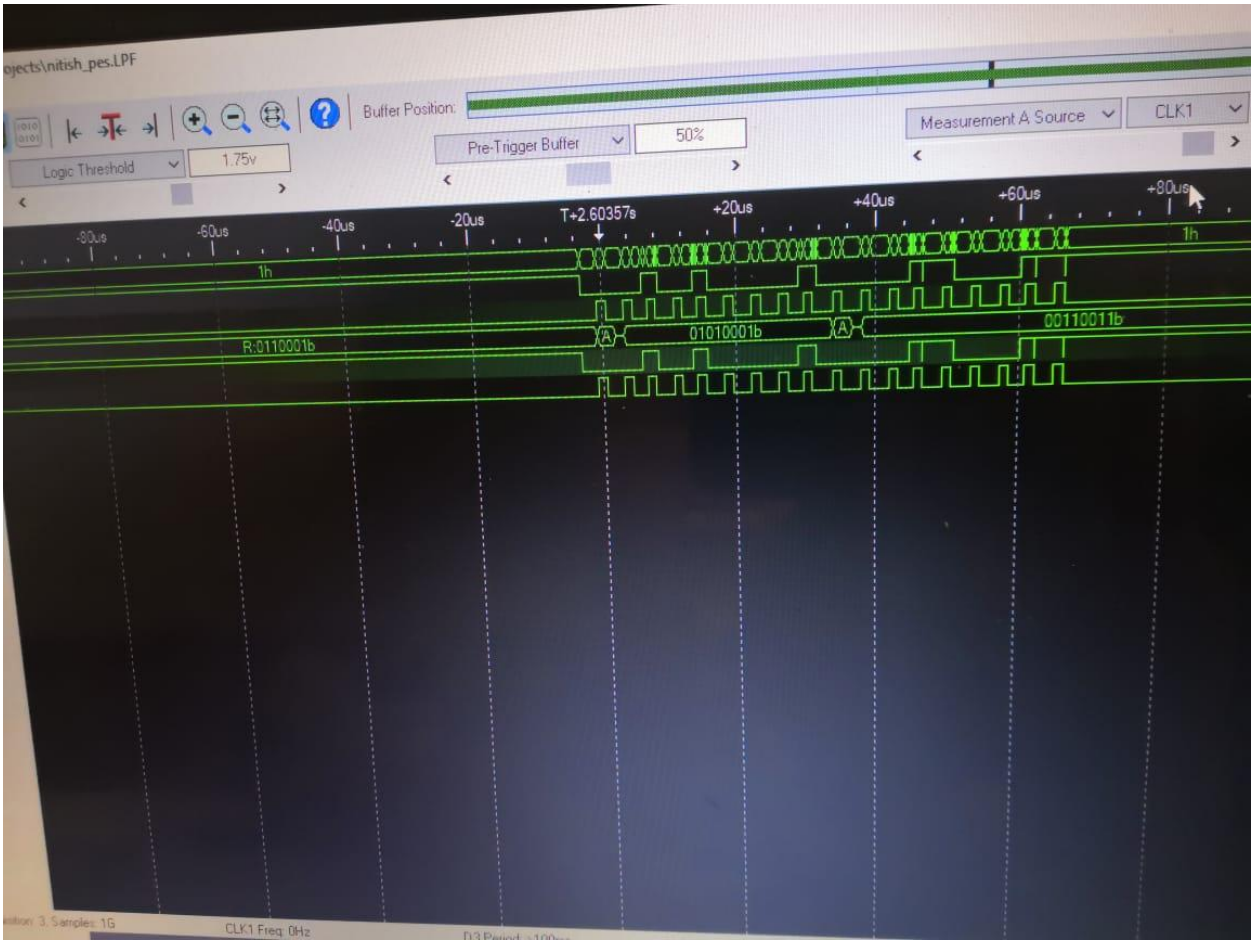
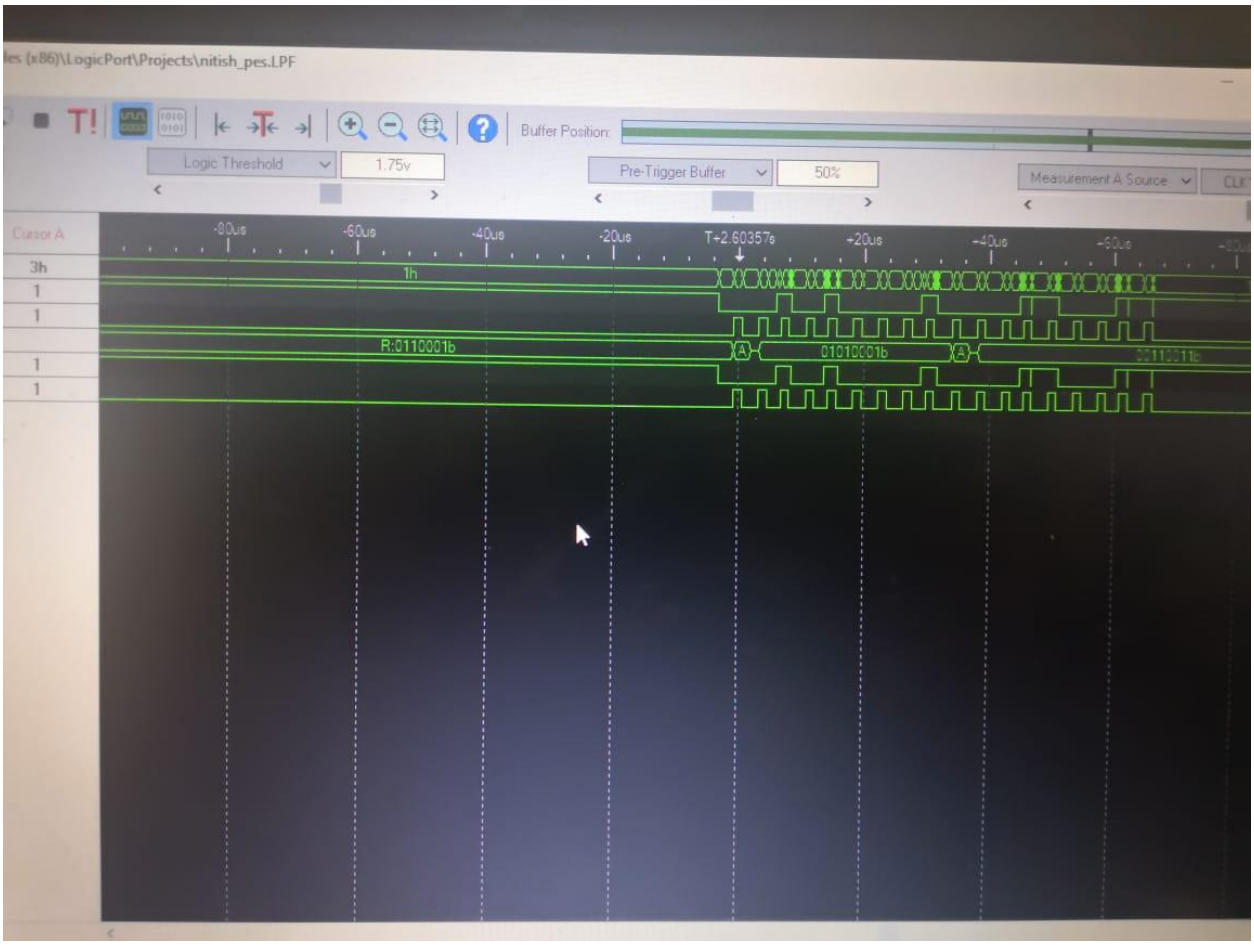
