ECEN 5813

Chutao Wei

Curry Buscher

PES Project 4 Code pdf

readme.md

```
# cu-ecen-5813-project-4
**Title:**
PES Project 4 Readme <br/>
**Name:**
Curry Buscher, Chutao Wei <br/>
**Repository Comments:** <br/>
In documents folder: <br/>
There are PES Project 4.pdf, and state machine diagram.jpg<br/>
In source folder: <br/>
main.c: main function wrapper has two versions. One runs the test script
without command line, one require user to put in command in console.<br/>
memory_utility.c/h: contains all memory utility functions.<br/>
pattern_gen.c/h: generate random byte array using linear feedback shit
register<br/>
led.c/h: contains RGB LED control functions<br/>
timer.c/h: contains only blocking delay function for now<br/>
gpio.c/h: contains gpio control functions<br/>
state.c/h: state machine function<br/>
touch_sen.c/h: contains touch sensor printing function<br/>
mma8451.c/h: contains mma8451 accelerameter function<br/>
test.c: contains test function for uCUnit testfunction<br/>
uCUnit.c/h: uCUnit test function<br/>
System.c/h: System for uCUnit<br/>
(see more details in PES Project 4.pdf) <br/>
**Project Comments:**
Please use semihost <br/>

### **Installation/Execution/Editing Notes:**<br/>

**Language:**
C<br/>
**Compiler:**
GCC version 7.4.0<br/>
**IDE:**
MCUExpresso<br/>
**Build Environment:**
Ubuntu 16 or up<br/>
**Target Environment:**
KL25Z/Linux<br/>
**License:**
MIT<br/>
```

Project_4.c (main.c)

```
/*
 * Copyright 2016-2020 NXP
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice,
this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright
notice, this
 *   list of conditions and the following disclaimer in the documentation
and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from
this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * @file    Project_4.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
```

```c
#include "i2c.h"
#include "gpio.h"
#include "led.h"
#include "mma8451.h"
#include "touch_sen.h"
#include "timer.h"
#include "state.h"
/*
 * @brief   Application entry point.
 */
int main(void) {

    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();

    /* Init Systick */
    Init_SysTick();

    /* Init LED */
    init_LED();
    turn_LED_blue(on);
    /* Init Touch Sensor */
    Touch_Init();

    /* Init i2c for MMA8451 */
    i2c_init();

    PRINTF("Hello, PES Project 4\n");

    /* Init MMA8451 */
    if(init_mma8451())
    {
        PRINTF("MMA8451 connection error\n");
        // if connection error, just halt the program
        // and turn LED red
        turn_LED_blue(off);
        turn_LED_red(on);
        while(1);
    }
    turn_LED_blue(off);
    turn_LED_green(on);
    /* Enter an infinite loop */
    while(1)
    {
        RunMachines();
    }
    return 0 ;
}
```

timer.c (main.c)

```
/*
 * timer.c
 *
 *  Created on: Feb 11, 2020
 *      Author: chutao
 */

/*************** Include *****************/

#include <logger.h>
#include <stdint.h>
#include <stdbool.h>
#include "state.h"

/*************** Define *****************/

#define INCLUDE_LOG_DEBUG 1
#define CPU_FREQ_MHZ    (48)
#define NUM_ASSE_FOR    (7)
#define DELAY_MS_TO_LOOP_COUNT(msec)\
    ((uint32_t)((msec*(CPU_FREQ_MHZ*1000))/(NUM_ASSE_FOR)))
//#define BLOCKWAITING

/*************** Global Variables *****************/

const uint32_t delay_look_up_table[] = {
        DELAY_MS_TO_LOOP_COUNT(500),
        DELAY_MS_TO_LOOP_COUNT(1000),
        DELAY_MS_TO_LOOP_COUNT(2000),
        DELAY_MS_TO_LOOP_COUNT(3000)};

uint64_t msec_count = 0;
uint64_t target_msec_count = 0;
bool delay_flag = 0;
/* Time out count for kWaitPollSlider state */
uint8_t timeout_count = 0;
