implementation where I can use

Sharing resources between two functions would have been better.

Feel free to give detailed feedback on the code. ^.^

## Makefile

```
# Makefile for Memory Test Project
```

```
# Author : Atharva Nandanwar
```

```
# Date: 10/31/2019
```

```
#####
```

```

# Build Variables

# Program for removing files
RM := rm -rf

# Program for making directories
MK := mkdir -p

# ARM compiler
ARM_CC := arm-none-eabi-gcc

# ARM linker
ARM_LL := arm-none-eabi-gcc

#####
# ARM Compiler Flags
ARM_FLAGS := -c \
             -std=c99 \
             -O0 \
             -g3 \
             -ffunction-sections \
             -fmessage-length=0 \
             -fno-common \
             -fdata-sections \
             -fno-builtin \
             -mcpu=cortex-m0plus \
             -mthumb

# ARM Linker Flags
ARM_LL_FLAGS := -v \
                -nostdlib \
                -Xlinker -Map="./Debug/pes_project_4.map" \
                -Xlinker --gc-sections \
                -Xlinker -print-memory-usage \
                -Xlinker --sort-section=alignment \
                -Xlinker --cref \
                -mcpu=cortex-m0plus \
                -mthumb \
                -T linkerfile.ld \
                -o $(EXE)

```

```
# ARM Defines
ARM_DEFS := \
    -D__REDLIB__ \
    -DCPU_MKL25Z128VLK4 \
    -DCPU_MKL25Z128VLK4_cm0plus \
    -DSDK_OS_BAREMETAL \
    -DFSL_RTOS_BM \
    -DCR_INTEGER_PRINTF \
    -DPRINTF_FLOAT_ENABLE=0 \
    -DSCANF_FLOAT_ENABLE=0 \
    -DPRINTF_ADVANCED_ENABLE=0 \
    -DSCANF_ADVANCED_ENABLE=0 \
    -D__MCUXPRESSO \
    -D__USE_CMSIS \
    -DDEBUG \
    -DFRDM_KL25Z \
    -DFREEDOM \
    -specs=redlib.specs \
    -DSDK_DEBUGCONSOLE=0 \
    -DSDK_DEBUGCONSOLE_UART
```

```
# Build Folders
SOURCE := ./source
DEBUG := ./Debug
```

```
# ARM Include Files
ARM_INCS := \
    -I"${SOURCE}" \
    -I"${SOURCE}/i2c" \
    -I"${SOURCE}/led_control" \
    -I"${SOURCE}/logger" \
    -I"${SOURCE}/POST" \
    -I"${SOURCE}/state_machine" \
    -I"${SOURCE}/test" \
    -I"${SOURCE}/uctest" \
    -I"board" \
    -I"CMSIS" \
    -I"drivers" \
    -I"startup" \
    -I"utilities" \
```

# ARM Object Files

```
ARM_OBJS := \
    $(DEBUG)/source/i2c/i2c.o \
    $(DEBUG)/source/i2c/tmp102.o \
    $(DEBUG)/source/led_control/led_control.o \
    $(DEBUG)/source/logger/logger.o \
    $(DEBUG)/source/logger/errno.o \
    $(DEBUG)/source/POST/post.o \
    $(DEBUG)/source/state_machine/state_machine.o \
    $(DEBUG)/source/uctest/System.o \
    $(DEBUG)/startup/startup_mkl25z4.o \
    $(DEBUG)/CMSIS/system_MKL25Z4.o \
    $(DEBUG)/board/board.o \
    $(DEBUG)/board/clock_config.o \
    $(DEBUG)/board/peripherals.o \
    $(DEBUG)/board/pin_mux.o \
    $(DEBUG)/drivers/fsl_clock.o \
    $(DEBUG)/drivers/fsl_common.o \
    $(DEBUG)/drivers/fsl_flash.o \
    $(DEBUG)/drivers/fsl_gpio.o \
    $(DEBUG)/drivers/fsl_lpsci.o \
    $(DEBUG)/drivers/fsl_smc.o \
    $(DEBUG)/drivers/fsl_uart.o \
    $(DEBUG)/utilities/fsl_debug_console.o
```

# ARM Dependencies Files

```
ARM_DEPS := \
    $(DEBUG)/source/i2c/i2c.d \
    $(DEBUG)/source/i2c/tmp102.d \
    $(DEBUG)/source/led_control/led_control.d \
    $(DEBUG)/source/logger/logger.d \
    $(DEBUG)/source/logger/errno.d \
    $(DEBUG)/source/POST/post.d \
    $(DEBUG)/source/state_machine/state_machine.d \
    $(DEBUG)/source/uctest/System.d \
    $(DEBUG)/startup/startup_mkl25z4.d \
    $(DEBUG)/CMSIS/system_MKL25Z4.d \
    $(DEBUG)/board/board.d \
    $(DEBUG)/board/clock_config.d \
    $(DEBUG)/board/peripherals.d \
    $(DEBUG)/board/pin_mux.d \
```

```

$(DEBUG)/drivers/fsl_clock.d \
$(DEBUG)/drivers/fsl_common.d \
$(DEBUG)/drivers/fsl_flash.d \
$(DEBUG)/drivers/fsl_gpio.d \
$(DEBUG)/drivers/fsl_lpsci.d \
$(DEBUG)/drivers/fsl_smc.d \
$(DEBUG)/drivers/fsl_uart.d \
$(DEBUG)/utilities/fsl_debug_console.d

# Executable file
EXE := $(DEBUG)/pes_project_4.axf

#####
# Build Rules
# Rules for making all
all : $(EXE)

#####
# Selecting Platform
ifeq ($(BUILD), TEST)
build_option := test
else ifeq ($(BUILD), LOG)
build_option := log
else ifeq ($(BUILD), RUN)
build_option := run
endif

#####

$(EXE) : $(build_option)

#####
# Rule for compiling tests
test : directories $(ARM_OBJS) $(SOURCE)/test/test.c
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) ./source/test/test.c -o $(DEB
    @arm-none-eabi-gcc -nostdlib -Xlinker -Map= "./Debug/pes_project_4.map" -Xlink
    @echo "Testing Code Compiled"

#####
# Rule for compiling detailed debug log
log : directories $(ARM_OBJS) $(SOURCE)/main.c
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -DLOG $(SOURCE)/main.c -o $(D

```

```

@arm-none-eabi-gcc -nostdlib -Xlinker -Map="./Debug/pes_project_4.map" -Xlink
@echo "KL25Z with logging on"

#####
# Rule for compiling program with normal logging
run : directories $(ARM_OBJS) $(SOURCE)/main.c
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -DRUN $(SOURCE)/main.c -o $(D
    @arm-none-eabi-gcc -nostdlib -Xlinker -Map="./Debug/pes_project_4.map" -Xlink
    @echo "KL25Z with logging off"

#####
# Essential Source Files
$(DEBUG)/source/i2c/%.o: ./source/i2c/%.c
    @echo 'Building file: $<'
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/source/led_control/%.o: ./source/led_control/%.c
    @echo 'Building file: $<'
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/source/logger/%.o: ./source/logger/%.c
    @echo 'Building file: $<'
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/source/POST/%.o: ./source/POST/%.c
    @echo 'Building file: $<'
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/source/state_machine/%.o: ./source/state_machine/%.c
    @echo 'Building file: $<'
    @$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