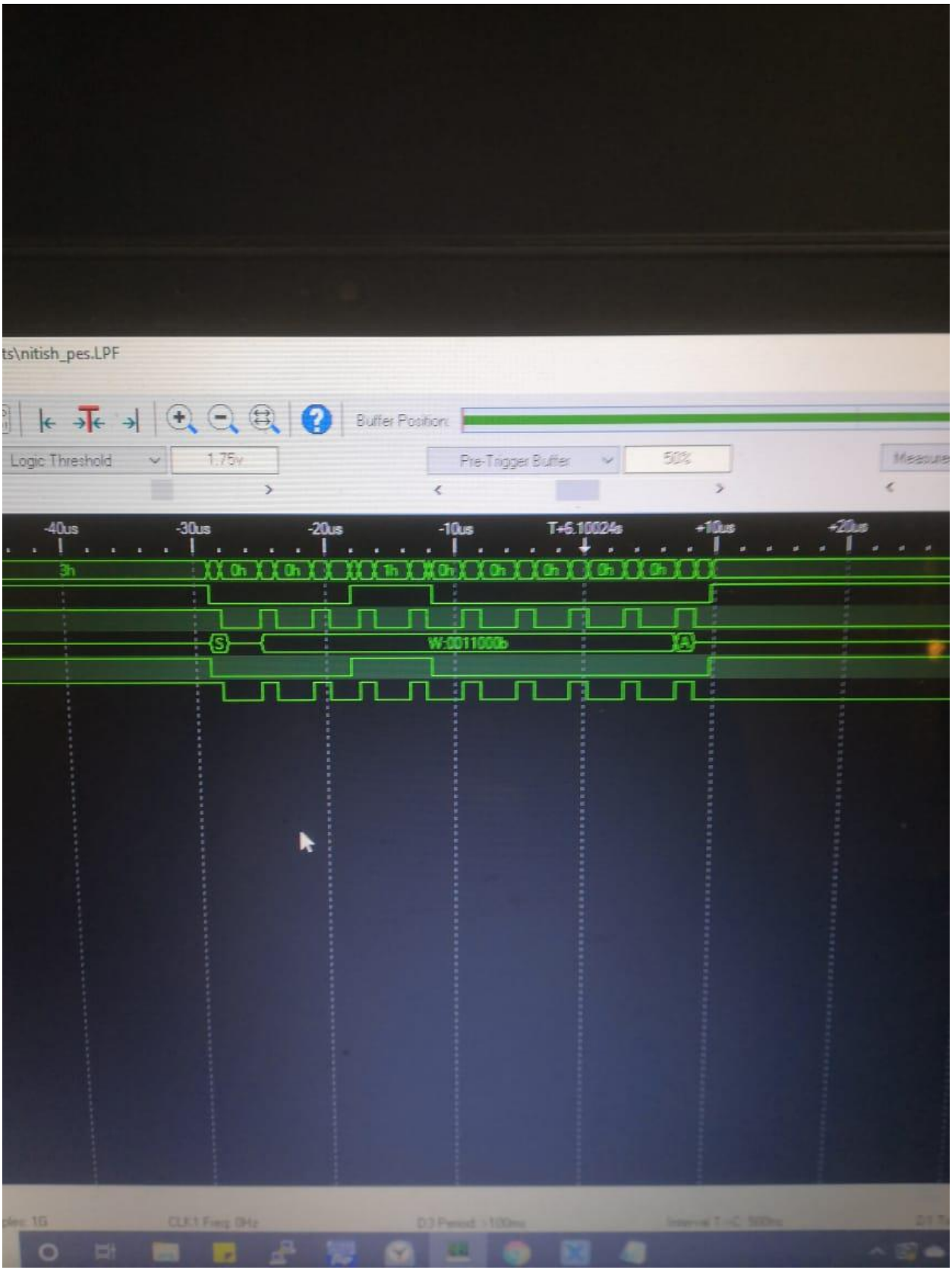
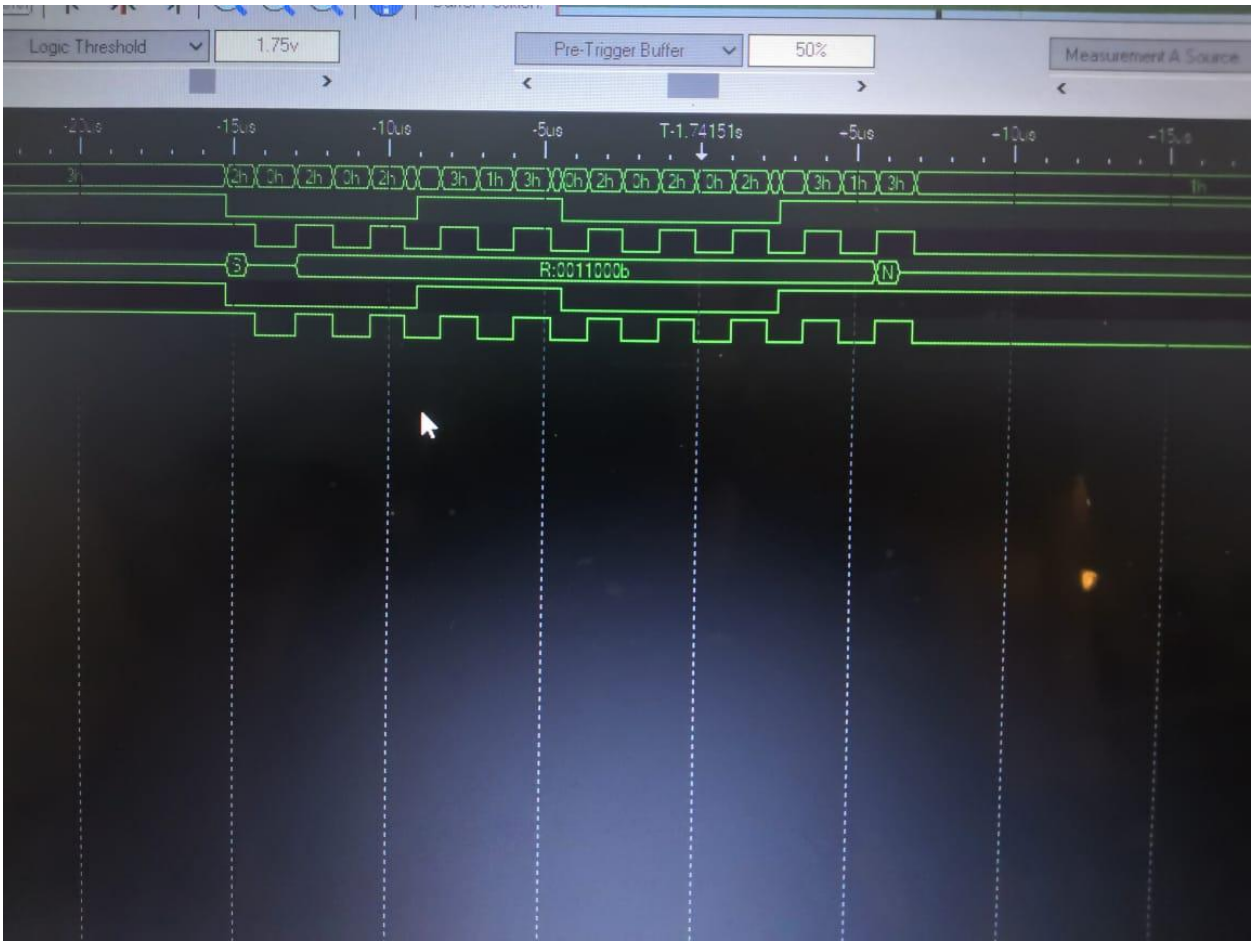
Logic Analyzer Images




