```c
/*
 * Level_project.c
 *
 *  Created on: Dec 3, 2020
 *
 *      Author: aditya.vny95
 *           Reference: https://github.com/alexander-g-
dean/ESF/tree/master/NXP/Code/Chapter_8
 *
 */

/*-----------------------------------------------------------------------------
 *-----------------------------------------------------------------------------*/
#include <MKL25Z4.H>
#include <stdio.h>
#include <math.h>
#include <board.h>
#include "gpio_defs.h"
#include "LEDs.h"
#include "i2c.h"
#include "mma8451.h"
#include "delay.h"
#include "fsl_debug_console.h"
#include "peripherals.h"
#include "pin_mux.h"

#define PWM_STEP_SIZE (2.83)
                          // 255/90  i.e. PWM max value divided by the maximum angle
calculated


/*-----------------------------------------------------------------------------
  MAIN function
 *-----------------------------------------------------------------------------*/
int main (void) {

       BOARD_InitBootPins();
                                  // All the necessary initializations
       BOARD_InitBootClocks();
       BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
       BOARD_InitDebugConsole();
#endif
       Init_RGB_LEDs();
       i2c_init();
                                        // Initialize I2C
       PRINTF("Welcome to the LEVEL\n\r");
       if (!init_mma()) {
                                  // Initialize MMA peripheral
              LED_control(255, 0, 0);
                                  // If the initialization fails, RED LED is switched
on at highest intensity i.e. 255
              while (1)
                                        // Unable to Initialize MMA
                     ;
```

```
	}

	test_mma();
                                        // Testing function to test MMA and I2C
	self_test();
                                  // Testing function to test accelerometer output
validity
	while (1) {
		read_xyz();
		convert_xyz_to_roll_pitch();
		float xaxis=fabs(roll);
		float yaxis=fabs(pitch);
		if(xaxis > 5 && yaxis < 5)
                        // Checking for angle with respect to X-axis if greater
than 5 Degrees
		{
                                                // and if angle with respect to Y-axis
is less than 5 Degrees
			LED_control(0,(int)PWM_STEP_SIZE*xaxis, 0);
			// Light green LED if xaxis > 5 degrees
			PRINTF("X: %f \t\n\r", xaxis);
                        // Printing the angle on the UART
		}
		else if (yaxis > 5 && xaxis <5)
                        // Checking for angle with respect to Y-axis if greater
than 5 Degrees
		{
                                                // and if angle with respect to X-axis
is less than 5 Degrees
			LED_control(0,0,(int)PWM_STEP_SIZE*yaxis);
			// Light blue LED if yaxis > 5 degrees
			PRINTF("Y: %f \t\n\r", yaxis);
                        // Printing the angle on the UART
		}
		else if ( xaxis > 5 && yaxis > 5)
                        // Checking for angle with respect to X-axis & Y-axis if
greater
		{
                                                // than 5 Degrees
			LED_control(0,(int)PWM_STEP_SIZE*xaxis,
(int)PWM_STEP_SIZE*yaxis);// Light green and blue LED if xaxis > 5 Degrees & yaxis >
5 degrees
			PRINTF("X: %f \t", fabs(roll));
                        // Printing the angle on the UART
			PRINTF("Y: %f\n\r", fabs(pitch));
		}
		else
		{
			LED_control(255, 255, 255);
                                  // Turn on White LED if both (x axis and y axis)
angles are less than 5 degrees
			PRINTF("X: %f \t", fabs(roll));
                        // Printing the angle on the UART
			PRINTF("Y: %f\n\r", fabs(pitch));
		}
```

```
            Delay(100);
                                        // Delay for legible output on UART
      }
}



/*
 * mma8451.c
 *
 *  Created on: Dec 3, 2020
 *
 *      Author: aditya.vny95
 *
 *
 */
#include <MKL25Z4.H>
#include "mma8451.h"
#include "i2c.h"
#include "delay.h"
#include <math.h>
#include <stdio.h>
#include "fsl_debug_console.h"


int16_t acc_X=0, acc_Y=0, acc_Z=0;
float roll=0.0, pitch=0.0;

//mma data ready
extern uint32_t DATA_READY;



//initializes mma8451 sensor
int init_mma()
{
      //set active mode, 14 bit samples and 800 Hz ODR
      i2c_write_byte(MMA_ADDR, REG_CTRL1, CTRL_REG1_ACTIVE);
      Delay(5);
                        // Delay for I2C to settle the value
      if(i2c_read_byte(MMA_ADDR, REG_CTRL1) == CTRL_REG1_ACTIVE)              //