```

```

$(DEBUG)/source/uctest/%.o: ./source/uctest/%.c
    @echo 'Building file: $<'
    @$ (ARM_CC) $ (ARM_FLAGS) $ (ARM_DEFS) $ (ARM_INCS) -MMD -MP -MF"./$ (@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

#####
# Essesntial ARM Object Files
$(DEBUG)/board/%.o: ./board/%.c
    @echo 'Building file: $<'
    @$ (ARM_CC) $ (ARM_FLAGS) $ (ARM_DEFS) $ (ARM_INCS) -MMD -MP -MF"./$ (@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/CMSIS/%.o: ./CMSIS/%.c
    @echo 'Building file: $<'
    @$ (ARM_CC) $ (ARM_FLAGS) $ (ARM_DEFS) $ (ARM_INCS) -MMD -MP -MF"./$ (@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/drivers/%.o: ./drivers/%.c
    @echo 'Building file: $<'
    @$ (ARM_CC) $ (ARM_FLAGS) $ (ARM_DEFS) $ (ARM_INCS) -MMD -MP -MF"./$ (@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/startup/%.o: ./startup/%.c
    @echo 'Building file: $<'
    @$ (ARM_CC) $ (ARM_FLAGS) $ (ARM_DEFS) $ (ARM_INCS) -MMD -MP -MF"./$ (@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

$(DEBUG)/utilities/%.o: ./utilities/%.c
    @echo 'Building file: $<'
    @$ (ARM_CC) $ (ARM_FLAGS) $ (ARM_DEFS) $ (ARM_INCS) -MMD -MP -MF"./$ (@:%.o=%.d)"
    @echo 'Finished building: $<'
    @echo ' '

#####
# Making directories

```



```

.PHONY : directories
directories :
    $(MK) \
    $(DEBUG) \
    $(DEBUG)/board \
    $(DEBUG)/CMSIS \
    $(DEBUG)/drivers \
    $(DEBUG)/startup \
    $(DEBUG)/utilities \
    $(DEBUG)/source/i2c \
    $(DEBUG)/source/led_control \
    $(DEBUG)/source/logger \
    $(DEBUG)/source/POST \
    $(DEBUG)/source/state_machine \
    $(DEBUG)/source/test \
    $(DEBUG)/source/uctest \

# Clean target
clean:
    @$(RM) \
    $(DEBUG)/board \
    $(DEBUG)/CMSIS \
    $(DEBUG)/drivers \
    $(DEBUG)/startup \
    $(DEBUG)/utilities \
    $(DEBUG)/source \
    $(DEBUG)/pes_project_4.axf \
    $(DEBUG)/pes_project_4.map
    @echo "Build cleaned"

```

## Source Code

main.c

```

/**
 * File Name      - main.c
 * Description    - main routine

```

```

* Author          - Atharva Nandanwar
* Tools           - GNU C Compiler / ARM Compiler Toolchain
* Leveraged Code  -
* URL             -
*/

#include "main.h"

error_t errno;
system_state_t __system = {0, 0, 0, 0, 0, 0, 0, 0, 1};
system_state_t* system_state = &__system;

int main(void)
{
    //Initializing board pins
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
    BOARD_InitDebugConsole();

    printf("\x1B[1;33m\n\r\n\r\n\r-----Starting Program-----\x1B[0m\n\r
    // Logger Setup
#ifdef RUN
    logger.Set_Log_Level(lNormal);
#elif LOG
    logger.Set_Log_Level(lDebug);
#endif

    // POST
    I2C_Init();
    errno = post();
    if(errno == POST_Successful)
    {
        logger.Log_Write(__func__, mStatus, Get_Error_Message(errno));
    }
    else if (errno == POST_Failed)
    {
        logger.Log_Write(__func__, mError, Get_Error_Message(errno));
        Turn_On_Only_LED(Red);
        End_Program();
    }
}

```

```

// I2C Initialization Routine
I2C_Write(TMP102.config_reg_address, 0x78, 0x80);
for(volatile int i = 10000; i > 0; i--);
I2C_Write(TMP102.tmp_HI_reg_address, 0x05, 0x00);
for(volatile int i = 10000; i > 0; i--);
I2C_Write(TMP102.tmp_LOW_reg_address, 0x00, 0x00);
I2C_Alert_Init();

// State Machine Setup
state_machine_t* sm1 = NULL;
state_machine_t* sm2 = NULL;
sm1 = State_Machine_Init(State_Driven);
sm2 = State_Machine_Init(Table_Driven);
system_state->state_machine_id = 1;
system_state->alert = 0;

while(1)
{
    // Check if device is disconnected
    if(I2C_Check() == DISCONNECTED)
    {
        system_state->disconnect = 1;
    }

    // Toggling between state machines
    if(system_state->state_machine_id == 1)
    {
        if(system_state->print_flag)
        {
            logger.Log_Write(__func__, mDebug, "State Machine 1");
            system_state->print_flag = 0;
        }
        Event_Handler(sm1, system_state);
    }
    else if (system_state->state_machine_id == 2)
    {
        if(system_state->print_flag)
        {
            logger.Log_Write(__func__, mDebug, "State Machine 2");
            system_state->print_flag = 0;
        }
    }
}

```

```

    }
    Event_Handler(sm2, system_state);
}

// Delay for disconnect check
for(volatile int i = 1000; i > 0; i--);
}
return 0;
}

```

```

void SysTick_Handler(void)
{
    if(system_state->timeout_started)
        system_state->counter++;
}

```

```

void PORTA_IRQHandler(void)
{
    if(PORTA->ISFR & ALERT_PIN)
    {
        system_state->alert = 1;
    }
    PORTA->PCR[5] |= PORT_PCR_ISF_MASK;
}

```

## main.h

```

/**
 * File Name      - main.h
 * Description    - main routine header
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */
#ifndef MAIN_H_
#define MAIN_H_

```

```

#include <stdint.h>
#include <stdio.h>

#include "pin_mux.h"
#include "peripherals.h"
#include "clock_config.h"
#include "board.h"

#include "state_machine.h"
#include "i2c.h"
#include "logger.h"
#include "post.h"

#endif /* MAIN_H_ */

```

## POST/post.c

```

/**
 * File Name      - post.c
 * Description    - contains power on self test
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#include "post.h"

/**
 * Function - post
 * Brief - executes Power On Self Test
 * Returns -
 * error enum value indicating POST success or fail
 */
uint16_t post(void)
{
    if(logger.Get_Log_Level() == lDebug)
    {

