Lab 2 Code

Michele Piperni, Aidan Johnson

Lab2_BF609_Core0_uTTCOSg2017_main.cpp

```
/*************************************************************************************
*
* AUTO-GENERATED COMMENT - DO NOT MODIFY
* Author: aidan
* Date: Thu 2019/10/24 at 02:31:12 PM
* File Type: TTCOS Main File
*************************************************************************************
/
#include "Lab2_BF609_Core0_uTTCOSg2017_main.h"

//change these names
extern volatile char ID_frontPanelThread1 = 0;
extern volatile char ID_frontPanelThread2 = 0;
extern volatile char ID_frontPanelThread3 = 0;
extern volatile char ID_frontPanelThread4 = 0;
extern volatile char ID_frontPanelThread5 = 0;
extern volatile char ID_REBThread1 = 0;
extern volatile char ID_REBThread2 = 0;
extern volatile char ID_REBThread3 = 0;

bool My_Init_LEDInterface_Done = false;
bool My_Init_SwitchInterface_Done = false;
bool My_Init_GPIO_REB_Input_Done = false;
bool My_Init_GPIO_REB_Output_Done = false;
bool My_Init_GPIO_REB_Done = false;


void main(void)
{
	int numBackgroundThreads = 5; // Make maxNumberThreads at least 5 larger than
the number of threads you plan to add
	int numberYourThreads = 8; //We have 8 threads (5 front panel threads and 3
REB threads)
	int maxNumberThreads = numBackgroundThreads + numberYourThreads;

	My_Init_SwitchInterface();  //This function is initiating the switches on the
panel
	My_Init_GPIO_REB_InputASM(); //This function is initiating the switches on the
board
	My_Init_LEDInterface();  //This function is initiating the LEDS on the panel
```

```
    My_Init_GPIO_REB_OutputASM(); //This function is initiating the LEDs on the
board

    Custom_uTTCOS_OS_Init(TIC_CONTROL_VALUE);  // Need to update to handle core-
timer interrupts

    //Code to run the FP threads
    ID_frontPanelThread1 = uTTCOSg_AddThread(frontPanelThread1, NO_DELAY, 0.25 *
ONE_SECOND);
    ID_frontPanelThread2 = uTTCOSg_AddThread(frontPanelThread2, NO_DELAY, 0.4 *
ONE_SECOND);
    ID_frontPanelThread3 = uTTCOSg_AddThread(frontPanelThread3, NO_DELAY, 0.5 *
ONE_SECOND);
    ID_frontPanelThread4 = uTTCOSg_AddThread(frontPanelThread4, NO_DELAY, 0.01 *
ONE_SECOND);
    ID_frontPanelThread5 = uTTCOSg_AddThread(frontPanelThread5, NO_DELAY, 0.01 *
ONE_SECOND);


    //Code to run the REB threads
    ID_REBThread1 = uTTCOSg_AddThread(REBThread1, NO_DELAY, 0.25 * ONE_SECOND);
    ID_REBThread2 = uTTCOSg_AddThread(REBThread2, NO_DELAY, 0.01 * ONE_SECOND);
    ID_REBThread3 = uTTCOSg_AddThread(REBThread3, NO_DELAY, 1.3 * ONE_SECOND);


    uTTCOSg_Start_CoreTimer_Scheduler(maxNumberThreads);   //  Start the scheduler
timer
                                 // Execution time of TT_COS_Dispatch( ) and
TT_COS_Update( ) improved by specifying maxNumberTasks

    while (1)
    {
        // Wait, in low power mode, for an interrupt
        // The interrupt service routine calls TTCOS_Update( )
        // uTTCOSg_GoToSleep( );                // Need to update to handle
coretimer interrupts
        Idle_WaitForInterrupts_ASM();

        // Run all the threads in the system according
        // to whether their delays have expired
        uTTCOSg_DispatchThreads();
    }
}
```

Lab2_BF609_Core0_uTTCOSg2017_main.h

```
/*************************************************************************************
*
* AUTO-GENERATED COMMENT - DO NOT MODIFY
* Author: aidan
* Date: Thu 2019/10/24 at 02:31:12 PM
* File Type: uTTCOS Task Header File
*************************************************************************************
/

#ifndef LAB2_BF609_CORE0_UTTCOSG2017_H
#define LAB2_BF609_CORE0_UTTCOSG2017_H

#include "faultyLED1_Thread.h"
#include <uTTCOSg2017/uTTCOSg.h>
#include <GPIO2017/ADSP_GPIO_interface.h>
#include <stdio.h>

#include "../../ENCM511_SpecificFiles/ENCM511_src/Example_faultyLED1_Thread.h"
#include "../../ENCM511_SpecificFiles/ENCM511_src/Example_uTTCOSg2017_main.h"