/*************** Interrupt Hanlder ***************/

void SysTick_Handler(void)
{
    msec_count ++;
    if (delay_flag == true)
    {
        if (msec_count == target_msec_count)
        {
            delay_flag = false;
            timeout_count++;
            if (timeout_count == 6)
            {
                timeout_count = 0;
                SetEvent(Timeout_6);
            }
            else
            {
                SetEvent(Timeout_1_5);
```

```
            }
        }
    }
}

/*************** Function *****************/

void Init_SysTick(void) {
    SysTick->LOAD = (48000L-1L);// count 1 msec
    NVIC_SetPriority(SysTick_IRQn, 3); // enable NVIC
    SysTick->VAL = (480000L-1L); // reset count value
    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk |
SysTick_CTRL_ENABLE_Msk;

}

#ifdef BLOCK_WAITING
// Block waiting function abandoned for now
void mdelay(uint32_t msec)
{
    LOG_DEBUG("Blocking wait for %d msec", msec);
    uint32_t i = 0;
    uint32_t delay_count = 0;
    if (msec == 500)
    {
        delay_count = delay_look_up_table[0];
    }
    else if (msec == 1000)
    {
        delay_count = delay_look_up_table[1];
    }
    else if (msec == 2000)
    {
        delay_count = delay_look_up_table[2];
    }
    else if (msec == 3000)
    {
        delay_count = delay_look_up_table[3];
    }
    else
    {
        LOG_ERROR("Unexpected msec value, has to be 500, 1000, 2000, 3000");
    }
    for(i=0; i<delay_count; i++);
}
#else
// Interrupt waiting function4
void mdelay(uint32_t msec)
{
    LOG_DEBUG("Interrupt wait for %d msec", msec);
    // read current count
    target_msec_count = msec_count + msec;
    delay_flag = true;
}
#endif
```

i2c.c

```
/*
 * i2c.c
 *
 *  Created on: Mar 31, 2020
 *      Author: user
 */
//adapted from https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter 8/I2C-Demo/


#include <MKL25Z4.h>
#include "i2c.h"
int lock_detect=0;
int i2c_lock=0;

//init i2c0
void i2c_init(void)
{
    //clock i2c peripheral and port E
    SIM->SCGC4 |= SIM_SCGC4_I2C0_MASK;
    SIM->SCGC5 |= (SIM_SCGC5_PORTE_MASK);

    //set pins to I2C function
    PORTE->PCR[24] |= PORT_PCR_MUX(5);
    PORTE->PCR[25] |= PORT_PCR_MUX(5);

    //set to 100k baud
    //baud = bus freq/(scl_div+mul)
    //~400k = 24M/(64); icr=0x12 sets scl_div to 64

    I2C0->F = (I2C_F_ICR(0x10) | I2C_F_MULT(0));

    //enable i2c and set to master mode
    I2C0->C1 |= (I2C_C1_IICEN_MASK);

    // Select high drive mode
    I2C0->C2 |= (I2C_C2_HDRS_MASK);
}


void i2c_busy(void){
    // Start Signal
    lock_detect=0;
    I2C0->C1 &= ~I2C_C1_IICEN_MASK;
    I2C_TRAN;
    I2C_M_START;
    I2C0->C1 |=  I2C_C1_IICEN_MASK;
    // Write to clear line
    I2C0->C1 |= I2C_C1_MST_MASK; /* set MASTER mode */
    I2C0->C1 |= I2C_C1_TX_MASK; /* Set transmit (TX) mode */
    I2C0->D = 0xFF;
    while ((I2C0->S & I2C_S_IICIF_MASK) == 0U) {
    } /* wait interrupt */
    I2C0->S |= I2C_S_IICIF_MASK; /* clear interrupt bit */
```

```
                            /* Clear arbitration error flag*/
    I2C0->S |= I2C_S_ARBL_MASK;


                            /* Send start */
    I2C0->C1 &= ~I2C_C1_IICEN_MASK;
    I2C0->C1 |= I2C_C1_TX_MASK; /* Set transmit (TX) mode */
    I2C0->C1 |= I2C_C1_MST_MASK; /* START signal generated */

    I2C0->C1 |= I2C_C1_IICEN_MASK;
                            /*Wait until start is send*/

                            /* Send stop */
    I2C0->C1 &= ~I2C_C1_IICEN_MASK;
    I2C0->C1 |= I2C_C1_MST_MASK;
    I2C0->C1 &= ~I2C_C1_MST_MASK; /* set SLAVE mode */
    I2C0->C1 &= ~I2C_C1_TX_MASK; /* Set Rx */
    I2C0->C1 |= I2C_C1_IICEN_MASK;


                                /* wait */