```

```

        logger.Log_Write(__func__, mDebug, "Power On Self Test Started");
    }

    // Do a write operation, and read if it's written
    I2C_Write(TMP102.config_reg_address, 0x78, 0x80);
    for(volatile int i = 10000; i > 0; i--);
    volatile uint16_t temp = I2C_Read(TMP102.config_reg_address);

    // If write value == read value, then SUCCESS!!
    if(temp == 0x7880 || temp == 0x78A0)
    {
        return POST_Successful;
    }
    else
    {
        return POST_Failed;
    }
}

```

## POST/post.h

```

/**
 * File Name      - post.c
 * Description    - header file for post.c
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#ifndef POST_POST_H_
#define POST_POST_H_
#include "i2c.h"
#include "logger.h"

// Prototype Function
uint16_t post(void);
#endif /* POST_POST_H_ */

```

## i2c/i2c.c

```

/**
 * File Name      - i2c.c
 * Description    - contains functions related to I2C routines
 * Author         - Atharva Nandanwar
 * Tools          - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code - KL25Z I2C Example by Dean
 * URL           -
 */

#include "i2c.h"

/*
 * Function - I2C_Init
 * Brief - Initializes I2C peripheral
 */
void I2C_Init(void)
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Starting I2C peripheral initializatio
    }
    // Enabling clock on Port C and I2C0 peripheral
    SIM->SCGC4 |= SIM_SCGC4_I2C0_MASK;
    SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTC_MASK;

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

    // Configuring I2C Peripheral
    I2C0->F |= I2C_F_MULT(0x00);
    I2C0->F |= I2C_F_ICR(0x3D);
    I2C0->C1 |= I2C_C1_IICEN_MASK;
    I2C0->C2 |= I2C_C2_HDRS_MASK;
    I2C0->SLTH |= I2C_SLTL_SSLT(0x01);

    // Log Message

```

```

        logger.Log_Write(__func__, mStatus, "I2C peripheral initialized");
    }

    /*
    * Function - I2C_Alert_Init
    * Brief - Initializes Alert Pin for TMP102
    */
    void I2C_Alert_Init(void)
    {
        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Starting TMP102 Alert Pin Initializat
        }
        // Port A Pin 5 setup - GPIO, Rising Edge Interrupt, Pull Down
        PORTA->PCR[5] |= PORT_PCR_MUX(1) | PORT_PCR_IRQC(0x09) | PORT_PCR_PE_MASK;
        PORTA->PCR[5] &= ~PORT_PCR_PS_MASK;
        GPIOA->PDDR &= ~ALERT_PIN;
        NVIC_EnableIRQ(PORTA_IRQn);

        // Log Message
        logger.Log_Write(__func__, mStatus, "TMP102 Alert Pin Initialized");
    }

    /*
    * Function - I2C_Check
    * Brief - function to check for disconnect event
    * Return -
    * returns connection status
    */
    uint8_t I2C_Check(void)
    {
        uint16_t data = 0;
        volatile uint8_t read;

        // Set I2C as Transmitter mode
        I2C0->C1 |= I2C_C1_TX_MASK;

        // Send start bit
        I2C0->C1 |= I2C_C1_MST_MASK;

        // Send slave address with write

```



```

I2C0->D = (TMP102.address << 1) | WRITE;

volatile int i = 0;
// Wait for ACK
while((I2C0->S & I2C_S_IICIF_MASK) == 0)
{
    i++;
    if(i >= 40000)
    {
        return DISCONNECTED;
        break;
    }
}
if((I2C0->S & I2C_S_IICIF_MASK) == 1)
    return DISCONNECTED;
I2C0->S |= I2C_S_IICIF_MASK;

// Send register address
I2C0->D = TMP102.config_reg_address;

// Wait for ACK
while((I2C0->S & I2C_S_IICIF_MASK) == 0)
{
    i++;
    if(i >= 40000)
    {
        return DISCONNECTED;
        break;
    }
}
if((I2C0->S & I2C_S_IICIF_MASK) == 1)
    return DISCONNECTED;

// Send stop signal
I2C0->C1 &= ~I2C_C1_MST_MASK;
I2C0->S |= I2C_S_IICIF_MASK;

return CONNECTED;
}

```

```

/*
 * Function - I2C_Read
 * Brief - Reads data from a given register address
 * Argument -
 * register_address -> input appropriate register address
 * Return -
 * returns 16-bit integer which has data from I2C device
 */
uint16_t I2C_Read(uint8_t register_address)
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Starting I2C Read Operation");
    }
    uint16_t data = 0;
    volatile uint8_t read;

    // Set I2C as Transmitter mode
    I2C0->C1 |= I2C_C1_TX_MASK;

    // Send start bit
    I2C0->C1 |= I2C_C1_MST_MASK;

    // Send slave address with write
    I2C0->D = (TMP102.address << 1) | WRITE;

    // Wait for ACK
    while((I2C0->S & I2C_S_IICIF_MASK) == 0);
    I2C0->S |= I2C_S_IICIF_MASK;

    // Send register address
    I2C0->D = register_address;

    // Wait for ACK
    while((I2C0->S & I2C_S_IICIF_MASK) == 0);
    I2C0->S |= I2C_S_IICIF_MASK;

    // Send repeated start
    I2C0->C1 |= I2C_C1_RSTA_MASK;

    // Send slave address with read

```

```

I2C0->D = (TMP102.address << 1) | READ;

// Wait for ACK
while((I2C0->S & I2C_S_IICIF_MASK) == 0);
I2C0->S |= I2C_S_IICIF_MASK;

// Receiver Mode
I2C0->C1 &= ~I2C_C1_TX_MASK;

// Send ACK
I2C0->C1 &= ~I2C_C1_TXAK_MASK;

// Dummy read
read = I2C0->D;

// Wait for data
while((I2C0->S & I2C_S_IICIF_MASK) == 0);
I2C0->S |= I2C_S_IICIF_MASK;

// Send ACK
I2C0->C1 &= ~I2C_C1_TXAK_MASK;

// Proper Read
read = I2C0->D;
data = read << 8;

// Wait for data
while((I2C0->S & I2C_S_IICIF_MASK) == 0);
I2C0->S |= I2C_S_IICIF_MASK;

// Send NACK
I2C0->C1 |= I2C_C1_TXAK_MASK;

// Proper Read
read = I2C0->D;
data |= read;

// Wait for data
while((I2C0->S & I2C_S_IICIF_MASK) == 0);
I2C0->S |= I2C_S_IICIF_MASK;

```

```

    // Send stop signal
    I2C0->C1 &= ~I2C_C1_MST_MASK;
    I2C0->S |= I2C_S_IICIF_MASK;

    logger.Log_Write(__func__, mStatus, "I2C Read Operation Finished");

    return data;
}

/*
 * Function - I2C_Write
 * Brief - Writes data to a given register
 * Argument -
 * register_address -> appropriate register address
 * byte1 & byte2 -> data to write
 */
void I2C_Write(uint8_t register_address, uint8_t byte1, uint8_t byte2)
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Starting I2C Write Operation");
    }
    // Send start bit
    I2C0->C1 |= I2C_C1_TX_MASK;
    I2C0->C1 |= I2C_C1_MST_MASK;

    // Send slave address
    I2C0->D = (TMP102.address << 1) | WRITE;

    // Wait for ACK
    while((I2C0->S & I2C_S_IICIF_MASK) == 0){ }
    I2C0->S |= I2C_S_IICIF_MASK;

    // Send register address
    I2C0->D = register_address;

    // Wait for ACK
    while((I2C0->S & I2C_S_IICIF_MASK) == 0){ }
    I2C0->S |= I2C_S_IICIF_MASK;

    // Send first data byte