#include "../../ENCM511_SpecificFiles/ENCM511_include/FrontPanel_LED_Switches.h"

#if defined(__ADSPBF609__)
#define TIC_CONTROL_VALUE ((unsigned long int) 4800000)          // BF609 EMULATOR
#define TICS_PER_SECOND   100
#define ONE_SECOND               TICS_PER_SECOND           // If TICS_CONTROL_VALUE
Adjusted correctly
#define RUN_ONCE                 0
#define NO_DELAY                 0
#else
#error "Unknown ADSP or ARM processor"
#endif

// extern "C" void BlackfinBF533_uTTCOSg_Audio_Rx_Tx_Task(void);
extern "C" void SHARC21469_uTTCOSg_Audio_Rx_Tx_Task(void);
extern "C" void ADSP_SC589_uTTCOSg_Audio_Rx_Tx_Task(void);

// TODO -- Once you have demonstrated the idea of uTTCOS working with print
statements
// Comment out the following include statement
// DON'T USE PRINT STATEMENT INSIDE uTTCOS as it is a real time system and
// print statements run on the HIGH priority emulator interrupt and disrupt real time
operations
#include "faultyLED1_Thread.h"
#include "Front_Panel_Threads.h"
#include "REB_Threads.h"
#include "Lab1And0FrontPanelFunctions.h"

extern "C" void ReadCycles_ASM(void);
```

```
void Custom_uTTCOS_OS_Init(unsigned long int);
extern "C" void Idle_WaitForInterrupts_ASM(void);
void uTTCOSg_Start_CoreTimer_Scheduler(unsigned int maxNumberThreads);


//Function Prototypes
void My_Init_SwitchInterface(void);
void My_Init_LEDInterface(void);
void My_Init_GPIO_REB_Input(void);
void My_Init_GPIO_REB_Output(void);

//extern "C" means that you are declaring these functions and they can be used in a
different file as the compiler knows they are in an external file
//Pretty much they act in this case as global functions and are declared below

extern "C" unsigned long long int ReadProcessorCyclesASM(void);
extern "C" void My_Write_GPIO_REB_OutputASM(unsigned short int);
extern "C" unsigned short int My_Read_GPIO_REB_InputASM(void);
extern "C" void My_Init_GPIO_REB_InputASM(void);
extern "C" void My_Init_GPIO_REB_OutputASM(void);

//These variables are all declared and initialized to be false to begin until the
respective function is called
//to initialize the respective equipment
extern bool My_Init_SwitchInterface_Done;
extern bool My_Init_LEDInterface_Done;
extern bool My_Init_GPIO_REB_Input_Done;
extern bool My_Init_GPIO_REB_Output_Done;
extern bool My_Init_GPIO_REB_Done;

//Making the ID extern
extern volatile char ID_frontPanelThread1;
extern volatile char ID_frontPanelThread2;
extern volatile char ID_frontPanelThread3;
extern volatile char ID_frontPanelThread4;
extern volatile char ID_frontPanelThread5;
extern volatile char ID_REBThread1;
extern volatile char ID_REBThread2;
extern volatile char ID_REBThread3;


#endif
```

Front_Panel_Threads.cpp

```
/*
 * Front_Panel_Threads.cpp
 *
 *  Created on: Nov 19, 2019
 *      Author: aidan
 */

#include "Front_Panel_Threads.h"
#include "Lab1And0FrontPanelFunctions.h"

//Global Variables
static unsigned int pauseFrontPanelThreadFour = 0;
static unsigned long long int displayRate = DISPLAYRATEVALUE;
static unsigned long long int newDisplayRate= DISPLAYRATEVALUE;

void frontPanelThread1(void)
{
       static unsigned int LEDState = 0;
       FRONTPANEL_LED_8BIT_VALUE lastLEDStateValue;
       unsigned int nextLEDState = LEDState;

       switch(LEDState)
       {
             case 0: //This case is where the LED is off
                    lastLEDStateValue = (myReadFrontPanelLEDs() & LED8MASK);
//Zeroing the 8th LED bit
                    myWriteFrontPanelLEDs(lastLEDStateValue);
                    #if DEBUG
                          printf("In Task_frontPanelThread1 LED OFF at time 0x%8X
system cycles\n", ReadProcessorCyclesASM());
                    #endif
                    nextLEDState = 1;
                    break;

             case 1: //This case is where the LED is on
                    lastLEDStateValue = (myReadFrontPanelLEDs() & LED8MASK);
//Zeroing the 8th LED bit