Condition to check if the MMA was initialized correctly
      {
            PRINTF("MMA in Active State now\n\r");
            return 1;
      }
      else
            return 0;
}

void test_mma()
                        // Test function to check the MMA 'WHO AM I' register
value
```

```
{
                                // Verifies the identity of the MMA
        if(i2c_read_byte(MMA_ADDR, REG_WHOAMI) == WHOAMI)
        {
                PRINTF("I2C tested Successfully!!!\n\r");
                PRINTF("MMA Initialized Successfully!!!\n\r");
        }
}


void read_xyz()
{
        int i;
                                // Temporary variable for loop
        uint8_t data[6];
                        // variable to capture repeated read from I2C
        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
        acc_X = temp[0]/4;
        acc_Y = temp[1]/4;
        acc_Z = temp[2]/4;
}

void self_test()
                // Testing function to verify the output thrown by the accelerometer
{
        PRINTF("Testing Accelerometer Readings.........\n\r");
        int x_test[10], y_test[10], z_test[10];
        int x_sum=0, y_sum=0, z_sum=0;
        for(int i=0;i<10;i++)
                // Taking 10 samples to verify our output
        {
                self_test_mode(ON);
                // Switching on SELF test mode and storing values
                read_xyz();
                x_test[i]=acc_X;
                y_test[i]=acc_Y;
```

```
                z_test[i]=acc_Z;

                self_test_mode(OFF);
                // Switching off the SELF test mode and storing values
                read_xyz();
                x_test[i]-=acc_X;
                // Calculating the difference between the 2 modes
                y_test[i]-=acc_Y;
                z_test[i]-=acc_Z;

                x_sum+=x_test[i];
                // To calculate the average, finding their sum
                y_sum+=y_test[i];
                z_sum+=z_test[i];

        }
        if((x_sum/10 > x_ref_STmode) && (y_sum/10 > y_ref_STmode) && (z_sum/10 >
z_ref_STmode))                  // checking the average with the reference values
        {
                PRINTF("Accelerometer Readings Verified Successfully!!!\n\r");
        }
        else
        {
                PRINTF("Accelerometer Readings Verification FAILED\n\r");
        }
}

void self_test_mode(int mode)
{
        if(mode == 1)
        {
                i2c_write_byte(MMA_ADDR, REG_CTRL1, 0x00);          // Putting into
standby mode by clearing the active bit in REG_CTRL1
                Delay(5);
        // Delay given to give time for the I2c values to set
                i2c_write_byte(MMA_ADDR, REG_CTRL2, 0x80);          // Putting into
Self_test mode by setting the ST bit as 1 in REG_CTRL2
                Delay(5);
        // Delay given to give time for the I2c values to set
                i2c_write_byte(MMA_ADDR, REG_CTRL1, 0x01);          // Putting into
Active mode by setting the Active bit as 1 in REG_CTRL1
                Delay(5);
        }
        else if(mode == 0)
        {
                i2c_write_byte(MMA_ADDR, REG_CTRL1, 0x00);          // Putting into
standby mode by clearing the active bit in REG_CTRL1
                Delay(5);
        // Delay given to give time for the I2c values to set
                i2c_write_byte(MMA_ADDR, REG_CTRL2, 0x00);          // Getting out of
the Self test mode by clearing the ST bit in REG_CTRL2
                Delay(5);
        // Delay given to give time for the I2c values to set
                i2c_write_byte(MMA_ADDR, REG_CTRL1, 0x01);          // Going back to
the Active mode by setting the active bit to 1 in REG_CTRL1
```

```
			Delay(5);
		}

}

void convert_xyz_to_roll_pitch(void)
{
		float ax = acc_X/COUNTS_PER_G,                                          //
Calibrating value of the acceleration for all x,y & z axis
                          ay = acc_Y/COUNTS_PER_G,
                          az = acc_Z/COUNTS_PER_G;
		roll = atan2(ay, sqrt(ax*ax + az*az))*180/M_PI;            // Converting the
values to degrees through inverse tan function
		pitch = atan2(ax, sqrt(ay*ay + az*az))*180/M_PI;

}
```

```
/*
 * i2c.c
 *
 *  Created on: Dec 3, 2020
 *      Author: aditya.vny95
 */
#include <MKL25Z4.H>
#include "i2c.h"
int lock_detect=0;
int i2c_lock=0;