```

```

I2C0->D = byte1;

// Wait for ACK
while((I2C0->S & I2C_S_IICIF_MASK) == 0){ }
I2C0->S |= I2C_S_IICIF_MASK;

// Send second data byte
I2C0->D = byte2;

// Wait for ACK
while((I2C0->S & I2C_S_IICIF_MASK) == 0){ }
I2C0->S |= I2C_S_IICIF_MASK;

// Send stop signal
I2C0->C1 &= ~I2C_C1_MST_MASK;
logger.Log_Write(__func__, mStatus, "I2C Write Operation Finished");
}

```

## i2c/i2c.h

```

/**
 * File Name      - i2c.h
 * Description    - header file for i2c.c
 * Author         - Atharva Nandanwar
 * Tools          - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL            -
 */

#ifndef I2C_I2C_H_
#define I2C_I2C_H_
#include <stdint.h>
#include "MKL25Z4.h"
#include "tmp102.h"
#include "logger.h"

// I2C Macros

```

```

#define READ    1
#define WRITE   0

#define DISCONNECTED 1
#define CONNECTED    0

// Prototype Functions
void I2C_Init(void);
void I2C_Alert_Init(void);
uint8_t I2C_Check(void);
uint16_t I2C_Read(uint8_t register_address);
void I2C_Write(uint8_t register_address, uint8_t byte1, uint8_t byte2);
#endif /* I2C_I2C_H */

```

## i2c/tmp102.c

```

/**
 * File Name      - tmp102.c
 * Description    - contains function for TMP102 device temperature
 *                  and device structure that has register values
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#include "tmp102.h"

/*
 * Function - Get_Temperature
 * Brief - Converts raw data into temperature value
 * Arguments -
 * data -> 16-bit integer value
 * Return -
 * 16-bit integer, which is converted temperature value of data argument
 */
int16_t Get_Temperature(uint16_t data)
{

```

```

if(logger.Get_Log_Level() == lDebug)
{
    logger.Log_Write(__func__, mDebug, "Starting Temperature Conversion Opera
}
// Manipulation done after referring to TMP102
// datasheet
data = data >> 4;
if ((data & MSB) == 1)
{
    // Negative temperatures
    data = ((~data & 0xFFFF) + 1) * -1;
    return data * RESOLUTION;
}
else if ((data & MSB) == 0)
{
    return data * RESOLUTION;
}
}

// TMP102 register addresses
device TMP102 = {0x48, 0x00, 0x01, 0x03, 0x02};

```

## i2c/tmp102.h

```

/**
 * File Name      - tmp102.h
 * Description    - header file for tmp102.c
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#ifndef I2C_TMP102_H_
#define I2C_TMP102_H_
#include <stdint.h>
#include "MKL25Z4.h"
#include "logger.h"

```

```

// Macros
#define MSB          (0x01 << 11)
#define RESOLUTION    0.0625
#define ALERT_PIN     (0x0001 << 5)

// Struct for device addresses and registers
typedef struct {
    uint8_t address;
    uint8_t tmp_reg_address;
    uint8_t config_reg_address;
    uint8_t tmp_HI_reg_address;
    uint8_t tmp_LOW_reg_address;
}device;

extern device TMP102;

// Prototype Functions
int16_t Get_Temperature(uint16_t data);
#endif /* I2C_TMP102_H_ */

```

## led\_control/led\_control.c

```

/**
 * File Name      - led_control.c
 * Description     - contains function for turning on LEDs
 * Author         - Atharva Nandanwar
 * Tools          - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL            -
 */

#include "led_control.h"

/*
 * Function - Turn_On_Only_LED
 * Brief - Turns on specified LED
 * Argument -

```



```

* led color enum
*/
void Turn_On_Only_LED(led_color_t LED)
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Turn ON LED Operation Started");
    }
    // LED_string is used to print messages
    LED_RED_INIT(LOGIC_LED_OFF);
    LED_BLUE_INIT(LOGIC_LED_OFF);
    LED_GREEN_INIT(LOGIC_LED_OFF);

    if(LED == Red)
    {
        errno = LED_Red_ON;
        LED_RED_ON();
        LED_GREEN_OFF();
        LED_BLUE_OFF();
    }

    else if (LED == Blue)
    {
        errno = LED_Blue_ON;
        LED_RED_OFF();
        LED_GREEN_OFF();
        LED_BLUE_ON();
    }

    else if (LED == Green)
    {
        errno = LED_Green_ON;
        LED_RED_OFF();
        LED_GREEN_ON();
        LED_BLUE_OFF();
    }

    logger.Log_Write(__func__, mStatus, Get_Error_Message(errno));
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Turn ON LED Operation Finshed");
    }
}

```

```

    }
}

```

## led\_control/led\_control.h

```

/**
 * File Name      - led_control.h
 * Description    - header file for led_control.c
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#ifndef LED_CONTROL_H_
#define LED_CONTROL_H_
#include <stdio.h>
#include <stdint.h>
#include "board.h"
#include "logger.h"

// Macros
typedef enum {
    Red,
    Green,
    Blue,
} led_color_t;

// Prototype Functions
void Turn_On_Only_LED(led_color_t LED);
#endif /* LED_CONTROL_H_ */

```

## logger/errno.c

```

/**

```

```

* File Name      - errno.c
* Description    - contains error enums, and related functions
* Author        - Atharva Nandanwar
* Tools         - GNU C Compiler / ARM Compiler Toolchain
* Leveraged Code - https://android.googlesource.com/kernel/lk/+upstream-ma
* URL           -
*/

#include "errno.h"

/*
* Function - Get_Error_Message
* Brief - Returns pre-defined error messages
* Argument -
* error_t -> error code for pre-defined errors, or events
* Return -
* returns a string with error message
*/
const char* Get_Error_Message(error_t error)
{
    switch(error)
    {
        case Starting_Program:
            return "Starting Program";
            break;
        case Initiating_POST:
            return "Initiating POST";
            break;
        case POST_Successful:
            return "POST Successful";
            break;
        case POST_Failed:
            return "POST Failed";
            break;
        case Entering_SM1:
            return "Entering State Machine 1";
            break;
        case Exiting_SM1:
            return "Exiting State Machine 1";
            break;
        case Entering_SM2:

```

```
        return "Entering State Machine 2";
        break;
case Exiting_SM2:
    return "Exiting State Machine 2";
    break;
case Reading_Temperature:
    return "Reading Temperature";
    break;
case Reading_Temperature_Complete:
    return "Reading Temperature Complete";
    break;
case Waiting:
    return "In Wait State";
    break;
case Timeout:
    return "Timeout!";
    break;
case Alert_LOW_Temperature:
    return "Alert - Low Temperature";
    break;
case LED_Red_ON:
    return "LED Red is ON";
    break;
case LED_Red_OFF:
    return "LED Red is OFF";
    break;
case LED_Green_ON:
    return "LED Green is ON";
    break;
case LED_Green_OFF:
    return "LED Green is OFF";
    break;
case LED_Blue_ON:
    return "LED Blue is ON";
    break;
case LED_Blue_OFF:
    return "LED Blue is OFF";
    break;
case Device_Disconnected:
    return "Device Disconnected";
    break;
```

```

    case Unhandled_Exception:
        return "Unhandled Exception";
        break;
    default:
        return "Incorrect Status Code";
        break;
    }
}

```

## logger/errno.h

```

/**
 * File Name      - errno.h
 * Description    - header file for errno.c
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#ifndef LOGGER_ERRNO_H_
#define LOGGER_ERRNO_H_
#include <stdint.h>

// Error/Event Enum
typedef enum {
    Starting_Program           = 0x0000,
    Initiating_POST            = 0x1000,
    POST_Successful             = 0x1001,
    POST_Failed                 = 0x1002,
    Entering_SM1                = 0x2000,
    Exiting_SM1                 = 0x200F,
    Entering_SM2                = 0x4000,
    Exiting_SM2                 = 0x400F,
    Reading_Temperature         = 0x8000,
    Reading_Temperature_Complete = 0x8001,
    Waiting                     = 0x8002,
    Timeout                     = 0x8004,
}

```

```

    Alert_LOW_Temperature      = 0x8008,
    LED_Red_ON                 = 0x8101,
    LED_Red_OFF                = 0x8102,
    LED_Green_ON               = 0x8201,
    LED_Green_OFF              = 0x8202,
    LED_Blue_ON                = 0x8401,
    LED_Blue_OFF               = 0x8402,
    Unhandled_Exception         = 0xEEEE,
    Device_Disconnected         = 0xFFFF,
}error_t;

```

```
extern error_t errno;
```

```
// Prototype function
```

```
const char* Get_Error_Message(error_t error);
```

```
#endif /* LOGGER_ERRNO_H_ */
```

## logger/logger.c

```

/**
 * File Name      - logger.c
 * Description    - contains logger subroutines
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code - https://github.com/ntpeters/SimpleLogger/
 * URL           -
 */
#include "logger.h"

// Struct for storing logger data
typedef struct {
    log_level_t Logger_Log_Level;
}logger_data;

logger_data thisLogger;

// Character codes for colors

```

```
// Leveraged Code - https://stackoverflow.com/questions/3585846/color-text-in-ter
const char* red = "\x1B[31m";
const char* green = "\x1B[32m";
const char* blue = "\x1B[34m";
const char* end = "\x1B[0m";

/*
 * Function - Log_Write
 * Brief - Prints a log message
 * Arguments -
 * function_name -> name of the calling function
 * message_type -> Error, Debug or Status message
 * msg, ... -> printf style argument to hold a string and format specifiers
 * Leveraged Code - https://www.ozzu.com/cpp-tutorials/tutorial-writing-custom-pr
 */
void Log_Write(const char* function_name, message_type_t message_type, const char
{
    // To process variable argument list
    va_list args;
    va_start(args, msg);

    // Activate color based on message type
    switch(message_type)
    {
    case mError:
        printf("%s", red);
        break;
    case mDebug:
        printf("%s", blue);
        break;
    case mStatus:
        printf("%s", green);
        break;
    }

    // Log Level Logic
    switch(thisLogger.Logger_Log_Level)
    {
    case lTest:
        printf("Test:");
        break;
    }
}
```

```

    case lDebug:
        printf("Debug:\t");
        break;
    case lNormal:
        printf("Run:\t");
        break;
}

printf("%-27s:\t\t", function_name);

// Message print with color termination code
vprintf(msg, args);
printf("%s\n\r", end);
}

/*
 * Function - Get_Log_Level
 * Brief - returns the current log level
 * Return -
 * returns log_level_t enum value
 */
log_level_t Get_Log_Level (void)
{
    return thisLogger.Logger_Log_Level;
}

/*
 * Function - Set_Log_Level
 * Brief - sets the current log level
 * Arguments -
 * log_level_t enum value
 */
void Set_Log_Level (log_level_t level)
{
    thisLogger.Logger_Log_Level = level;
}

// Declaration for logger struct
logger_instance const logger = {Log_Write, Set_Log_Level, Get_Log_Level};

```



## logger/logger.h

```

/**
 * File Name      - logger.h
 * Description    - header file for logger.c
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#ifndef LOGGER_LOGGER_H_
#define LOGGER_LOGGER_H_
#include <stdint.h>
#include <stdio.h>
#include <errno.h>
#include <stdarg.h>

// Log Level and Message Type enums
typedef enum {lTest, lDebug, lNormal} log_level_t;
typedef enum {mError, mDebug, mStatus} message_type_t;

// Logger Instance struct
typedef struct {
    void ( * const Log_Write )( const char* function_name, \
                                message_type_t message_type, const char *msg, ... );
    void ( * const Set_Log_Level )( log_level_t level );
    log_level_t ( * const Get_Log_Level )( void );
}logger_instance;

extern logger_instance const logger;
#endif /* LOGGER_LOGGER_H_ */

```

## state\_machine/state\_machine.c

```

/**

```

```

* File Name          - state_machine.c
* Description        - contains state machine subroutines
* Author             - Atharva Nandanwar
* Tools              - GNU C Compiler / ARM Compiler Toolchain
* Leveraged Code     - https://github.com/EduMacedo99/FEUP-LCOM
* URL                -
*/

#include "state_machine.h"

// Prototype to functions for Table-Driven State Machine
void fStart(state_machine_t* sm);
void fRead_Complete(state_machine_t* sm);
void fTimeoutComplete(state_machine_t* sm);
void fAlert(state_machine_t* sm);
void fDisconnect(state_machine_t* sm);

// Look Up Table for Table Driven State Machine
state_struct LookUpTable[] = {{sTemperature_Reading, {fStart, fRead_Complete, fTi
                                {sAverage_Wait, {fStart, fRead_Complete, fTimeoutCo
                                {sTemperature_Alert, {NULL, fRead_Complete, NULL, f
                                {sDisconnected, {NULL, NULL, NULL, NULL, fDisconnec
                                }};

// Macros for Table Driven State Machine Execution
#define S_TEMPERATURE_READING    LookUpTable[0]
#define S_AVERAGE_WAIT          LookUpTable[1]
#define S_TEMPERATURE_ALERT      LookUpTable[2]
#define S_DISCONNECTED           LookUpTable[3]

#define F_START                  event_action[0]
#define F_READ_COMPLETE          event_action[1]
#define F_TIMEOUT_COMPLETE       event_action[2]
#define F_ALERT                  event_action[3]
#define F_DISCONNECT             event_action[4]

// Abstracting some magic numbers
#define SECONDS_15    150    //Counter value when 15 seconds have passed

/*
* Function - average_temperature