                            /* Clear arbitration error & interrupt flag*/
    I2C0->S |= I2C_S_IICIF_MASK;
    I2C0->S |= I2C_S_ARBL_MASK;
    lock_detect=0;
    i2c_lock=1;
}

//#pragma no_inline
void i2c_wait(void) {
    lock_detect = 0;
    while(((I2C0->S & I2C_S_IICIF_MASK)==0) & (lock_detect < 200)) {
        lock_detect++;
    }
    if (lock_detect >= 200)
        i2c_busy();
    I2C0->S |= I2C_S_IICIF_MASK;
}

//send start sequence
void i2c_start()
{
    I2C_TRAN;                       /* set to transmit mode */
    I2C_M_START;                    /* send start */
}

//send device and register addresses
//#pragma no_inline
void i2c_read_setup(uint8_t dev, uint8_t address)
{
    I2C0->D = dev<<1;           /* send dev address */
    I2C_WAIT                    /* wait for completion */

    I2C0->D = address;          /* send read address */
    I2C_WAIT                    /*wait for completion */
```

```
    I2C_M_RSTART;           /* repeated start */
    I2C0->D = (dev<<1)|0x1; /* send dev address (read) */
    I2C_WAIT                /* wait for completion */

    I2C_REC;                /* set to receive mode */
}

//read a byte and ack/nack as appropriate
// #pragma no_inline
uint8_t i2c_repeated_read(uint8_t isLastRead)
{
    uint8_t data;

    lock_detect = 0;

    if(isLastRead)  {
        NACK;                   /* set NACK after read */
    } else  {
        ACK;                    /* ACK after read */
    }

    data = I2C0->D;             /* dummy read */
    I2C_WAIT                    /* wait for completion */

    if(isLastRead)  {
        I2C_M_STOP;             /* send stop */
    }
    data = I2C0->D;             /* read data */

    return  data;
}



//////////funcs for reading and writing a single byte
//using 7bit addressing reads a byte from dev:address
// #pragma no_inline
uint8_t i2c_read_byte(uint8_t dev, uint8_t address)
{
    uint8_t data;

    I2C_TRAN;               /* set to transmit mode */
    I2C_M_START;            /* send start */
    I2C0->D = (dev<<1);     /* send dev address */
    I2C_WAIT                /* wait for completion */

    I2C0->D = address;      /* send read address */
    I2C_WAIT                /* wait for completion */

    I2C_M_RSTART;           /* repeated start */
    I2C0->D = (dev<<1)|0x1; /* send dev address (read) */
    I2C_WAIT                /* wait for completion */

    I2C_REC;                /* set to recieve mode */
    NACK;                   /* set NACK after read */

    data = I2C0->D;         /* dummy read */
```

```
    I2C_WAIT                /* wait for completion */

    I2C_M_STOP;             /* send stop */
    data = I2C0->D;         /* read data */

    return data;
}



//using 7bit addressing writes a byte data to dev:address
//#pragma no_inline
void i2c_write_byte(uint8_t dev, uint8_t address, uint8_t data)
{

    I2C_TRAN;               /* set to transmit mode */
    I2C_M_START;            /* send start */
    I2C0->D = (dev<<1);     /* send dev address */
    I2C_WAIT                /* wait for ack */

    I2C0->D = address;      /* send write address */
    I2C_WAIT                /* wait for ack */

    I2C0->D = data;         /* send data */
    I2C_WAIT                /* wait for ack */
    I2C_M_STOP;             /* send stop */

}
```

touch_sen.c

```
/*
 * touch_sen.c
 *
 *  Created on: Feb 11, 2020
 *      Author: Ben Roloff
 *      link:
https://www.digikey.com/eewiki/display/microcontroller/Using+the+Capacitive+Touch+Sensor+on+the+FRDM-KL46Z
 *      Editor: chutao
 */

#include "MKL25Z4.h"
#include "touch_sen.h"
#include "fsl_debug_console.h"
#include "logger.h"