(x86)\LogicPort\Projects\nitish_pes.LPF
Buffer Position:
Logic Threshold
1.75v
Pre-Trigger Buffer
50%
Measurement A Source
Cursor A
-80us
-60us
-40us
-20us
T+2.60357s
+20us
+40us
+60us
3h
1
1
1
1
1h
R:01100001b
A
01010001b
A


ts\nitish_pes.LPF
Buffer Position:
Logic Threshold
1.75v
Pre-Trigger Buffer
50%
-40us
-30us
-20us
-10us
T+6.10024s
+10us
+20us
3h
0h
1h
S
W:0011000b
A



These images show captured I2C communication between LIS331h and the FRDM KL25z.

```
/*----------------------------------------------------------------------------
*----------------------------------------------------------------------------*/
#include <MKL25Z4.H>
#include <stdio.h>
#include <math.h>
#include "gpiodefs.h"
#include "led.h"
#include "i2c.h"
#include "mma8541.h"
#include "delay.h"
#include "statemachine1.h"
#include "selftest.h"
#include "typedef.h"
#include "logger.h"
//#define ENABLE_LOGGER 1

uint8_t state=0;
u8 byte_counter = 0;
u8 rx_buffer[BUFFER_SIZE];
/*----------------------------------------------------------------------------
MAIN function
*----------------------------------------------------------------------------*/
//int main (void) {
//
// Init_RGB_LEDs();
// i2c_init(); /* init i2c */
// if (!init_mma()) { /* init mma peripheral */
// Control_RGB_LEDs(1, 0, 0);/* Light red error LED */
// while (1) /* not able to initialize mma */
```

```
// ;
// }
//
// Control_RGB_LEDs(1, 1, 0);
//
// Delay(100);
//
// while (1) {
// read_full_xyz();
// DisplayXYZ();
// convert_xyz_to_roll_pitch();
// // Light green LED if pitch > 10 degrees
// // Light blue LED if roll > 10 degrees
// Control_RGB_LEDs(0, (fabs(roll) > 10)? 1:0, (fabs(pitch) > 10)? 1:0);
// }
//}


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

#ifdef ENABLE_LOGGER
Log_enable();
#else
```

```
Log_disable();
#endif
int status = Log_status();
Log_string("Logger Status is", NO_NEWLINE);
Log_integer(status);

Init_RGB_LEDs();
i2c_init();
if (!init_mma()){
Control_RGB_LEDs(1, 0, 0);
return 0;
}
Delay(500);
int ST=self_test();

if(1==ST)
{
Log_string("Self Test Fail", NEWLINE);
Control_RGB_LEDs(1, 0, 0);
return 1;
}
Log_string("Self Test Pass", NEWLINE);
Control_RGB_LEDs(0, 1, 0);
Delay(100);
Control_RGB_LEDs(0, 0, 0);
while(1)
{
//if(0==state)
{
Statemachine1();
}
```

```
if(5==state)
{
printf("\nEND");
return 0;
}

}
//printf("\nNot in loop ");
return 0;
}


//void Enable_irq (int irq)
//{
// /* Make sure that the IRQ is an allowable number. Up to 32 is
// * used.
// *
// * NOTE: If you are using the interrupt definitions from the header
// * file, you MUST SUBTRACT 16!!!
// */
//
// /* Set the ICPR and ISER registers accordingly */
// NVIC->ICPR[0] |= 1 << (irq%32);
// NVIC->ISER[0] |= 1 << (irq%32);
//}
//
//
//
//void I2C_Master_Init(void)
//{
// /* Enable clock for I2C0 module */
```

```
// SIM->SCGC4 |= SIM_SCGC4_I2C0_MASK;
//
// /* Enable clock for Port E */
// SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;
//
//
// /* Port E MUX configuration */
// PORTE->PCR[24] |= PORT_PCR_MUX(5);
// PORTE->PCR[25] |= PORT_PCR_MUX(5);
//
//
// /* Configure Divider Register */
// I2C0->F |= (I2C_F_MULT(2) | I2C_F_ICR(22));
//
// /* Enable I2C module and interrupt */
// I2C0->C1 |= I2C_C1_IICEN_MASK | I2C_C1_IICIE_MASK;
//
//
// /* Enable TX mode */
// I2C0->C1 |= I2C_C1_TX_MASK;
//
// /* Enable I2C0 NVIC interrupt */
// Enable_irq(8);
// printf("\nMaster init is done");
//}
//
//
//
//
//void I2C_Master_Transmit(void)
```

```
//{
// I2C0->C1 |= I2C_C1_MST_MASK; // Generate START SIGNAL
// I2C0->D = (0x3B);// Write 7-bit Slave Address + READ bit
// printf("\nMaster transmit is done");
//}
//
//
//
//
//void I2C0_IRQHandler(void)
//{
// u8 status = 0x00;
// u8 dummy_var;
//
// status = I2C0->S; // Read status
//
// I2C0->S |= I2C_S_IICIF_MASK; // Clear interrupt flag
//
// if(I2C0->C1 & I2C_C1_TX_MASK) // Transmitter mode?
// {
// if((status & I2C_S_RXAK_MASK) == 0) // ACK Received?
// {
// I2C0->C1 &= ~I2C_C1_TX_MASK; // Change to receiver mode
// dummy_var = I2C0->D; // Dummy read to start reception
// }
// else
// {
// I2C0->C1 &= ~I2C_C1_MST_MASK; // Generate STOP signal
// }
// }
```

```
// else
// {
// byte_counter++; // Increment the count of bytes received
//
// if(byte_counter == BUFFER_SIZE) // Last byte to be received?
// {
// I2C0->C1 &= ~I2C_C1_MST_MASK; // Generate STOP signal
// }
// else
// {
// if(byte_counter == (BUFFER_SIZE - 1)) // Only 1 more byte pending to read?
// {
// I2C0->C1 |= I2C_C1_TXAK_MASK; // Generate NACK in the next reception
// }
// }
//
// rx_buffer[byte_counter-1] = I2C0->D;// Copy data register to buffer
// //rx_buffer[0]='c';
// }
//}
//
//

//int main(void)
//{
// I2C_Master_Init(); // Initialize I2C module in master mode
//
// I2C_Master_Transmit(); // Start transmission
// printf("\nDone");
// I2C0_IRQHandler();
// for(int i=0;i<=BUFFER_SIZE;i++){printf("\n\rvalue is %d",rx_buffer[i]);}
```

```
// //for(;;){}
//
// return 0;
//}


#include <MKL25Z4.H>

void Delay (uint32_t dly) {
volatile uint32_t t;

for (t=dly*10000; t>0; t--)
;
}
#ifndef DELAY_H
#define DELAY_H
extern void Delay(uint32_t dlyTicks);

#endif


#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; /*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
// #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; /*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
#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; /*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;

}
void Enable_irq (int irq)
{
/* Make sure that the IRQ is an allowable number. Up to 32 is
* used.
*
* NOTE: If you are using the interrupt definitions from the header
```

```
* file, you MUST SUBTRACT 16!!!
*/

/* Set the ICPR and ISER registers accordingly */
NVIC->ICPR[0] |= 1 << (irq%32);
NVIC->ISER[0] |= 1 << (irq%32);
}

void I2C_Master_Init(void)
{
/* Enable clock for I2C0 module */
SIM->SCGC4 |= SIM_SCGC4_I2C0_MASK;

/* Enable clock for Port E */
SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

/* Port E MUX configuration */
PORTE->PCR[24] |= PORT_PCR_MUX(5);
PORTE->PCR[25] |= PORT_PCR_MUX(5);

/* Configure Divider Register */
I2C0->F |= (I2C_F_MULT(2) | I2C_F_ICR(22));

/* Enable I2C module and interrupt */
I2C0->C1 |= I2C_C1_IICEN_MASK | I2C_C1_IICIE_MASK;

/* Enable TX mode */
I2C0->C1 |= I2C_C1_TX_MASK;

/* Enable I2C0 NVIC interrupt */
Enable_irq(8);
printf("\nMaster init is done");
}
```

```
void I2C_Master_Transmit(void)
{
I2C0->C1 |= I2C_C1_MST_MASK; // Generate START SIGNAL
I2C0->D = ((0x3A)<<1|0x01);// Write 7-bit Slave Address + READ bit
printf("\nMaster transmit is done");
}


void I2C0_IRQHandler(void)
{
u8 status = 0x00;
u8 dummy_var;

status = I2C0->S; // Read status

I2C0->S |= I2C_S_IICIF_MASK; // Clear interrupt flag

if(I2C0->C1 & I2C_C1_TX_MASK) // Transmitter mode?
{
if((status & I2C_S_RXAK_MASK) == 0) // ACK Received?
{
I2C0->C1 &= ~I2C_C1_TX_MASK; // Change to receiver mode
dummy_var = I2C0->D; // Dummy read to start reception
}
else
{
I2C0->C1 &= ~I2C_C1_MST_MASK; // Generate STOP signal
}
}
else
{
byte_counter++; // Increment the count of bytes received
```

```
if(byte_counter == BUFFER_SIZE) // Last byte to be received?
{
I2C0->C1 &= ~I2C_C1_MST_MASK; // Generate STOP signal
}
else
{
if(byte_counter == (BUFFER_SIZE - 1)) // Only 1 more byte pending to read?
{