```

```

* Brief - averages temperature value
* Arguments -
* system_state global struct -> to update global value of system
*/
void average_temperature(system_state_t* system)
{
    // If average == zero - that is acquisition just started
    if (system->average_temperature == 0)
        system->average_temperature = system->temperature;
    else
        system->average_temperature = (system->average_temperature + system->temp
}

/*
* Function - Print_Message
* Brief - prints status messages based on error enums
* Arguments -
* function_name -> to print function name
* error_t -> error enum value
* NOTE - this function only prints status messages
*/
void Print_Message(const char* function_name, error_t error)
{
    errno = error;
    logger.Log_Write(function_name, mStatus, "%s", Get_Error_Message(errno));
}

/*
* Function - End_Program
* Brief - ends the program by going into infinite loop
*/
void End_Program(void)
{
    while(1);
}

/*
* Function - State_Machine_Init
* Brief - Initializes state machine
* Arguments -
* state_machine_type -> State-driven or Table-driven state machine

```

```

* Return -
* returns a pointer to state machine struct
*/
state_machine_t* State_Machine_Init(state_machine_type_t type)
{
    if(logger.Get_Log_Level() == lDebug)
        logger.Log_Write(__func__, mDebug, "Initializing State Machine");
    state_machine_t* sm = (state_machine_t *) malloc(sizeof(state_machine_t));

    if(sm == NULL)
    {
        logger.Log_Write(__func__, mStatus, "Failed to initialize state machine")
        return NULL;
    }

    sm->state = sTemperature_Reading;
    sm->event = eStart;
    sm->type = type;

    logger.Log_Write(__func__, mStatus, "State Machine Initialized");

    return sm;
}

/*
* Function - State_Machine_End
* Brief - Ends state machine
* Arguments -
* pointer to state_machine struct
*/
void State_Machine_End(state_machine_t* sm)
{
    if(sm == NULL)
        return;

    free(sm);
    sm = NULL;
}

/*
* Function - Set_Event

```

```

* Brief - Sets events
* Arguments -
* state_machine struct -> which state_machine
* event -> which event to set the state_machine to
*/
void Set_Event(state_machine_t* sm, event_t event)
{
    sm->event = event;
}

/*
* Function - Set_State
* Brief - Sets state
* Arguments -
* state_machine struct -> which state_machine
* state -> which state to set the state_machine to
*/
void Set_State(state_machine_t* sm, state_t state)
{
    sm->state = state;
}

/*
* Function - Temperature_Reading_State
* Brief - Addresses Temperature Reading State
* Arguments -
* state_machine struct -> which state_machine
* system_state struct -> a pointer to system state global struct
*/
static inline void Temperature_Reading_State(state_machine_t* sm, system_state_t*
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "State - Temperature Reading");
    }

    // Check for Alert
    if(system->alert == 1)
    {
        // Transitions
        if(sm->type == State_Driven)

```

```

        Set_Event(sm, eAlert);
    else if (sm->type == Table_Driven)
        S_TEMPERATURE_READING.F_ALERT(sm);
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Alert Addressed");
    }
    system->alert = 0;
}
// Check for Disconnect
else if(system->disconnect == 1)
{
    if(sm->type == State_Driven)
        Set_Event(sm, eDisconnect);
    else if (sm->type == Table_Driven)
        S_TEMPERATURE_READING.F_DISCONNECT(sm);
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Disconnect Addressed");
    }
}
// Start Event
if(sm->event == eStart)
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Event - Start");
    }

    // Start Event actions
    Turn_On_Only_LED(Green);
    Print_Message(__func__, Reading_Temperature);
    int16_t temp = I2C_Read(TMP102.tmp_reg_address);
    system->temperature = Get_Temperature(temp);
    logger.Log_Write(__func__, mStatus, "Temperature = %d", system->temperatu
    Print_Message(__func__, Reading_Temperature_Complete);

    // Transitions
    Set_State(sm, sAverage_Wait);
    if(sm->type == State_Driven)
        Set_Event(sm, eRead_Complete);

```

```

        else if (sm->type == Table_Driven)
            S_TEMPERATURE_READING.F_READ_COMPLETE(sm);
    }

    // Alert Event
    else if(sm->event == eAlert)
    {
        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Event - Alert");
        }
        // Transitions
        Set_State(sm, sTemperature_Alert);
    }

    // Disconnect Event
    else if(sm->event == eDisconnect)
    {
        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Event - Disconnect");
        }
        // Transitions
        Set_State(sm, sDisconnected);
    }
}

/*
 * Function - Average_Wait_State
 * Brief - Addresses Average Wait State
 * Arguments -
 * state_machine struct -> which state_machine
 * system_state struct -> a pointer to system state global struct
 */
static inline void Average_Wait_State(state_machine_t* sm, system_state_t* system
{
    // This is in wait, and I just want it to be printed once
    if(logger.Get_Log_Level() == lDebug && system->timeout_started == 0)
    {
        logger.Log_Write(__func__, mDebug, "State - Average Wait");
    }
}

```

```

//Check for Disconnect
if(system->disconnect == 1)
{
    // Transitions
    if(sm->type == State_Driven)
        Set_Event(sm, eDisconnect);
    else if (sm->type == Table_Driven)
        S_AVERAGE_WAIT.F_DISCONNECT(sm);

    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Disconnect Addressed");
    }
}

// Event - Read Complete
if(sm->event == eRead_Complete)
{

    // Timeout
    if(system->timeout_started != 1)
    {
        // This is in wait, and I just want it to be printed once
        if(logger.Get_Log_Level() == lDebug && system->timeout_started == 0)
        {
            logger.Log_Write(__func__, mDebug, "Event - Read Complete");
        }
        // Do an average
        average_temperature(system);
        logger.Log_Write(__func__, mStatus, "Average Temperature = %d",\
            system->average_temperature);
        Print_Message(__func__, Waiting);

        // Start Timeout Counting
        system->counter = 0;
        system->timeout_started = 1;
        SysTick_Config(48000000L/10L);
    }
    else
    {

```



```

    if(system->counter >= SECONDS_15)
    {
        // Transitions
        if(sm->type == State_Driven)
            Set_Event(sm, eTimeout_Complete);
        else if (sm->type == Table_Driven)
            S_AVERAGE_WAIT.F_TIMEOUT_COMPLETE(sm);

        Print_Message(__func__, Timeout);

        // Reset System State related to Timeout
        system->timeout_started = 0;
        system->counter = 0;
        SysTick->CTRL = 0;
    }
}

// Event - Timeout Complete
else if(sm->event == eTimeout_Complete)
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "Event - Timeout Complete");
    }

    // Increment Timeout Count
    system->timeout_count++;
    // Transition to other state machine once Timeout(4) occurs
    if(system->timeout_count == 4)
    {
        if(system->state_machine_id == 1)
            system->state_machine_id = 2;
        else if (system->state_machine_id == 2)
            system->state_machine_id = 1;
        system->timeout_count = 0;
        system->print_flag = 1;
    }

    // Transitions