//#define INCLUDE_LOG_DEBUG 1


tsi_position_t position;
#ifdef L_M_R_POSITION_MODE
const char * position_string[3] ={"left","middle","right"};
#endif

#ifdef L_R_POSITION_MODE
const char * position_string[3] ={"left","right"};
#endif

// TSI initialization function
void Touch_Init()
{
    // Enable clock for TSI PortB 16 and 17
    SIM->SCGC5 |= SIM_SCGC5_TSI_MASK;


    TSI0->GENCS = TSI_GENCS_OUTRGF_MASK |  // Out of range flag, set to 1 to
clear
                                //TSI_GENCS_ESOR_MASK |  // This is disabled
to give an interrupt when out of range.  Enable to give an interrupt when end
of scan
                                TSI_GENCS_MODE(0u) |  // Set at 0 for
capacitive sensing.  Other settings are 4 and 8 for threshold detection, and
12 for noise detection
                                TSI_GENCS_REFCHRG(0u) | // 0-7 for Reference
charge
                                TSI_GENCS_DVOLT(0u) | // 0-3 sets the Voltage
range
                                TSI_GENCS_EXTCHRG(0u) | //0-7 for External
charge
                                TSI_GENCS_PS(0u) | // 0-7 for electrode
prescaler
                                TSI_GENCS_NSCN(31u) | // 0-31 + 1 for number
of scans per electrode
                                TSI_GENCS_TSIEN_MASK | // TSI enable bit
```

```
                                    //TSI_GENCS_TSIIEN_MASK | //TSI interrupt is
disables
                                    TSI_GENCS_STPE_MASK | // Enables TSI in low
power mode
                                    //TSI_GENCS_STM_MASK | // 0 for software
trigger, 1 for hardware trigger
                                    //TSI_GENCS_SCNIP_MASK | // scan in progress
flag
                                    TSI_GENCS_EOSF_MASK ; // End of scan flag,
set to 1 to clear
                                    //TSI_GENCS_CURSW_MASK; // Do not swap
current sources
}

// Function to read touch sensor from low to high capacitance for left to
right
int Touch_Scan_LH(void)
{
    int scan;
    TSI0->DATA =  TSI_DATA_TSICH(10u); // Using channel 10 of The TSI
    TSI0->DATA |= TSI_DATA_SWTS_MASK; // Software trigger for scan
    scan = SCAN_DATA;
    TSI0->GENCS |= TSI_GENCS_EOSF_MASK ; // Reset end of scan flag

    return scan - SCAN_OFFSET;
}

// Function to read touch sensor from high to low capacitance for left to
right
int Touch_Scan_HL(void)
{
    int scan;
    TSI0->DATA =    TSI_DATA_TSICH(9u); // Using channel 9 of the TSI
    TSI0->DATA |=   TSI_DATA_SWTS_MASK; // Software trigger for scan
    scan = SCAN_DATA;
    TSI0->GENCS |=  TSI_GENCS_EOSF_MASK ; // Reset end of scan flag

    return scan - SCAN_OFFSET;
}

// Function to scan the position of slider
tsi_position_t Touch_Scan_Position(void)
{
    LOG_DEBUG("Scan Capacitance");
    // read capacitance value
    int LH_value = Touch_Scan_LH();
    int HL_value = Touch_Scan_HL();

    LOG_DEBUG("LH_value: %d, HL_value: %d", LH_value, HL_value);

#ifdef L_M_R_POSITION_MODE
    // Check the capacitance, and decide the position
    if ((LH_value>50) && (LH_value<350) && (HL_value>50) && (HL_value<350))
    {
        position = left;
    }
```

```
    else if ((LH_value>450) && (LH_value<750) && (HL_value>450) &&
(HL_value<750))
    {
        position = middle;
    }
    else if ((LH_value>800) && (LH_value<1200) && (HL_value>800) &&
(HL_value<1200))
    {
        position = right;
    }
    LOG_DEBUG("Slider Position is  %s", position_string[position]);
#endif


#ifdef L_R_POSITION_MODE
    // Check the capacitance, and decide the position
    if ((LH_value>0) && (LH_value<1000) && (HL_value>0) && (HL_value<1000))
    {
        position = left;
    }