                    lastLEDStateValue = lastLEDStateValue | LED8VALUE; //Placing a
one into the 8th LED bit
                    myWriteFrontPanelLEDs(lastLEDStateValue);
                    #if DEBUG
                          printf("In Task_frontPanelThread1 LED ON at time 0x%8X
system cycles\n", ReadProcessorCyclesASM());
                    #endif
                    nextLEDState = 0;
                    break;
       }

       LEDState = nextLEDState; //placing the nextLEDState into the LEDState variable
```

```
}

void frontPanelThread2(void)
{
	static unsigned int LEDState = 0;
	FRONTPANEL_LED_8BIT_VALUE lastLEDStateValue;
	unsigned int nextLEDState;

	switch(LEDState)
	{
		case 0: //This case is where the LED is off
			lastLEDStateValue = (myReadFrontPanelLEDs() & LED7MASK);
//Zeroing the 7th LED bit
			myWriteFrontPanelLEDs(lastLEDStateValue);
			#if DEBUG
				printf("In Task_frontPanelThread2 LED OFF at time 0x%8X
system cycles\n", ReadProcessorCyclesASM());
			#endif
			nextLEDState = 1;
			break;

		case 1: //This case is where the LED is on
			lastLEDStateValue = (myReadFrontPanelLEDs() & LED7MASK);
			lastLEDStateValue = (lastLEDStateValue | LED7VALUE); //Placing a
one into the 7th LED bit
			myWriteFrontPanelLEDs(lastLEDStateValue);
			#if DEBUG
				printf("In Task_frontPanelThread2 LED ON at time 0x%8X
system cycles\n", ReadProcessorCyclesASM());
			#endif
			nextLEDState = 2;
			break;

		case 2: //This is another case where the LED is on
			lastLEDStateValue = (myReadFrontPanelLEDs() & LED7MASK);
			lastLEDStateValue = (lastLEDStateValue | LED7VALUE); //Placing a
one into the 7th LED bit
			myWriteFrontPanelLEDs(lastLEDStateValue);
			#if DEBUG
				printf("In Task_frontPanelThread2 LED ON at time 0x%8X
system cycles\n", ReadProcessorCyclesASM());
			#endif
			nextLEDState = 0;
			break;
	}

	LEDState = nextLEDState; //Placing the nextLEDState into the LEDState variable
}

void frontPanelThread3(void)
{
	static unsigned int LEDState = 0;
	FRONTPANEL_LED_8BIT_VALUE lastLEDStateValue;
	unsigned int nextLEDState = LEDState;
```

```
    switch(LEDState)
    {
        case 0: //This case is where LED 1 and LED 2 are off representing 0
            lastLEDStateValue = (myReadFrontPanelLEDs() & LED1TO2MASK);
            myWriteFrontPanelLEDs(lastLEDStateValue); //This will be
displaying the value 0
            #if DEBUG
                printf("In Task_frontPanelThread3 the LED Value is %d at
time 0x%8X system cycles\n", lastLEDStateValue, ReadProcessorCyclesASM());
            #endif
            nextLEDState = 1;
            break;

        case 1: //This case is where LED 1 is on and LED 2 is off representing 1
            lastLEDStateValue = (myReadFrontPanelLEDs() & LED1TO2MASK);
            lastLEDStateValue = (lastLEDStateValue | LED1VALUE); //This will
be displaying the value 1
            myWriteFrontPanelLEDs(lastLEDStateValue);
            #if DEBUG
                printf("In Task_frontPanelThread3 the LED Value is %d at
time 0x%8X system cycles\n", lastLEDStateValue, ReadProcessorCyclesASM());
            #endif
            nextLEDState = 2;
            break;

        case 2: //This case is where LED 1 is off and LED 2 is on representing 2
            lastLEDStateValue = (myReadFrontPanelLEDs() & LED1TO2MASK);
            lastLEDStateValue = (lastLEDStateValue | LED2VALUE); //This will
be displaying the value 2
            myWriteFrontPanelLEDs(lastLEDStateValue);
            #if DEBUG
                printf("In Task_frontPanelThread3 the LED Value is %d at
time 0x%8X system cycles\n", lastLEDStateValue, ReadProcessorCyclesASM());
            #endif
            nextLEDState = 3;
            break;

        case 3: //This case is where LED 1 is on and LED 2 is on representing 3
            lastLEDStateValue = (myReadFrontPanelLEDs() & LED1TO2MASK);
            lastLEDStateValue = (lastLEDStateValue | LED1TO2VALUE); //This
will be displaying the value 3