```

```

        Set_State(sm, sTemperature_Reading);
        if(sm->type == State_Driven)
            Set_Event(sm, eStart);
        else if (sm->type == Table_Driven)
            S_AVERAGE_WAIT.F_START(sm);
    }

    // Event - Disconnect
    else if(sm->event == eDisconnect)
    {
        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Event - Timeout Complete");
        }
        Set_State(sm, sDisconnected);
    }
}

/*
 * Function - Temperature_Alert_State
 * Brief - Addresses Temperature Alert State
 * Arguments -
 * state_machine struct -> which state_machine
 * system_state struct -> a pointer to system state global struct
 */
static inline void Temperature_Alert_State(state_machine_t* sm, system_state_t* s
{
    if(logger.Get_Log_Level() == lDebug)
    {
        logger.Log_Write(__func__, mDebug, "State - Temperature Alert");
    }

    //Check for Disconnect
    if(system->disconnect == 1)
    {
        // Transitions
        if(sm->type == State_Driven)
            Set_Event(sm, eDisconnect);
        else if (sm->type == Table_Driven)
            S_AVERAGE_WAIT.F_DISCONNECT(sm);
    }
}

```

```

        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Disconnect Addressed");
        }
    }

    // Event - Alert
    if(sm->event == eAlert)
    {
        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Event - Alert");
        }

        Turn_On_Only_LED(Blue);
        Print_Message(__func__, Alert_LOW_Temperature);

        // Transitions
        Set_State(sm, sAverage_Wait);
        if(sm->type == State_Driven)
            Set_Event(sm, eRead_Complete);
        else if (sm->type == Table_Driven)
            S_AVERAGE_WAIT.F_READ_COMPLETE(sm);
    }

    // Event Disconnect
    else if(sm->event == eDisconnect)
    {
        if(logger.Get_Log_Level() == lDebug)
        {
            logger.Log_Write(__func__, mDebug, "Event - Alert");
        }

        // Transitions
        Set_State(sm, sDisconnected);
    }
}

/*
 * Function - Event_Handler
 * Brief - handles the events related to state machines

```

```

* Arguments -
* state_machine struct -> which state_machine
* system_state struct -> a pointer to system state global struct
*/
void Event_Handler(state_machine_t* sm, system_state_t* system)
{
    // If state machine doesn't exist, exit
    if (sm == NULL)
        exit(1);

    // Check for state
    switch(sm->state)
    {

        // State - Temperature Reading
        case sTemperature_Reading:
            Temperature_Reading_State(sm, system);
            break;

        // State - Average Wait
        case sAverage_Wait:
            Average_Wait_State(sm, system);
            break;

        // State - Temperature Alert
        case sTemperature_Alert:
            Temperature_Alert_State(sm, system);
            break;

        // State - Disconnect
        case sDisconnected:
            if(logger.Get_Log_Level() == lDebug)
            {
                logger.Log_Write(__func__, mDebug, "State - Disconnected");
            }
            if(sm->event == eDisconnect)
            {
                if(logger.Get_Log_Level() == lDebug)
                {
                    logger.Log_Write(__func__, mDebug, "Event - Disconnect");
                }
            }
        }
    }
}

```

```

        Turn_On_Only_LED(Red);
        logger.Log_Write(__func__, mError, "Disconnect Event Occured, Ending
        Print_Message(__func__, Device_Disconnected);
        State_Machine_End(sm);
        End_Program();
    }
    break;
}

}

//-----
// Functions for Table Driven Event Transitions
void fStart(state_machine_t* sm)
{
    Set_Event(sm, eStart);
}

void fRead_Complete(state_machine_t* sm)
{
    Set_Event(sm, eRead_Complete);
}

void fTimeoutComplete(state_machine_t* sm)
{
    Set_Event(sm, eTimeout_Complete);
}

void fAlert(state_machine_t* sm)
{
    Set_Event(sm, eAlert);
}

void fDisconnect(state_machine_t* sm)
{
    Set_Event(sm, eDisconnect);
}

```

state\_machine/state\_machine.h

```

/**
 * File Name      - state_machine.c
 * Description    - header file for state_machine.c
 * Author         - Atharva Nandanwar
 * Tools          - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL            -
 */

#ifndef STATE_MACHINE_STATE_MACHINE_H_
#define STATE_MACHINE_STATE_MACHINE_H_
#include <stdint.h>
#include <stdlib.h>
#include "logger.h"
#include "errno.h"
#include "tmp102.h"
#include "i2c.h"
#include "led_control.h"

typedef enum {
    eStart = 0x00,
    eRead_Complete = 0x01,
    eAlert = 0x02,
    eTimeout_Complete = 0x04,
    eDisconnect = 0x08,
} event_t;

typedef enum {
    sTemperature_Reading = 0x01,
    sAverage_Wait = 0x02,
    sTemperature_Alert = 0x04,
    sDisconnected = 0x08,
} state_t;

typedef enum {
    State_Driven = 0x01,
    Table_Driven = 0x02,
} state_machine_type_t;

typedef struct {

```

```

    state_t state;
    event_t event;
    state_machine_type_t type;
} state_machine_t;

typedef struct {
    int16_t temperature;
    int16_t average_temperature;
    uint8_t timeout_count;
    uint8_t counter;
    uint8_t timeout_started;
    uint8_t disconnect;
    uint8_t alert;
    uint8_t state_machine_id;
    uint8_t print_flag;
} system_state_t;

typedef struct {
    state_t state_id;
    void (*event_action[5])(state_machine_t* sm);
}state_struct;

void End_Program(void);
state_machine_t* State_Machine_Init(state_machine_type_t);
void State_Machine_End(state_machine_t* sm);
void Set_Event(state_machine_t* sm, event_t event);
void Set_State(state_machine_t* sm, state_t state);
void Event_Handler(state_machine_t* sm, system_state_t* system);
void Print_Message(const char* function_name, error_t error);

#endif /* STATE_MACHINE_STATE_MACHINE_H_ */

```

test/test.c

```

/**
 * File Name      - test.c
 * Description    - contains test cases for the program
 * Author        - Atharva Nandanwar