    else if ((LH_value>1100) && (LH_value<3000) && (HL_value>1100) &&
(HL_value<3000))
    {
        position = right;
    }
    else
    {
        position = unknown;
        LOG_DEBUG("Slider Position is a little off, calibration needed,
default to left");
    }
    LOG_DEBUG("Slider Position is  %s", position_string[position]);
#endif

    return (position);

}
```

state.c

```
/*
 * state.c
 *
 *  Created on: Mar 31, 2020
 *      Author: user
 */

/********************** Include **********************/

#include <stdio.h>
#include "state.h"
#include "timer.h"
#include "mma8451.h"
#include "touch_sen.h"
#include "logger.h"
#include "led.h"

/********************** Global Variables **********************/

/* State machine relted varibles */
tEvent event = RightSlider;
tMachine machine=mStateCentric;
tState currentState= kReadXYZ;

/* State Table used by RunTableDriven() function */
struct sStateTableEntry stateTable[] = {
/* CurrentState         Action function  Event:complete   timeout_1_5,
timout_6, left_slider, right_slider */
/* kReadXYZ */          {state_ReadXYZ, {kProcessDisplay, kError,
kError,   kError,      kError      }},
/* kProcessDisplay */   {state_Display, {kWaitPollSlider, kError,
kError,   kError,      kError      }},
/* kWaitPollSlider */   {state_WaitPoll,{kError,          kReadXYZ,
kReadXYZ, kReadXYZ,    kEnd        }},
/* kend */              {state_End,     {kEnd,            kEnd,        kEnd,
kEnd,        kEnd        }},
/* kerror */            {state_Error,   {kError,          kError,
kError,   kError,      kError      }}
};

const char * state_machine_type_string[2] ={"StateCentric","TableDriven"};
/********************** Function **********************/

/* State Machine action functions */
void state_ReadXYZ(){
    read_full_xyz();
    SetEvent(Complete);
}

void state_Display(){
    display_dataset();
    SetEvent(Complete);
}

void state_WaitPoll(){
```

```
    mdelay(3000);

    tsi_position_t final_pos;
    // wait until SysTick_Handler set timeout event flag
    uint32_t left_pos_count = 0;
    uint32_t right_pos_count = 0;
    while(event == Complete)
    {
        // keep polling slider position
        tsi_position_t pos = Touch_Scan_Position();
        if (pos == left)
            left_pos_count++;
        else if (pos == right)
            right_pos_count++;
    }

    if (left_pos_count > right_pos_count)
        final_pos = left;
    else if (left_pos_count < right_pos_count)
        final_pos = right;
    else
        final_pos = left;
    // if slider is right or event is timeout_6
    if ((final_pos == right) || (event == Timeout_6))
    {
        // change to the other state machine
        machine = machine ^ 0x1;
    }
}

void state_End(){
    // halt the program
    turn_LED_green(off);
    turn_LED_red(on);
    while(1);
}

void state_Error(){
    // halt the program
    while(1);
}

void SetEvent(tEvent evt){
    event = evt;
}

/* state machine related functions */
void RunMachines(void){
    LOG_INFO("current state machine
is %s",state_machine_type_string[machine]);
    switch(machine){
        case mStateCentric:
            RunStateCentric();
            break;
        case mTableDriven:
            RunTableDriven();
            break;
```

```
    }
}

void RunStateCentric(void){
    //sStateTableEntry *currentState= stateTable[kReadXYZ];
    /* Do the action in this state */
    stateTable[currentState].func_p();

    /* Set the state to go next */
    switch(currentState){
    case kReadXYZ:
        if (event==Complete){
            HandleEventComplete();
        }
        else{
            printf("last event is %d",(int)event);
        }
        break;