I2C0->C1 |= I2C_C1_TXAK_MASK; // Generate NACK in the next reception
}
}

rx_buffer[byte_counter-1] = I2C0->D;// Copy data register to buffer
//rx_buffer[0]='c';
}
}

#include <stdint.h>
#include "typedef.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);
```

```
void i2c_start(void);
void i2c_read_setup(uint8_t dev, uint8_t address);
uint8_t i2c_repeated_read(uint8_t);

uint8_t i2c_read_byte(uint8_t dev, uint8_t address);
void i2c_write_byte(uint8_t dev, uint8_t address, uint8_t data);

void Enable_irq (int irq);
void I2C_Master_Init(void);
void I2C0_IRQHandler(void);
extern u8 byte_counter;
extern u8 rx_buffer[];
#define BUFFER_SIZE 10


#include <MKL25Z4.H>
#include "led.h"
#include "gpiodefs.h"

void Init_RGB_LEDs(void) {
// Enable clock to ports B and D
SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;;

// Make 3 pins GPIO
PORTB->PCR[RED_LED_POS] &= ~PORT_PCR_MUX_MASK;
PORTB->PCR[RED_LED_POS] |= PORT_PCR_MUX(1);
PORTB->PCR[GREEN_LED_POS] &= ~PORT_PCR_MUX_MASK;
PORTB->PCR[GREEN_LED_POS] |= PORT_PCR_MUX(1);
PORTD->PCR[BLUE_LED_POS] &= ~PORT_PCR_MUX_MASK;
PORTD->PCR[BLUE_LED_POS] |= PORT_PCR_MUX(1);

// Set ports to outputs
PTB->PDDR |= MASK(RED_LED_POS) | MASK(GREEN_LED_POS);
PTD->PDDR |= MASK(BLUE_LED_POS);
```

```
}

void Control_RGB_LEDs(unsigned int red_on, unsigned int green_on, unsigned int blue_on) {
if (red_on) {
PTB->PCOR = MASK(RED_LED_POS);
} else {
PTB->PSOR = MASK(RED_LED_POS);
}
if (green_on) {
PTB->PCOR = MASK(GREEN_LED_POS);
} else {
PTB->PSOR = MASK(GREEN_LED_POS);
}
if (blue_on) {
PTD->PCOR = MASK(BLUE_LED_POS);
} else {
PTD->PSOR = MASK(BLUE_LED_POS);
}
}

#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

// function prototypes
void Init_RGB_LEDs(void);
void Control_RGB_LEDs(unsigned int red_on, unsigned int green_on, unsigned int blue_on);
void Toggle_RGB_LEDs(unsigned int red, unsigned int green, unsigned int blue);

#endif
```

```
/*
* logger.c
*
* Created on: Feb 21, 2020
* Author: Kristina Brunsgaard
*/
#include "logger.h"

int enable;

void Log_enable(){ //begin printing log messages when called
printf("\nLog Messages Enabled");
enable = 1;
}
void Log_disable() { //ignore any log messages until re-enabled
printf("\nLog Messages Disabled");
enable = 0;
}

int Log_status(){ //returns a flag to indicate whether the logger is enabled or disabled
return enable;
}

void Log_data(uint8_t *bytes, int length){ //display in hexadecimal an address and contents of a memory location, arguments are a pointer to a sequence of bytes and a specified length (in dword)
int i;
int space = 0;
if (enable){
if (length == 0) {
printf("Log Error: 0 Bytes Entered\n");
} else {
printf("\nLOG: address: %08X\n memory: ", bytes);
```

```
for (i = 0; i < length; i++) {
if (space == 4) {
printf(" ");
space = 0;
}
printf("%02X", bytes[i]);
space++;
}
printf("\n");
}
}
}

void Log_string(char *string, int newLine){ //display a string
if (enable && newLine) {
printf("\nLOG: %s\n", string);
} else if ((enable == 1) && (newLine == 0)) {
printf("\nLOG: %s ", string);
}
}

void Log_integer(int integer) { //display an integer
if (enable) {
printf("%d\n", integer);
}
}

void Log_pointer(uint8_t *pointer) { //display an integer
if (enable) {
printf("%08X\n", pointer);
}
}
```

```
/*
* logger.h
*
* Created on: Feb 21, 2020
* Author: Kristina Brunsgaard
*/

#ifndef LOGGER_H_
#define LOGGER_H_

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

extern int enable;

typedef enum newline
{ NO_NEWLINE = 0, //log_string print newline
NEWLINE // log_string don't print newline
} newline;

void Log_enable();
void Log_disable();
int Log_status();
void Log_data(uint8_t *bytes, int length);
void Log_string(char *string, int newLine);
void Log_integer(int integer);
void Log_pointer(uint8_t *pointer);

#endif /* LOGGER_H_ */

#include <MKL25Z4.H>
#include "mma8541.h"
#include "i2c.h"
```

```
#include "delay.h"
#include <math.h>
#include<stdio.h>
int16_t acc_X=0, acc_Y=0, acc_Z=0;
int16_t te_X,te_Y,te_Z=0;
int16_t avg_X,avg_Y,avg_Z;
static uint16_t ctr=1;
float roll=0.0, pitch=0.0;

//mma data ready
extern uint32_t DATA_READY;


//initializes mma8451 sensor
//i2c has to already be enabled
int init_mma()
{
//set active mode, 14 bit samples and 800 Hz ODR
i2c_write_byte(MMA_ADDR, REG_CTRL1, 0x01);
return 1;
}

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
acc_X = temp[0]/4;
acc_Y = temp[1]/4;
acc_Z = temp[2]/4;


//printf("\n\r%d\n",acc_X);
}

void avg_xyz(void)
{

{
te_X=te_X+acc_X;
te_Y=te_Y+acc_Y;
te_Z=te_Z+acc_Z;
//ctr++;
avg_X=(te_X)/ctr;
avg_Y=(te_Y)/ctr;
avg_Z=(te_Z)/ctr;
}
ctr++;
printf("\nAverage is %d",avg_X);
printf("\nAverage is %d",avg_Y);
printf("\nAverage is %d",avg_Z);
```

```
}

void DisplayXYZ(void)
{
printf("\nXcoordinate is %d",acc_X);
printf("\nYcoordinate is %d",acc_Y);
printf("\nZcoordinate is %d",acc_Z);
printf("\n");
}

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 = (int8_t) i2c_read_byte(MMA_ADDR, REG_XHI);
Delay(100);
acc_Y = (int8_t) i2c_read_byte(MMA_ADDR, REG_YHI);
Delay(100);
acc_Z = (int8_t) i2c_read_byte(MMA_ADDR, REG_ZHI);

}

void convert_xyz_to_roll_pitch(void) {
float ax = acc_X/COUNTS_PER_G,
ay = acc_Y/COUNTS_PER_G,
az = acc_Z/COUNTS_PER_G;

roll = atan2(ay, az)*180/M_PI;
pitch = atan2(ax, sqrt(ay*ay + az*az))*180/M_PI;

}
void read_full_xyz1()
{
//I2C_Master_Init(); // Initialize I2C module in master mode
```

```
//I2C_Master_Transmit(); // Start transmission
//printf("\nDone");
I2C0_IRQHandler();
//for(int i=0;i<=BUFFER_SIZE;i++){printf("\n\rvalue is %d",rx_buffer[i]);}
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++) {
rx_buffer[i] = i2c_repeated_read(0);
}
// Read last byte ending repeated mode
rx_buffer[i] = i2c_repeated_read(1);

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

// Align for 14 bits
acc_X = temp[0]/4;
acc_Y = temp[1]/4;
acc_Z = temp[2]/4;


//printf("\n\r%d\n",acc_X);
}

//mma data ready irq
// void PORTA_IRQHandler()
// {
```

```
// NVIC_ClearPendingIRQ(PORTA_IRQn);
// DATA_READY = 1;
// }
#ifndef MMA8451_H
#define MMA8451_H
#include <stdint.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_CNRL2 0x2B
#define REG_CNRL3 0x2C
#define REG_CTRL4 0x2D

#define WHOAMI 0x1A

#define COUNTS_PER_G (4096.0)
#define M_PI (3.14159265)

int init_mma(void);
void read_full_xyz(void);
void read_full_xyz1();
void DisplayXYZ(void);
void read_xyz(void);
void convert_xyz_to_roll_pitch(void);
void avg_xyz(void);
```

```
void PORTA_IRQHandler();

extern float roll, pitch;
extern int16_t acc_X, acc_Y, acc_Z;
extern int16_t te_X,te_Y,te_Z;
extern int16_t avg_X,avg_Y,avg_Z;
#endif

/*
* 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 mtb.c

* @brief MTB initialization file.

* @details Symbols controlling behavior of this code...

* __MTB_DISABLE

* If this symbol is defined, then the buffer array for the MTB

* will not be created.

*

* __MTB_BUFFER_SIZE

* Symbol specifying the sizer of the buffer array for the MTB.

* This must be a power of 2 in size, and fit into the available

* RAM. The MTB buffer will also be aligned to its 'size'

* boundary and be placed at the start of a RAM bank (which

* should ensure minimal or zero padding due to alignment).

*

* __MTB_RAM_BANK

* Allows MTB Buffer to be placed into specific RAM bank. When

* this is not defined, the "default" (first if there are