```

```

* Tools          - GNU C Compiler / ARM Compiler Toolchain
* Leveraged Code -
* URL            -
*/

#include "test.h"

// Global variables
error_t errno;
system_state_t __system = {0, 0, 0, 0, 0, 0, 0, 0};
system_state_t* system_state = &__system;

static inline void delay(void)
{
    for(volatile int i = 10000; i > 0; i--);
}
/*
 * Function - unit_tests
 * Brief - Executes unit tests
 */
void unit_tests(void)
{
    system_state->alert = 0;
    UCUNIT_TestcaseBegin("Starting Test Cases\n\r");
    UCUNIT_TestcaseBegin("Test Case for Write Log\n\r");
    errno = post();
    UCUNIT_CheckIsEqual(errno, POST_Successful);
    logger.Log_Write(__func__, mStatus, "%s", Get_Error_Message(errno));
    UCUNIT_TestcaseEnd();

    uint16_t dummy;
    UCUNIT_TestcaseBegin("Test Case for I2C Read & Write Operation\n\r");
    I2C_Init();
    I2C_Write(TMP102.tmp_LOW_reg_address, 0x00, 0x00);
    delay();
    logger.Log_Write(__func__, mStatus, "I2C Data Receive is %d", dummy = I2C_Rea
    UCUNIT_CheckIsEqual(dummy, 0x0500);
    delay();
    logger.Log_Write(__func__, mStatus, "I2C Data Receive is %d", dummy = I2C_Rea
    UCUNIT_CheckIsEqual(dummy, 0x0000);
    delay();

```



```

logger.Log_Write(__func__, mStatus, "I2C Data Receive is %d", dummy = I2C_Rea
delay();
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("State Machine Testing - State Driven\n\r");
UCUNIT_TestcaseBegin("Test Case for State Machine Init\n\r");
state_machine_t* sm_test = NULL;
sm_test = (state_machine_t *) State_Machine_Init(Table_Driven);
logger.Log_Write(__func__, mStatus, "Testing initial cases");
UCUNIT_CheckIsEqual(sm_test->event, eStart);
UCUNIT_CheckIsEqual(sm_test->state, sTemperature_Reading);
UCUNIT_CheckIsEqual(sm_test->type, Table_Driven);
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("Test Case for State Machine Temperature Reading State\n
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(sm_test->event, eRead_Complete);
UCUNIT_CheckIsEqual(sm_test->state, sAverage_Wait);
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("Test Case for State Machine Temperature Reading Complet
UCUNIT_WriteString("and Transition to Average_Wait state\n\r");
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(sm_test->event, eRead_Complete);
UCUNIT_CheckIsEqual(sm_test->state, sAverage_Wait);
logger.Log_Write(__func__, mStatus, "%d", system_state->counter);
UCUNIT_CheckIsEqual(system_state->timeout_started, 1);
UCUNIT_TestcaseEnd();

while(system_state->counter != 150);

UCUNIT_TestcaseBegin("Test Case for State Machine Timeout\n\r");
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(sm_test->event, eTimeout_Complete);
UCUNIT_CheckIsEqual(sm_test->state, sAverage_Wait);
UCUNIT_CheckIsEqual(system_state->timeout_started, 0);
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("Test Case for State Machine Temperature Reading after T
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(sm_test->event, eStart);

```

```

UCUNIT_CheckIsEqual(sm_test->state, sTemperature_Reading);
UCUNIT_CheckIsEqual(system_state->timeout_started, 0);
UCUNIT_TestcaseEnd();

while(1)
{
    if(system_state->alert == 1)
        break;
}

UCUNIT_TestcaseBegin("Test Case for State Machine Alert in Reading State\n\r"
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(sm_test->event, eAlert);
UCUNIT_CheckIsEqual(sm_test->state, sTemperature_Alert);
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("Test Case for State Machine Alert State\n\r");
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(sm_test->event, eRead_Complete);
UCUNIT_CheckIsEqual(sm_test->state, sAverage_Wait);
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("Test Case for State Machine Alert Addressed, shifting t
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(system_state->counter, 0);
UCUNIT_CheckIsEqual(sm_test->event, eRead_Complete);
UCUNIT_CheckIsEqual(sm_test->state, sAverage_Wait);
UCUNIT_TestcaseEnd();

while(system_state->counter < 150);

UCUNIT_TestcaseBegin("Test Case for State Machine Timeout\n\r");
Event_Handler(sm_test, system_state);
UCUNIT_CheckIsEqual(system_state->counter, 150);
UCUNIT_CheckIsEqual(sm_test->event, eTimeout_Complete);
UCUNIT_CheckIsEqual(sm_test->state, sAverage_Wait);
UCUNIT_CheckIsEqual(system_state->timeout_started, 0);
UCUNIT_TestcaseEnd();

UCUNIT_TestcaseBegin("Test Case for State Machine Temperature Reading after T
Event_Handler(sm_test, system_state);

```

```

    UCUNIT_CheckIsEqual(sm_test->event, eStart);
    UCUNIT_CheckIsEqual(sm_test->state, sTemperature_Reading);
    UCUNIT_TestcaseEnd();

    UCUNIT_TestcaseBegin("Test Case for State Machine Disconnect\n\r");
    system_state->disconnect = 1;
    Event_Handler(sm_test, system_state);
    UCUNIT_CheckIsEqual(sm_test->event, eDisconnect);
    UCUNIT_CheckIsEqual(sm_test->state, sDisconnected);
    UCUNIT_TestcaseEnd();
}

/*
 * Function - Main
 * Brief - Main testing routine
 */
int main(void)
{
    //Initializing board pins
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
    BOARD_InitDebugConsole();
    I2C_Init();
    I2C_Write(TMP102.config_reg_address, 0x78, 0x80);
    for(volatile int i = 10000; i > 0; i--);
    I2C_Write(TMP102.tmp_HI_reg_address, 0x05, 0x00);
    for(volatile int i = 10000; i > 0; i--);
    I2C_Write(TMP102.tmp_LOW_reg_address, 0x00, 0x00);
    I2C_Alert_Init();
    //logger.Set_Log_Level(Test);

    //Calling function to run tests
    unit_tests();
    return 0;
}

/*
 * Function - SysTick_Handler
 * Brief - SysTick interrupt handler
 */

```

```

void SysTick_Handler(void)
{
    // Count up if timeout timer is started
    if(system_state->timeout_started)
        system_state->counter++;
}

/*
 * Function - PORTA_IRQHandler
 * Brief - Port A interrupt handler
 */
void PORTA_IRQHandler(void)
{
    // If interrupt from Pin 5, then set alert
    if(PORTA->ISFR & ALERT_PIN)
    {
        system_state->alert = 1;
    }
    // Clearing the interrupt
    PORTA->PCR[5] |= PORT_PCR_ISF_MASK;
}

```

## test/test.h

```

/**
 * File Name      - test.h
 * Description    - header file for test.c
 * Author        - Atharva Nandanwar
 * Tools         - GNU C Compiler / ARM Compiler Toolchain
 * Leveraged Code -
 * URL           -
 */

#ifndef TEST_TEST_H_
#define TEST_TEST_H_
#include <stdint.h>
#include <stdio.h>

```

```
#include "pin_mux.h"
#include "peripherals.h"
#include "clock_config.h"
#include "board.h"

#include "System.h"
#include "uCUnit.h"

#include "state_machine.h"
#include "logger.h"
#include "errno.h"
#include "i2c.h"
#include "tmp102.h"
#include "post.h"

#endif /* TEST_TEST_H_ */
```