    case kProcessDisplay:
        if (event==Complete){
            HandleEventComplete();
        }
        else{
            printf("last event is %d",(int)event);
        }
        break;
    case kWaitPollSlider:
        if (event==Timeout_1_5){
            HandleEventTimeout_1_5();
        }
        else if (event==Timeout_6){
            HandleEventTimeout_6();
        }
        else if (event==LeftSlider){
            HandleEventLeftSlider();
        }
        else if (event==RightSlider){
            HandleEventRightSlider();
        }
        else{
            LOG_ERROR("last event is %d",(int)event);
        }
        break;
    default:
        LOG_ERROR("Error state reached");
        LOG_ERROR("last event is %d",(int)event);
        break;
    }
}


//kReadXYZ, kProcessDisplay, kSensorDisconnect, kWaitPollSlider
void RunTableDriven(void){
    //sStateTableEntry *currentState= stateTable[kReadXYZ];
    /* Do the action in this state */
    stateTable[currentState].func_p();
```

```
    /* Set the state to go next */
    currentState=stateTable[currentState].next_state[event];
}

void HandleEventComplete(void){
    currentState=stateTable[currentState].next_state[Complete];
}
void HandleEventTimeout_1_5(void){
    currentState=stateTable[currentState].next_state[Timeout_1_5];
}
void HandleEventTimeout_6(void){
    currentState=stateTable[currentState].next_state[Timeout_6];
}
void HandleEventLeftSlider(void){
    currentState=stateTable[currentState].next_state[LeftSlider];
}
void HandleEventRightSlider(void){
    currentState=stateTable[currentState].next_state[RightSlider];
}
```

mma8451.c

```
/*
 * mma8451.c
 *
 *  Created on: Apr 1, 2020
 *      Author: user
 */
//adapted from https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter 8/I2C-Demo/

/*************** Include *****************/

#include <MKL25Z4.h>
#include "mma8451.h"
#include "i2c.h"
#include "timer.h"
#include "logger.h"

/*************** Global *****************/
#define MAX_DATA_SET_COUNT  3

/*************** Global *****************/
// data variables
uint32_t data_set_count = 0;
uint32_t current_data_set_num = 0;
int16_t acc_x[MAX_DATA_SET_COUNT]; // syntax for every entry to be 0
int16_t acc_y[MAX_DATA_SET_COUNT];
int16_t acc_z[MAX_DATA_SET_COUNT];


/*************** Function *****************/
int16_t average(int16_t arr[], uint32_t n)
{
    int i;
    int average = arr[0];
    for (i = 0; i < n; i++)
        average = average + arr[i];
    return (average/n);
}
// source: https://www.geeksforgeeks.org/c-program-find-largest-element-array/
int16_t max(int16_t arr[], uint32_t n)
{
    int i;
    int max = arr[0];
    for (i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

int16_t min(int16_t arr[], uint32_t n)
{
    int i;
    int min = arr[0];
    for (i = 1; i < n; i++)
```

```
        if (arr[i] < min)
            min = arr[i];
    return min;
}

int init_mma8451()
{
    // make sure device is connected
    uint8_t whoami_val = 0;
    whoami_val = i2c_read_byte(MMA_ADDR, REG_WHOAMI);
    if (whoami_val != WHOAMI)
    {
        // error, either not connected
        // or corrupted data
        return 1;
    }

    // set active mode, 14 bit samples and 800 Hz ODR
    i2c_write_byte(MMA_ADDR, REG_CTRL1, 0x03);
    return 0;
}

int verify_connect_mma8451()
{
    // make sure device is connected
    uint8_t whoami_val = 0;
    whoami_val = i2c_read_byte(MMA_ADDR, WHOAMI);
    if (whoami_val != 0x1A)
    {
        // error, either not connected
        // or corrupted data
        return 1;
    }

    return 0;
}


void read_full_xyz()
{
    int i;
    uint8_t data[6];
    int16_t temp[3];