void i2c_init(void)
			// Initializing I2C
{
		//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);

		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

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

	// 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;
}


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

void i2c_read_setup(uint8_t dev, uint8_t address)
{
	I2C0->D = dev;
	//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|0x1);
	//send dev address (read)
	I2C_WAIT
	//wait for completion

	I2C_REC;
	//set to receive mode

}

//read a byte and ack/nack as appropriate
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
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;                                            //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|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

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;                                               //send dev
address
	I2C_WAIT                                                     //wait for
ack

	I2C0->D =  address;                                        //send write
address
	I2C_WAIT

	I2C0->D = data;                                              //send data
	I2C_WAIT
	I2C_M_STOP;

}




/*
 * delay.c
 *
 *  Created on: Dec 3, 2020
 *      Author: aditya.vny95
 */
#include <MKL25Z4.H>

void Delay (uint32_t dly)                                  // Delay function to give
a little pause between 2 commands, takes no of ticks as input
{
  volatile uint32_t t;

	for (t=dly*10000; t>0; t--)
		;
}

/*
 * GPIO_defs.h
 *
 *  Created on: Dec 3, 2020
 *      Author: aditya.vny95
```

```
 */

#ifndef GPIO_DEFS_H
#define GPIO_DEFS_H

// basic light switch
#define LED1_POS (1)      // on port A
#define LED2_POS (2)      // on port A
#define SW1_POS (5)       // on port A

#define MASK(x) (1UL << (x))

// Speaker output
#define SPKR_POS (0)      // on port C

#endif


/*
 * i2c.h
 *
 *  Created on: Dec 3, 2020
 *      Author: aditya.vny95
 */

#include <stdint.h>

#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 BUSY_ACK        while(I2C0->S & 0x01)
#define TRANS_COMP        while(!(I2C0->S & 0x80))
#define I2C_WAIT                i2c_wait();

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

void i2c_init(void);
/*
 *  Initializing I2C
 * Arguments :            None
 * Return type      :            None
 * Return           :            None
 */

void i2c_start(void);
/*
 * send start sequence
 * Arguments :            None
 * Return type      :            None
 * Return           :            None
```

```
 */
void i2c_read_setup(uint8_t dev, uint8_t address);
/*
 * Send device and register addresses
 * Arguments :            Dev and address
 * Return type      :            None
 * Return           :            None
 */

uint8_t i2c_repeated_read(uint8_t);
/*
 * read a byte and ack/nack as appropriate
 * Arguments :            Integer
 * Return type      :            Integer
 * Return           :            Data
 */

uint8_t i2c_read_byte(uint8_t dev, uint8_t address);
/*
 * using 7bit addressing reads a byte from dev:address
 * Arguments :            Dev, Address
 * Return type      :            Integer
 * Return           :            Data
 */

void i2c_write_byte(uint8_t dev, uint8_t address, uint8_t data);
/*
 * using 7bit addressing writes a byte data to dev:address
 * Arguments :            Dev, Address, Data
 * Return type      :            None
 * Return           :            None
 */


/*
 * LEDs.h
 *
 *  Created on: Dec 3, 2020
 *      Author: aditya.vny95
 */

#ifndef LEDS_H
#define LEDS_H

// Freedom KL25Z LEDs
#define RED_LED_POS (18)                                // on port B
#define GREEN_LED_POS (19)                              // on port B
#define BLUE_LED_POS (1)                                // on port D
#define PWM_PERIOD (48000)                              // PWM period 48000
#define LED_COLOR_STEP (187.5)                          // Calculated step size by
48000/256 = 187.5

// function prototypes
void Init_RGB_LEDs(void);
/*
```

```
 * This function initiates the LEDs and GPIOs for output
 * Arguments :            Integer
 * Return type      :            None
 * Return           :            None
 */

void LED_control(int r,int g,int b);
/*
 * To set the PWM output for all the three LEDs.