            myWriteFrontPanelLEDs(lastLEDStateValue);
            #if DEBUG
                printf("In Task_frontPanelThread3 the LED Value is %d at
time 0x%8X system cycles\n", lastLEDStateValue, ReadProcessorCyclesASM());
            #endif
            nextLEDState = 0;
            break;
    }

    LEDState = nextLEDState; // Placing nextLEDState into the LEDState variable
}

void frontPanelThread4(void)
{
	static const FRONTPANEL_LED_8BIT_VALUE initials[] = {0x00, 0xe0, 0x1c, 0x13,
0x1c, 0xe0, 0x00, 0xc0, 0x00, 0xe0, 0xc3, 0xff, 0x03, 0x00, 0xc0}; //Aidan's initials
array from Lab 0
	static unsigned int index = 0;

	FRONTPANEL_LED_8BIT_VALUE lastLEDValue;
	FRONTPANEL_LED_8BIT_VALUE newLEDValue;

	unsigned int nextState;

	if(pauseFrontPanelThreadFour == 0) //This code will be run if thread four is
not paused
	{
		lastLEDValue = (myReadFrontPanelLEDs() & LED3TO6MASK);
		newLEDValue = initials[index] & LED3TO6VALUE; //newLEDValue will have
the value in the initials array at the index, index for only the 3rd bit to the 6th
bit
		newLEDValue = newLEDValue | lastLEDValue; //Putting the lastLEDValue
with the value from the array to make sure that no bits were overwritten
		myWriteFrontPanelLEDs(newLEDValue);
		if (displayRate == 0) //if displayRate is equal to zero display the next
value
		{
			displayRate = DISPLAYRATEVALUE;
			index++; //Incrementing the index
			#if DEBUG
				printf("In Task_frontPanelThread4 the LED Value is %d at
time 0x%8X system cycles\n", newLEDValue, ReadProcessorCyclesASM());
			#endif
			if(initials[index] == INITIALSARRAYLENGTH) //Making sure we reset
the initials array when the index gets to its length
			{
				index = 0;
			}
		}
		displayRate--; //Decrementing the displayRate
	}
}

void frontPanelThread5(void)
{
	static unsigned int switchState = 0;
	FRONTPANEL_LED_8BIT_VALUE lastLEDStateValue;
	unsigned int nextSwitchState;
	unsigned int switchOneValue;
	static unsigned long long int timeSwitchIsPressedFor = 0;
	static unsigned long long int timeThatThePressOccured;
	static unsigned long long int currentTime;

	switch(switchState)
	{
		case 0: //This case is where it is checking if SW on the Front Panel has
been pressed and if it has it records the time that it is pressed at
			switchOneValue = (myReadFrontPanelSwitches()) &
FRONTPANELSWITCHONEVALUE;
```

```
            if(switchOneValue == 1)
            {
                timeThatThePressOccured = ReadProcessorCyclesASM();
                #if DEBUG
                    printf("In Task_frontPanelThread5 the switch State
is %d at time 0x%8X system cycles\n", switchState, ReadProcessorCyclesASM());
                #endif
                nextSwitchState = 1;
            }
            else
            {
                nextSwitchState = 0;
            }
            break;

        case 1: //This case is where it is checking if SW1 is still pressed or
if it has been released
            switchOneValue = (myReadFrontPanelSwitches()) &
FRONTPANELSWITCHONEVALUE;
            if (switchOneValue == 1)
            {
                nextSwitchState = 1;
            }
            else //If SW1 has been relased it records the time and then
calculates the time that the SW was pressed for
            {
                currentTime = ReadProcessorCyclesASM();

                #if DEBUG
                    printf("In Task_frontPanelThread5 the switch State
is %d at time 0x%8X system cycles\n", switchState, ReadProcessorCyclesASM());
                #endif

                timeSwitchIsPressedFor = currentTime -
timeThatThePressOccured;

                #if DEBUG
                    printf("%llu \n", timeSwitchIsPressedFor);
                    printf("%f \n",
(timeSwitchIsPressedFor/(double)(ONESECOND)));
                #endif

                if ((timeSwitchIsPressedFor >= ONESECOND) &&
(timeSwitchIsPressedFor <= TWOSECONDS)) //If the time that SW1 was pressed is between