    i2c_start();
    i2c_read_setup(MMA_ADDR , REG_XHI);

    // Read five bytes in repeated mode
    for( i=0; i<5; i++) {
        data[i] = i2c_repeated_read(0);
    }
    // Read last byte ending repeated mode
    data[i] = i2c_repeated_read(1);

    for ( i=0; i<3; i++ ) {
        temp[i] = (int16_t) ((data[2*i]<<8) | data[2*i+1]);
    }
```

```
    // Align for 14 bits
    // assign to the current array
    acc_x[current_data_set_num] = temp[0]>>2;
    acc_y[current_data_set_num] = temp[1]>>2;
    acc_z[current_data_set_num] = temp[2]>>2;

    // update data_set_count
    if (data_set_count != MAX_DATA_SET_COUNT)
        data_set_count ++;

    // update current data_set
    if (current_data_set_num == 2)
        current_data_set_num = 0;
    else if ((data_set_count==1)||(current_data_set_num==0))
        current_data_set_num = current_data_set_num;
    else
        current_data_set_num ++;
}


void read_xyz(void)
{
    // sign extend byte to 16 bits - need to cast to signed since function
    // returns uint8_t which is unsigned
    acc_x[current_data_set_num] = (int8_t) i2c_read_byte(MMA_ADDR,
REG_XHI)<<6;
    mdelay(100);
    acc_y[current_data_set_num] = (int8_t) i2c_read_byte(MMA_ADDR,
REG_YHI)<<6;
    mdelay(100);
    acc_z[current_data_set_num] = (int8_t) i2c_read_byte(MMA_ADDR,
REG_ZHI)<<6;

    // update data_set_count
    if (data_set_count != MAX_DATA_SET_COUNT)
        data_set_count ++;

    // update current data_set
    if (current_data_set_num == 2)
        current_data_set_num = 0;
    else if ((data_set_count==1)||(current_data_set_num==0))
        current_data_set_num = current_data_set_num;
    else
        current_data_set_num ++;

}

void display_dataset()
{
    // Last X, Y, Z readings and the state entry count
    LOG_INFO("Last: acc_x = %d, acc_y = %d, acc_z = %d",
            acc_x[current_data_set_num],
            acc_y[current_data_set_num],
            acc_z[current_data_set_num]);

    // Average X, Y, Z readings
```

```
    LOG_INFO("Average: acc_x = %d, acc_y = %d, acc_z = %d",
            average(acc_x,MAX_DATA_SET_COUNT),
            average(acc_y,MAX_DATA_SET_COUNT),
            average(acc_z,MAX_DATA_SET_COUNT));

    // Low X, Y, Z readings
    LOG_INFO("Low: acc_x = %d, acc_y = %d, acc_z = %d",
            min(acc_x,MAX_DATA_SET_COUNT),
            min(acc_y,MAX_DATA_SET_COUNT),
            min(acc_z,MAX_DATA_SET_COUNT));

    // High X, Y, Z readings
    LOG_INFO("High: acc_x = %d, acc_y = %d, acc_z = %d",
            max(acc_x,MAX_DATA_SET_COUNT),
            max(acc_y,MAX_DATA_SET_COUNT),
            max(acc_z,MAX_DATA_SET_COUNT));

}
```

gpio.c

```c
/*
 * gpio.c
 *
 *  Created on: Feb 11, 2020
 *      Author: chutao
 *
 *  Minic the functions from fsl_gpio.c
 *  Still use MKL25Z4.h for hardware addresses
 */

/********************** Include **********************/

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


/********************** Function **********************/

void set_GPIO_Pinout(GPIO_Type *port, uint32_t pin)
{
    port->PSOR = (0x1 << pin);
}

void clear_GPIO_Pinout(GPIO_Type *port, uint32_t pin)
{
    port->PCOR = (0x1 << pin);
}

void toggle_GPIO_Pinout(GPIO_Type *port, uint32_t pin)
{
    port->PTOR = (0x1 << pin);
}

void init_GPIO_Pin(GPIO_Type *port, uint32_t pin,
        gpio_pin_direct_t pin_direction, uint8_t pin_data)
{
    if (pin_direction == GPIO_DigitalInput)
    {
        // Set pin to input direction
        port->PDDR &= ~(0x1 << pin);
    }
    else if (pin_direction == GPIO_DigitalOutput)
    {
        // Set pin to output direction
        port->PDDR |= (0x1 << pin);
        if (pin_data)
        {