```
* several) RAM bank is used.
*/

/* This is a template for board specific configuration created by MCUXpresso IDE Project Wizard.*/

// Allow MTB to be removed by setting a define (via command line)
#if !defined (__MTB_DISABLE)

// Allow for MTB buffer size being set by define set via command line
// Otherwise provide small default buffer
#if !defined (__MTB_BUFFER_SIZE)
#define __MTB_BUFFER_SIZE 128
#endif

// Check that buffer size requested is >0 bytes in size
#if (__MTB_BUFFER_SIZE > 0)
// Pull in MTB related macros
#include <cr_mtb_buffer.h>

// Check if MYTB buffer is to be placed in specific RAM bank
#if defined(__MTB_RAM_BANK)
// Place MTB buffer into explicit bank of RAM
__CR_MTB_BUFFER_EXT(__MTB_BUFFER_SIZE,__MTB_RAM_BANK);
#else
// Place MTB buffer into 'default' bank of RAM
__CR_MTB_BUFFER(__MTB_BUFFER_SIZE);

#endif // defined(__MTB_RAM_BANK)

#endif // (__MTB_BUFFER_SIZE > 0)

#endif // !defined (__MTB_DISABLE)

/*
* selftest.c
*
```

```
* Created on: Mar 24, 2020
* Author: nitis
*/

#include <stdint.h>
#include "mma8541.h"
#include "i2c.h"
#include <stdio.h>

int self_test(void)
{
uint8_t cntrl_reg = 0x2B; //control register2
uint8_t data = 0x80; //normal mode, 400Hz, xyz-enabled, self test value 0b1000 0000
int16_t pre_x, pre_y, pre_z;
int16_t post_x, post_y, post_z;
int16_t diff_x, diff_y, diff_z;
int8_t flag_x, flag_y, flag_z;
//accel_read();
read_full_xyz();
pre_x = acc_X;
pre_y = acc_Y;
pre_z = acc_Z;
//i2c_write_byte(dev_id, cntrl_reg, data); //Full scale, Self-test enabled, self-test plus mode
i2c_write_byte(MMA_ADDR, cntrl_reg, data);
//accel_read();
read_full_xyz();
post_x = acc_X;
post_y = acc_Y;
post_z = acc_Z;
diff_x = post_x - pre_x;
if(diff_x<0){diff_x=-diff_x;}
```

```
diff_y = -(post_y - pre_y);
if(diff_y<0){diff_y=-diff_y;}
diff_z = post_z - pre_z;
if(diff_z<0){diff_z=-diff_z;}
printf("\nDiff_x is %d ",diff_x);
printf("\nDiff_y is %d ",diff_y);
printf("\nDiff_z is %d ",diff_z);
if((diff_x <= 200))// && (diff_x <= 180))
{
flag_x = 1;
printf("Flag x is set");
}
if((diff_y <= 300))// && (diff_y <= 180))
{
flag_y = 1;
printf("Flag y is set");
}
if((diff_z >=0))// && (diff_y <= 370))
{
flag_z = 1;
printf("Flag z is set");
}
if((flag_x & flag_y & flag_z) == 1)
{
data=0x00;
i2c_write_byte(MMA_ADDR, cntrl_reg, data);
printf("\nSelf Test pass!");
return 0;
}
else
```

```
{
data=0x00;
i2c_write_byte(MMA_ADDR, cntrl_reg, data);
printf("\nSelf Test failed!");
return 1;
}
}
/*
* selftest.h
*
* Created on: Mar 24, 2020
* Author: nitis
*/

#ifndef SELFTEST_H_
#define SELFTEST_H_
int self_test(void);

#endif /* SELFTEST_H_ */
/*
* Statemachine1
*/
#include <stdio.h>
#include <stdint.h>
#include "i2c.h"
#include "mma8541.h"
#include "touchSen.h"
#include "statemachine2.h"
#include "statemachine1.h"
#include "timer.h"
#include "delay.h"
```

```
#include "logger.h"
#include "uCUnit-v1.0.h"

uint8_t n=0;

int currentevent=ePass;

void Statemachine1(void)
{
switch (state)
{
case sReadXYZ:
{
UCUNIT_CheckIsEqual(0,currentevent);
if(currentevent==ePass || currentevent==eTimeout)
{
read_full_xyz();
if(acc_X!=0)
{
currentevent=eComplete;
}
else
{
currentevent=eDisconnnected;
}
}

else
{
printf("\nTerminated");
}
//if(readDone!=0){state=sDisconnect;break;}
```

```
if(currentevent==eComplete)
{
state=sDisplay;
}
else if(currentevent==eDisconnnected)
{
state=sDisconnect;
}
break;

