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_5.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"



#include "i2c.h"
#include "gpio.h"
#include "led.h"
#include "mma8451.h"
#include "touch_sen.h"
#include "timer.h"
#include "state.h"
#include "logger.h"
#include "command_parser.h"
#include "buffer.h"
#include "test.h"
```

```c
/*
 * @brief   Application entry point.
 */


//#define DEBUG_MODE
//#define NORMAL_MODE
#define TEST_MODE

//#define ECHO_SUBMODE
//#define APPL_SUBMODE

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();

    LOG_INFO("Hello, PES Project 5\n");
    LOG_INFO("LED will blink green for a sec to indicate start\n");
```

```
    turn_LED_green(on);
    mdelay(1000);
    turn_LED_green(off);

    /* Enter an infinite loop */
    while(1)
    {
      // DEBUG MODE
#ifdef DEBUG_MODE

#define INCLUDE_LOG_DEBUG 1
#ifdef ECHO_SUBMODE
#endif

#ifdef APPL_SUBMODE
      command_parser();
#endif

#endif
      // NORMAL MODE
#ifdef NORMAL_MODE

#ifdef ECHO_SUBMODE
#endif

#ifdef APPL_SUBMODE
      command_parser();
#endif
#endif
      // TEST MODE
#ifdef TEST_MODE
```

```
      Test();
#endif

    }
    return 0 ;
}
```

timer.c (main.c)

```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 BLOCK_WAITING
#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;

/*************** Interrupt Hanlder ***************/

void SysTick_Handler(void)
{
	msec_count ++;
	if (delay_flag == true)
	{
		if (msec_count == target_msec_count)
		{
			delay_flag = false;
		}
	}
}

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

void Init_SysTick(void) {
	SysTick->LOAD = (48000L-1L);// count 1 msec
	NVIC_SetPriority(SysTick_IRQn, 4); // 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


/
*****************************************************************************//
**
 * @brief
 *   get the time had been run since powered up
 *
 * @note
 *   Please #include config.h
 *   Now only based on 1000 Hz clock,
 *
 * @return
 *   runtime [in unit ms]

*****************************************************************************
*/
uint64_t timerGetRunTimeMilliseconds(void)
{
      return msec_count;
}
```

buffer.c

```c
#include "buffer.h"

errors_t error;


errors_t initBuffer(struct circular_buffer *cbuff){
	error=no_error;
	cbuff->buff_ptr= malloc(BUFFER_SIZE * sizeof(char));
	if (!verifyValidPointer(cbuff)){
		for (int i=0; i<BUFFER_SIZE; i++){
			cbuff->buff_ptr[i]='0';
		}
		if (!verifyInit(cbuff)){
			cbuff->head=0;
			cbuff->tail=0;
			cbuff->count=0;
		}
	}
	return error;
}
errors_t verifyInit(struct circular_buffer *cbuff){
	error=no_error;
	for (int i=0; i<BUFFER_SIZE; i++){
		if (cbuff->buff_ptr[i]!='0'){
			error=error_buf_init;
		}
	}
	return error;
}
errors_t verifyValidPointer(struct circular_buffer *cbuff){
	error=no_error;
	if (cbuff->buff_ptr==NULL){
```

```
            error=error_buff_ptr;
      }
      return error;
}
errors_t destroyBuffer(struct circular_buffer *cbuff){
      error=no_error;
      free(cbuff->buff_ptr);
      return error;
}
errors_t addItem(struct circular_buffer *cbuff, char item){
      error=no_error;
      if (!isFull(cbuff)){
            cbuff->buff_ptr[cbuff->head]=item;
            cbuff->head=(cbuff->head+1)%BUFFER_SIZE;
            cbuff->count++;
      }
      return error;
}
errors_t removeItem(struct circular_buffer *cbuff, char *item){
      error=no_error;
      if (!isEmpty(cbuff)){
            *item=cbuff->buff_ptr[cbuff->tail];
            cbuff->tail=(cbuff->tail+1)%BUFFER_SIZE;
            cbuff->count--;
      }
      return error;
}
errors_t isFull(struct circular_buffer *cbuff){
      error=no_error;
      if (cbuff->count==BUFFER_SIZE){
            error=error_full;
```