            set_GPIO_Pinout(port,pin);
        }
        else
        {
            clear_GPIO_Pinout(port,pin);
        }
    }
```

```
    else
    {
#ifdef LOGGING_DEBUG
        // TODO: Debug message
#endif
    }
}
```

led.c

```c
/*
 * led.c
 *
 *  Created on: Feb 11, 2020
 *      Author: chutao
 */


/*************** Include *****************/

#include <logger.h>
#include <stdint.h>
#include "gpio.h"
#include "led.h"
#include "timer.h"

/*************** Global Varibles *****************/

led_color_t color = red;
const char * led_color_string[3] ={"on","off","toggle"};


/*************** Function *****************/

void init_LED(void)
{
    init_GPIO_Pin(LED3_RED_PORT,    LED3_RED_PIN,   GPIO_DigitalOutput, 1);
    init_GPIO_Pin(LED3_GREEN_PORT,  LED3_GREEN_PIN, GPIO_DigitalOutput, 1);
    init_GPIO_Pin(LED3_BLUE_PORT,   LED3_BLUE_PIN,  GPIO_DigitalOutput, 1);
}


void turn_LED(led_state_t LED_state)
{
    if (color == red)
    {
        turn_LED_red(LED_state);
    }
    else if (color == green)
    {
        turn_LED_green(LED_state);
    }
    else if (color == blue)
    {
        turn_LED_blue(LED_state);
    }
    else
    {
        LOG_ERROR("Unexpected led_state_t");
    }
}

void change_LED_color(led_color_t LED_color)
{
    color = LED_color;
```

```
}

void turn_LED_red(led_state_t LED_state)
{
    LOG_DEBUG("Turn LED red %s",led_color_string[LED_state]);
    color = red;
    if (LED_state == off)
    {
        set_GPIO_Pinout(LED3_RED_PORT, LED3_RED_PIN);
    }
    else if (LED_state == on)
    {
        clear_GPIO_Pinout(LED3_RED_PORT, LED3_RED_PIN);
    }
    else if (LED_state == toggle)
    {
        toggle_GPIO_Pinout(LED3_RED_PORT, LED3_RED_PIN);
    }
    else
    {
        LOG_ERROR("Unexpected led_state_t");
    }
}

void turn_LED_green(led_state_t LED_state)
{
    LOG_DEBUG("Turn LED green %s",led_color_string[LED_state]);
    color = green;
    if (LED_state == off)
    {
        set_GPIO_Pinout(LED3_GREEN_PORT, LED3_GREEN_PIN);
    }
    else if (LED_state == on)
    {
        clear_GPIO_Pinout(LED3_GREEN_PORT, LED3_GREEN_PIN);
    }
    else if (LED_state == toggle)
    {
        toggle_GPIO_Pinout(LED3_GREEN_PORT, LED3_GREEN_PIN);
    }
    else
    {
        LOG_ERROR("Unexpected led_state_t");
    }
}

void turn_LED_blue(led_state_t LED_state)
{
    LOG_DEBUG("Turn LED blue %s",led_color_string[LED_state]);
    color = blue;
    if (LED_state == off)
    {
        set_GPIO_Pinout(LED3_BLUE_PORT, LED3_BLUE_PIN);
    }
    else if (LED_state == on)
    {
        clear_GPIO_Pinout(LED3_BLUE_PORT, LED3_BLUE_PIN);
```

```
    }
    else if (LED_state == toggle)
    {
        toggle_GPIO_Pinout(LED3_BLUE_PORT, LED3_BLUE_PIN);
    }
    else
    {
        LOG_ERROR("Unexpected led_state_t");
    }
}
```

logger.c

```c
/*
 * log.c
 *
 *  Created on: Dec 18, 2018
 *      Author: Chutao Wei
 *
 */
/********************** Include **********************/

#include <stdint.h>
#include "logger.h"


/********************** Global **********************/

log_status_t log_status = disable;


/********************** Functions **********************/
void Log_enable(void)
{
    log_status = enable;
}

void Log_disable(void)
{
    log_status = disable;
}

log_status_t Log_status(void)
{
    return log_status;
}

void Log_data(uint32_t data)
{
    if(log_status == enable)
    {
        printf("Addr: 0x%08x, Data: 0x%08x",&data,data);
    }
}

void Log_string(const char * str)
{
    if(log_status == enable)
    {
        printf("%s\n", str);
    }
}

void Log_integer(int integer)
{
    if(log_status == enable)
    {
        printf("%d\n", integer);
```

```
        }
}
```

test.c