}
case sDisplay :
{
UCUNIT_CheckIsEqual(3,currentevent);
if(currentevent==eComplete)
{
DisplayXYZ();
printf("\n-------");
avg_xyz();
printf("\n-------");
state=sPollSlider;
currentevent=eComplete;
}

break;
}
case sPollSlider :
{

UCUNIT_CheckIsEqual(3,currentevent);
if(currentevent==eComplete)
{
```

```
Touch_Init();
Init_SysTick();
while(state==sPollSlider)
{

uint16_t touch_val=0;
touch_val=Touch_Scan_LH1();
printf("\nTouch_val has value is %d",touch_val);
Log_string("TouchSensor has value ", NEWLINE);
Log_integer(touch_val);
if (touch_val>=650 && touch_val<750)
{
currentevent=eLeftSlider;state=nextStatemachine;break;
}
else if(touch_val>800)
{
currentevent=eRightSlider;state=sEnd;break;
}

}
}
break;
}
case sDisconnect :
{
UCUNIT_CheckIsEqual(2,currentevent);
if(currentevent==eDisconnnected)
{
```

```
printf("\nDisconnected !");
state=sEnd;
currentevent=eFinish;
break;
}
}
case sEnd :
{
if(currentevent==eRightSlider||currentevent==eFinish)
{
state=5;
break;
}
}
case nextStatemachine :
{
UCUNIT_CheckIsEqual(4,currentevent);
UCUNIT_CheckIsEqual(6,currentevent);
if(currentevent==eLeftSlider || currentevent==eTimeout)
{
//statemachine2();
//printf("\nIn Statemachine2");
statemachine2();
break;
}
}
default:{printf("\nIn default state press reset");state=5;break;}
}
}
/*
```

```
* staemachine1.h
*
* Created on: Mar 19, 2020
* Author: nitis
*/

#ifndef STATEMACHINE1_H_
#define STATEMACHINE1_H_
extern uint8_t state;
//extern uint8_t readDone;

enum states{
sReadXYZ,
sDisplay,
sPollSlider,
sDisconnect,
nextStatemachine,
sEnd

};

enum events{
ePass,
eFail,
eDisconnnected,
eComplete,
eLeftSlider,
eRightSlider,
eTimeout,
eFinish
};

extern int currentevent;
```

```
void Statemachine1(void);

#endif /* STATEMACHINE1_H_ */

/*
* Statemachine2 is table driven.
*/
#include "i2c.h"
#include "mma8541.h"
#include "touchSen.h"
#include <stdio.h>
#include "statemachine1.h"
#include "led.h"
static int Current_state=0;
static int Current_event=0;

typedef enum states2{
sReadXYZ2,
sDisplay2,
sPollSlider2,
sDisconnect2,
nextStatemachine2,
sEnd2

}systemstate;

typedef enum events2{
ePass2,
eFail2,
eDisconnnected2,
eComplete2,
eLeftSlider2,
eRightSlider2,
```

```
eTimeout2,
eFinish2
};

void PollSliderHandler(void)
{
Touch_Init();
Init_SysTick();

while(state==sPollSlider)
{


uint16_t touch_val=0;

touch_val=Touch_Scan_LH1();
printf("\nTouch_val has value is %d",touch_val);
if (touch_val>=800 && touch_val<1000)
{
currentevent=eLeftSlider;state=nextStatemachine;Statemachine1();

}
else if(touch_val>1000)
{
currentevent=eRightSlider;state=sEnd;Statemachine1();
}


}
}
typedef void(*functionPointerType)(void);

struct commandStruct{
int *name;
functionPointerType execute;
```

```
};

void DisconectHandler(void)
{
state=3;
Statemachine1();
}

void NextSM(void)
{
state=4;
Statemachine1();
}

void EndHandler(void)
{
state=5;
Statemachine1();
}

const struct commandStruct commands[6]= {
{sReadXYZ2, &read_full_xyz},
{sDisplay2, &DisplayXYZ},
{sPollSlider2, &PollSliderHandler},
{sDisconnect2, &DisconectHandler},
{nextStatemachine2, &NextSM},
{sEnd2, &EndHandler},
// {"",0,""} /* End of table indicator */
};

const struct commandStruct *commandPtr = commands;
/* Declare function pointer */
void (*command)(void);
```

```
void statemachine2()
{
uint8_t cmdReturn;

for (uint8_t i = 0; i < 6; i++)
{
commandPtr = commands; /* Set ptr back to beginning */
for (uint8_t j = 0; j < 6; j++)
{ /* Iterate through every command */
//if (commandScript[i].name == commands[j].name)
//if (Current_state==command[j].name)
{
command = commandPtr->execute;
command();
if (cmdReturn)
{ /* error */
Control_RGB_LEDs(1, 0, 0);
} else
{ /* success */
Control_RGB_LEDs(0, 1, 0);
}
}
/* Next pointer element */
commandPtr++;
}
}
}


//systemstate ReadXYZHandler(void)
//{
```

```
// printf("inReadXYZ");
// eNewevent=eComplete;
//
// return sDisplay;
//}
//
//systemstate DisplayHandler(void)
//{
// printf("inDisplay");
// eNewevent=eComplete;
// return sPollSlider;
//}
//
//systemstate PollSliderHandler(void)
//{
// printf("inPollSlider");
// return sReadXYZ;
//}
//
//systemstate DisconnectHandler(void)
//{
// printf("inDisconnect");
// return sDisconnect;
//}
//
//systemstate SM1Handler(void)
//{
// printf("inSM!");
// return nextStatemachine;
//}
```

```
//
//systemstate EndHandler(void)
//{
// printf("inEnd");
// return sEnd;
//}
//
//void statemachine2(void)
//{
// //printf("\nInside statemachine2");
//
// if(statetable[Current_state][Current_event]==sReadXYZ2)
// { Current_state=sReadXYZ2;
// read_full_xyz();
// printf("\nRead is done");
// Current_state=sDisplay2;
// Current_event=eComplete2;
// statemachine2();
// }
// else if(statetable[Current_state][Current_event]==sDisplay2)
// { Current_state=sDisplay2;
// DisplayXYZ();
// avg_xyz();
// statemachine2();
// }
// else if(statetable[Current_state][Current_event]==sPollSlider2)
// { Current_state=sPollSlider;
// printf("\nPoll Slider");
// statemachine2();
// }
```

```
// else if (statetable[Current_state][Current_event]==sEnd2)
// { Current_state=sEnd2;
// state=sEnd; //for statemachine 1
// Statemachine1();
// }
// else if(statetable[Current_state][Current_event]==sDisconnect2)
// {
// Current_state=sDisconnect2;
// state=sDisconnect;
// Statemachine1();
// }
// else
// {
// printf("\nError");
// printf("\nCurrent_State is %d",Current_state);
// printf("\nCurrent_Event is %d",Current_event);
// }
//
//
//}
/*
* staemachine2.h
*
* Created on: Mar 19, 2020
* Author: nitis
*/

#ifndef STATEMACHINE2_H_
#define STATEMACHINE2_H_
```

```
void statemachine2(void);
#endif /* STATEMACHINE2_H_ */
/*
* timer.c
*
* Created on: Mar 25, 2020
* Author: nitis
*/
#include <MKL25Z4.h>
#include "statemachine1.h"
#include <stdio.h>
#include <stdint.h>
#include "touchSen.h"
#include "timer.h"
void Init_SysTick(void)
{
SysTick->LOAD = (48000000L/48);
NVIC_SetPriority(SysTick_IRQn, 3);
SysTick->VAL = 0;
SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;
}
void SysTick_Handler()
{
SysTick->CTRL &=~ (SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk);
n++;
//change state
if(n<6 && state==sPollSlider)
{
```