```
	}
	return error;
}
errors_t isEmpty(struct circular_buffer *cbuff){
	error=no_error;
	if (cbuff->count==0){
		error=error_empty;
	}
	return error;
}
```

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] ={"off","on","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");
	}
}
```

command_parser.c

```c
/*
 *    .c
 *
 *  Created on: Mar 9, 2020
 *      Author: Curry
 */

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

#include "command_parser.h"
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include "led.h"
#include "logger.h"
/********************** Define **********************/
#define MAX_USER_BUF 64
#define MAX_USER_ARG 4




/********************** Function **********************/
void command_parser(void)
{
      // Initialize char array
      uint8_t str[MAX_USER_BUF];

      uint8_t i = 0;
    turn_LED_blue(on);
      //takes all the characters until enter is pressed
```

```
	while((str[i]=getchar())!='\n'){
		//increment the index of the character array
		i++;
		// protect against long input
		if(i == MAX_USER_BUF-1)
		{
		    str[MAX_USER_BUF-1] = '\n';
		    LOG_ERROR("user input too long, the rest of the string will
be counted next time \n");
		    break;
		}
	}
	turn_LED_blue(off);

	i = 0;
	uint8_t char_count[255];
	memset(char_count,0,255);
	// go through the string
	while(str[i]!='\n')
	{
		// count which character is incremented
		char_count[str[i]]++;
		i++;
	}
	turn_LED_green(on);

	// print them out
	i = 'A';
	while(i<='Z')
	{
		if(char_count[i]!=0)
		{
```

```
                printf("%c - %u; ",i,char_count[i]);
            }
            i++;
        }
        i = 'a';
        while(i<='z')
        {
            if(char_count[i]!=0)
            {
                printf("%c - %u; ",i,char_count[i]);
            }
            i++;
        }
        printf("\n");
        printf("\n");
        turn_LED_green(off);
}
```

test.c

```
/*
 * test.c
 *
 *  Created on: Apr 6, 2020
 *      Author: user
 */


#include "uCUnit.h"
#include "buffer.h"


//adapted from
//https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-microcontrollers/
void Test(void) {
      struct circular_buffer cbuff;
    char item;


    UCUNIT_Init(); /* initialize framework */
      UCUNIT_TestcaseBegin("Buffer Tests");


      UCUNIT_CheckIsEqual(0, initBuffer(&cbuff));
      UCUNIT_CheckIsEqual(0, destroyBuffer(&cbuff));


      initBuffer(&cbuff);
      cbuff.head=BUFFER_SIZE-1;
      cbuff.tail=BUFFER_SIZE-1;
      UCUNIT_CheckIsEqual(0, addItem(&cbuff,'a'));
      UCUNIT_CheckIsEqual(0, cbuff.head);
      UCUNIT_CheckIsEqual(0, addItem(&cbuff,'b'));
      UCUNIT_CheckIsEqual(0, addItem(&cbuff,'c'));
      UCUNIT_CheckIsEqual(0, removeItem(&cbuff, &item));
```

```
	printf("getitem: %s", item);
	UCUNIT_CheckIsEqual(0, cbuff.tail);
	UCUNIT_CheckIsEqual(0, removeItem(&cbuff, &item));
	printf("getitem: %s", item);
	UCUNIT_CheckIsEqual(0, removeItem(&cbuff, &item));
	printf("getitem: %s", item);
	destroyBuffer(&cbuff);

	initBuffer(&cbuff);
	for (int i=0; i<BUFFER_SIZE; i++){
		addItem(&cbuff,'a');
	}
	UCUNIT_CheckIsEqual(1U, addItem(&cbuff,'a'));

	for (int i=0; i<BUFFER_SIZE; i++){
		printf("%d, %s", removeItem(&cbuff, &item),item);
	}
	UCUNIT_CheckIsEqual(2U, removeItem(&cbuff, &item));
	destroyBuffer(&cbuff);



	UCUNIT_TestcaseEnd();

	/* finish all the tests */
	UCUNIT_WriteSummary();
	UCUNIT_Shutdown();
}
```