 * Arguments :            Integer, values ranging between 0-255
 * Return type      :            None
 * Return           :            None
 */

#endif


/*
 * mma8451.h
 *
 *  Created on: Dec 3, 2020
 *      Author: aditya.vny95
 */

#ifndef MMA8451_H
#define MMA8451_H
#include <stdint.h>
#include "fsl_debug_console.h"

#define MMA_ADDR 0x3A

#define REG_XHI 0x01
#define REG_XLO 0x02
#define REG_YHI 0x03
#define REG_YLO 0x04
#define REG_ZHI     0x05
#define REG_ZLO 0x06

#define REG_WHOAMI 0x0D
#define REG_CTRL1  0x2A
#define REG_CTRL2  0x2B
#define REG_CTRL4  0x2D

#define CTRL_REG1_ACTIVE 0x01
#define WHOAMI 0x1A

#define COUNTS_PER_G (4096.0)
#define M_PI (3.14159265)
#define x_ref_STmode (181)                          // Reference Value of x axis
when used in Self test mode
#define y_ref_STmode (255)                          // Reference Value of y axis
when used in Self test mode
#define z_ref_STmode (1680)                               // Reference Value of z
axis when used in Self test mode
```

```
#define ON (1)                                                  // This is the
variable to switch ON the self test mode
#define OFF (0)                                                 // This is the
variable to switch OFF the self test mode

int init_mma(void);
/*
 * Initializes mma8451 sensor
 * Arguments :            Dev, Address
 * Return type      :            Integer
 * Return           :            1 if initialized correctly, 0 if incorrect
initialization
 */

void read_xyz(void);
/*
 * Used to read the values from the MMA accelerometer and stores them in acc_x, acc_y
& acc_z variables
 * Arguments :            NONE
 * Return type      :            NONE
 * Return           :            NONE
 */

void convert_xyz_to_roll_pitch(void);
/*
 * This function is used to convert accelerometer x,y,z values to roll and pitch
 * Arguments :            NONE
 * Return type      :            NONE
 * Return           :            NONE
 */

void test_mma(void);
/*
 * Test function to check the MMA 'WHO AM I' register value
 * Arguments :            NONE
 * Return type      :            NONE
 * Return           :            NONE
 */

void self_test(void);
/*
 * Function to verify the accelerometer readings
 * Arguments :            NONE
 * Return type      :            NONE
 * Return           :            NONE
 */

void self_test_mode(int mode);
/*
 * Function to switch on and off the self test mode of the MMA,
 * Arguments :            takes ON or OFF as arguments
 * Return type      :            NONE
 * Return           :            NONE
 */
```

```
extern float roll, pitch;                           // Roll is the angle made about
the x axis,Pitch is the angle made about the y axis
extern int16_t acc_X, acc_Y, acc_Z;

#endif
```