```
state=sReadXYZ;
currentevent=eTimeout;
}
else if(n==6 && state==sPollSlider)
{
state=nextStatemachine;
currentevent=eTimeout;
n=0;
}
}
/*
* timer.h
*
* Created on: Mar 25, 2020
* Author: nitis
*/

#ifndef TIMER_H_
#define TIMER_H_
void Init_SysTick(void);
void SysTick_Handler(void);
extern uint8_t n;
#endif /* TIMER_H_ */
/*
* touchSen.c
*
* Created on: Mar 19, 2020
* Author: nitis
*/
#include <stdint.h>
```

```
#include "MKL25Z4.h"
#include "board.h"
#include "touchSen.h"
#include "pin_mux.h"

//// Function to read touch sensor from low to high capacitance for left to right
uint16_t Touch_Scan_LH1(void)
{
uint16_t scan=0;
TSI0->DATA = ((TSI0->DATA)&~TSI_DATA_TSICH_MASK)| (TSI_DATA_TSICH(10u));
// Using channel 10 of The TSI
TSI0->DATA |= TSI_DATA_SWTS_MASK; // Software trigger for scan
while(!(TSI0->GENCS & TSI_GENCS_EOSF_MASK))
{
}
//scan = SCAN_DATA;
scan=(uint16_t)(TSI0->DATA & TSI_DATA_TSICNT_MASK );
TSI0->GENCS |= TSI_GENCS_EOSF_MASK ; // Reset end of scan flag
//DELAY_itr(BOARD_delay[2]);

return scan;
}

// Function to read touch sensor from high to low capacitance for left to right
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

// The TSI threshold isn't used is in this application
// TSI0->TSHD = TSI_TSHD_THRESH(0x00) |
// TSI_TSHD_THRESL(0x00);

}

// 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;
}
/*
* touchSen.h
*
* Created on: Mar 19, 2020
* Author: nitis
*/

#ifndef TOUCHSEN_H_
#define TOUCHSEN_H_

#include "MKL25Z4.h"
#include <stdint.h>

// Touch Sensor function prototypes
void Touch_Init(void);
int Touch_Scan_LH(void);
uint16_t Touch_Scan_LH1(void);
int Touch_Scan_HL(void);

// Macros
#define SCAN_OFFSET 544 // Offset for scan range
#define SCAN_DATA TSI0->DATA & 0xFFFF // Accessing the bits held in
TSI0_DATA_TSICNT
```

```
#endif /* TOUCHSEN_H_ */

/*****************************************************************************
*
* *
* uCUnit - A unit testing framework for microcontrollers *
* *
* (C) 2007 - 2008 Sven Stefan Krauss *
* https://www.ucunit.org *
* *
* File : uCUnit-v1.0.h *
* Description : Macros for Unit-Testing *
* Author : Sven Stefan Krauss *
* Contact : www.ucunit.org *
* *
*****************************************************************************
/

/*
* This file is part of ucUnit.
*
* You can redistribute and/or modify it under the terms of the
* Common Public License as published by IBM Corporation; either
* version 1.0 of the License, or (at your option) any later version.
*
* uCUnit is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* Common Public License for more details.
*
* You should have received a copy of the Common Public License
* along with uCUnit.
```

*

* It may also be available at the following URL:

* http://www.opensource.org/licenses/cpl1.0.txt

*

* If you cannot obtain a copy of the License, please contact the

* author.

*/

#ifndef UCUNIT_0101_H_

#define UCUNIT_0101_H_

/**
*/

/****** Customizing area ******/

/**
*/

/**

* @Macro: UCUNIT_WriteString(msg)

*

* @Description: Encapsulates a function which is called for

* writing a message string to the host computer.

*

* @param msg: Message which shall be written.

*

* @Remarks: Implement a function to write an integer to a host

* computer.

*

* For most microcontrollers a special implementation of

* printf is available for writing to a serial

* device or network. In some cases you will have

* also to implement a putch(char c) function.

*/

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

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

/**
* @Macro: UCUNIT_Safestate()
*
* @Description: Encapsulates a function which is called for
* putting the hardware to a safe state.
*
* @Remarks: Implement a function to put your hardware into
* a safe state.
*
* For example, imagine a motor controller
* application:
```

```
* 1. Stop the motor
* 2. Power brake
* 3. Hold the brake
* 4. Switch warning lamp on
* 5. Wait for acknowledge
* ...
*
*/
#define UCUNIT_Safestate() System_Safestate()

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

/**
* @Macro: UCUNIT_Init()
```

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

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

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

```
#define UCUNIT_MODE_VERBOSE

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

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

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

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

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

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

/* Action to take if check fails */
#define UCUNIT_ACTION_WARNING 0 /* Goes through the checks
with message depending on level */
```

```
#define UCUNIT_ACTION_SHUTDOWN 1 /* Stops on the end of the checklist
if any check has failed */
#define UCUNIT_ACTION_SAFESTATE 2 /* Goes in safe state if check fails */

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

/* Variables for simple statistics */
static int ucunit_checks_failed = 0; /* Numer of failed checks */
static int ucunit_checks_passed = 0; /* Number of passed checks */

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

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

/**
* @Macro: UCUNIT_DefineToStringHelper(x)
*
* @Description: Helper macro for converting a define constant into
* a string.
*
```

```
* @Param x: Define value to convert.
*
* @Remarks: This macro is used by UCUNIT_DefineToString().
*
*/
#define UCUNIT_DefineToStringHelper(x) #x

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

#ifdef UCUNIT_MODE_VERBOSE
/**
* @Macro: UCUNIT_WritePassedMsg(msg, args)
*
* @Description: Writes a message that check has passed.
*
* @Param msg: Message to write. This is the name of the called
* Check, without the substring UCUNIT_Check.
* @Param args: Argument list as string.
*
* @Remarks: This macro is used by UCUNIT_Check(). A message will
* only be written if verbose mode is set
```

```
* to UCUNIT_MODE_VERBOSE.
*
*/
#define UCUNIT_WritePassedMsg(msg, args) \
do \
{ \
UCUNIT_WriteString(__FILE__); \
UCUNIT_WriteString(":"); \
UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__)); \
UCUNIT_WriteString(": passed:"); \
UCUNIT_WriteString(msg); \
UCUNIT_WriteString("("); \
UCUNIT_WriteString(args); \
UCUNIT_WriteString(")\n"); \
} while(0)
#else
#define UCUNIT_WritePassedMsg(msg, args)
#endif

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

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

/**
* @Macro: UCUNIT_FailCheck(msg, args)
*
* @Description: Fails a check.
*
* @Param msg: Message to write. This is the name of the called
* Check, without the substring UCUNIT_Check.
* @Param args: Argument list as string.
*
* @Remarks: This macro is used by UCUNIT_Check(). A message will
* only be written if verbose mode is set
* to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
```

```
*
*/
#define UCUNIT_FailCheck(msg, args) \
do \
{ \
if (UCUNIT_ACTION_SAFESTATE==ucunit_action) \
{ \
UCUNIT_Safestate(); \
} \
UCUNIT_WriteFailedMsg(msg, args); \
ucunit_checks_failed++; \
ucunit_checklist_failed_checks++; \
} while(0)

/**
* @Macro: UCUNIT_PassCheck(msg, args)
*
* @Description: Passes a check.
*
* @Param msg: Message to write. This is the name of the called
* Check, without the substring UCUNIT_Check.
* @Param args: Argument list as string.
*
* @Remarks: This macro is used by UCUNIT_Check(). A message will
* only be written if verbose mode is set
* to UCUNIT_MODE_VERBOSE.
*
*/
#define UCUNIT_PassCheck(message, args) \
do \
{ \
```

```
UCUNIT_WritePassedMsg(message, args); \
ucunit_checks_passed++; \
} while(0)

/*****************************************************************************
*/
/* Checklist Macros */
/*****************************************************************************
*/

/**
* @Macro: UCUNIT_ChecklistBegin(action)
*
* @Description: Begin of a checklist. You have to tell what action
* shall be taken if a check fails.
*
* @Param action: Action to take. This can be:
* * UCUNIT_ACTION_WARNING: A warning message will be printed
* that a check has failed
* * UCUNIT_ACTION_SHUTDOWN: The system will shutdown at
* the end of the checklist.
* * UCUNIT_ACTION_SAFESTATE: The system goes into the safe state
* on the first failed check.
* @Remarks: A checklist must be finished with UCUNIT_ChecklistEnd()
*
*/
#define UCUNIT_ChecklistBegin(action) \
do \
{ \
ucunit_action = action; \
ucunit_checklist_failed_checks = 0; \
} while (0)
```

```
/**
* @Macro: UCUNIT_ChecklistEnd()
*
* @Description: End of a checklist. If the action was UCUNIT_ACTION_SHUTDOWN
* the system will shutdown.
*
* @Remarks: A checklist must begin with UCUNIT_ChecklistBegin(action)
*
*/
#define UCUNIT_ChecklistEnd() \
if (ucunit_checklist_failed_checks!=0) \
{ \
UCUNIT_WriteFailedMsg("Checklist",""); \
if (UCUNIT_ACTION_SHUTDOWN==ucunit_action) \
{ \
UCUNIT_Shutdown(); \
} \
} \
else \
{ \
UCUNIT_WritePassedMsg("Checklist",""); \
}

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

/**
* @Macro: UCUNIT_Check(condition, msg, args)
*
```

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

/**
* @Macro: UCUNIT_CheckIsEqual(expected,actual)
*
* @Description: Checks that actual value equals the expected value.
*
* @Param expected: Expected value.
* @Param actual: Actual value.
*
* @Remarks: This macro uses UCUNIT_Check(condition, msg, args).
*
*/
#define UCUNIT_CheckIsEqual(expected,actual) \
UCUNIT_Check( (expected) == (actual), "IsEqual", #expected "," #actual )

/**
* @Macro: UCUNIT_CheckIsNull(pointer)
*
* @Description: Checks that a pointer is NULL.
*
* @Param pointer: Pointer to check.
```

```
*
* @Remarks: This macro uses UCUNIT_Check(condition, msg, args).
*
*/
#define UCUNIT_CheckIsNull(pointer) \
UCUNIT_Check( (pointer) == NULL, "IsNull", #pointer)

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

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

```
*
*/
#define UCUNIT_CheckIsInRange(value, lower, upper) \
UCUNIT_Check( ( (value>=lower) && (value<=upper) ), "IsInRange", #value "," #lower "," #upper)

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

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

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

/**
* Checks if bit is set
*/
/**
* @Macro: UCUNIT_CheckIsBitSet(value, bitno)
*
* @Description: Checks if a bit is set in value.
*
* @Param value: Value to check.
* @Param bitno: Bit number. The least significant bit is 0.
*
* @Remarks: This macro uses UCUNIT_Check(condition, msg, args).
*
*/
#define UCUNIT_CheckIsBitSet(value, bitno) \
UCUNIT_Check( (1==(((value)>>(bitno)) & 0x01) ), "IsBitSet", #value "," #bitno)

/**
```

```
* @Macro: UCUNIT_CheckIsBitClear(value, bitno)
*
* @Description: Checks if a bit is not set in value.
*
* @Param value: Value to check.
* @Param bitno: Bit number. The least significant bit is 0.
*
* @Remarks: This macro uses UCUNIT_Check(condition, msg, args).
*
*/
#define UCUNIT_CheckIsBitClear(value, bitno) \
UCUNIT_Check( (0==(((value)>>(bitno)) & 0x01) ), "IsBitClear", #value "," #bitno)

/************************************************************************
*/
/* Testcases */
/************************************************************************
*/

/**
* @Macro: UCUNIT_TestcaseBegin(name)
*
* @Description: Marks the beginning of a test case and resets
* the test case statistic.
*
* @Param name: Name of the test case.
*
* @Remarks: This macro uses UCUNIT_WriteString(msg) to print the name.
*
*/
#define UCUNIT_TestcaseBegin(name) \
do \
```

```
{ \
UCUNIT_WriteString("\n\r=====================================\n"); \
UCUNIT_WriteString(name); \
UCUNIT_WriteString("\n\r=====================================\n"); \
ucunit_testcases_failed_checks = ucunit_checks_failed; \
} \
while(0)

/**
* @Macro: UCUNIT_TestcaseEnd()
*
* @Description: Marks the end of a test case and calculates
* the test case statistics.
*
* @Remarks: This macro uses UCUNIT_WriteString(msg) to print the result.
*
*/
#define UCUNIT_TestcaseEnd() \
do \
{ \
UCUNIT_WriteString("=====================================\n\r"); \
if( 0==(ucunit_testcases_failed_checks - ucunit_checks_failed) ) \
{ \
UCUNIT_WriteString("Testcase passed.\n\r"); \
ucunit_testcases_passed++; \
} \
else \
{ \
UCUNIT_WriteFailedMsg("EndTestcase",""); \
ucunit_testcases_failed++; \
} \
```

```
UCUNIT_WriteString("====================================\n\r"); \
} \
while(0)

/*****************************************************************************
*/
/* Support for code coverage */
/*****************************************************************************
*/

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

```
}

/**
* @Macro: UCUNIT_ResetTracepointCoverage()
*
* @Description: Resets the trace point coverage state to 0.
*
* @Param index: Index of the trace point.
*
* @Remarks: This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
*
*/
#define UCUNIT_ResetTracepointCoverage() \
for (ucunit_index=0; ucunit_index<UCUNIT_MAX_TRACEPOINTS; ucunit_index++) \
{ \
ucunit_checkpoints[ucunit_index]=0; \
}

/**
* @Macro: UCUNIT_CheckTracepointCoverage(index)
*
* @Description: Checks if a trace point was covered.
*
* @Param index: Index of the trace point.
*
* @Remarks: This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
*
*/
#define UCUNIT_CheckTracepointCoverage(index) \
UCUNIT_Check( (ucunit_checkpoints[index]!=0), "TracepointCoverage", #index);
```

```
/*****************************************************************************
*/
/* Testsuite Summary */
/*****************************************************************************
*/

/**
* @Macro: UCUNIT_WriteSummary()
*
* @Description: Writes the test suite summary.
*
* @Remarks: This macro uses UCUNIT_WriteString(msg) and
* UCUNIT_WriteInt(n) to write the summary.
*
*/
#define UCUNIT_WriteSummary() \
{ \
UCUNIT_WriteString("\n\r****************************************"); \
UCUNIT_WriteString("\n\rTestcases: failed: "); \
UCUNIT_WriteInt(ucunit_testcases_failed); \
UCUNIT_WriteString("\n\r passed: "); \
UCUNIT_WriteInt(ucunit_testcases_passed); \
UCUNIT_WriteString("\n\rChecks: failed: "); \
UCUNIT_WriteInt(ucunit_checks_failed); \
UCUNIT_WriteString("\n\r passed: "); \
UCUNIT_WriteInt(ucunit_checks_passed); \
UCUNIT_WriteString("\n\r****************************************\n"); \
}

#endif /*UCUNIT_H_*/
```