1-2seconds it will speed up the FP LEDS 3-6
                {
                    pauseFrontPanelThreadFour = 0;
                    nextSwitchState = 2;

                    #if DEBUG
                    printf("You have speed up the Front Panel LEDS \n");
                    #endif
                }
                else if((timeSwitchIsPressedFor >= THREESECONDS) &&
(timeSwitchIsPressedFor <= FOURSECONDS)) //If the time that SW1 was pressed is
between 3-4 seconds it will slow down the FP LEDS 3-6
                {
                    pauseFrontPanelThreadFour = 0;
                    nextSwitchState = 3;

                    #if DEBUG
                        printf("You have slowed down the Front Panel
LEDS \n");
                    #endif
                }
                else
                {
                    pauseFrontPanelThreadFour = 1; //Giving the
pauseFrontPanelThreadFour variable a one will pause the FP LEDS 3-6
                    nextSwitchState = 0;

                    #if DEBUG
                    printf("You have paused the Front Panel LEDs \n");
                    #endif
                }
            }
            break;

        case 2: //This case is where the DisplayRate gets incremented, which
speeds up the time the initials array pattern is displayed by the LEDs
            newDisplayRate = newDisplayRate / INCREMENTORDECREMENTVALUE;
            if (newDisplayRate == 0)
            {
                newDisplayRate = 1; //This is making sure that the
newDisplayRate doesn't ever stop the LEDs
            }

            nextSwitchState = 0;

            #if DEBUG
                printf("In Task_frontPanelThread5 the switch State is %d
at time 0x%8X system cycles\n", switchState, ReadProcessorCyclesASM());
            #endif
            break;

        case 3: //This case is where the DisplayRate gets decremented, which
slows down the time the initials array pattern is displayed by the LEDs
            newDisplayRate = newDisplayRate * INCREMENTORDECREMENTVALUE;
            nextSwitchState = 0;
            #if DEBUG
                printf("In Task_frontPanelThread5 the switch State is %d
at time 0x%8X system cycles\n", switchState, ReadProcessorCyclesASM());
            #endif
            break;
    }

    switchState = nextSwitchState;
}
```

Front_Panel_Threads.h

```
/*
 * Front_Panel_Threads.h
 *
 *  Created on: Nov 19, 2019
 *      Author: aidan
 */

#ifndef FRONT_PANEL_THREADS_H_
#define FRONT_PANEL_THREADS_H_


#include "Lab2_BF609_Core0_uTTCOSg2017_main.h"


//MACROS
#define LED8VALUE 0x80
#define LED8MASK (~LED8VALUE)

#define LED7VALUE 0x40
#define LED7MASK (~LED7VALUE)

#define LED6VALUE 0x20
#define LED6MASK (~LED6VALUE)

#define LED5VALUE 0x10
#define LED5MASK (~LED5VALUE)

#define LED4VALUE 0x08
#define LED4MASK (~LED4VALUE)

#define LED3VALUE 0x04
#define LED3MASK (~LED3VALUE)

#define LED2VALUE 0x02
#define LED2MASK (~LED2VALUE)

#define LED1VALUE 0x01
#define LED1MASK (~LED1VALUE)

#define LED3TO6VALUE 0x3c
#define LED3TO6MASK (~LED3TO6VALUE)

#define LED1TO2VALUE 0x03
#define LED1TO2MASK (~LED1TO2VALUE)

#define FRONTPANELSWITCHONEVALUE 0x01
#define FRONTPANELSWITCHONEMASK (~FRONTPANELSWITCHONEVALUE)

#define DISPLAYRATEVALUE 50
#define INITIALSARRAYLENGTH 15
#define INCREMENTORDECREMENTVALUE 1.25
```

```
#define ONESECOND ((unsigned long int) 480000000)
#define TWOSECONDS (ONESECOND*2)
#define THREESECONDS (ONESECOND*3)
#define FOURSECONDS (ONESECOND*4)

#define MASK_KEEP_LOWER_FIVE_BITS 0x1F

#define DEBUG 0

//Function Prototypes used in Front_Panel_Thread.cpp
void frontPanelThread1(void);
void frontPanelThread2(void);
void frontPanelThread3(void);
void frontPanelThread4(void);
void frontPanelThread5(void);

#endif /* FRONT_PANEL_THREADS_H_ */
```

REB_Threads.cpp

```
/*
 * REB_Threads.cpp
 *
 *  Created on: Nov 20, 2019
 *      Author: miche
 */
#include "REB_Threads.h"
#include "Lab2_BF609_Core0_uTTCOSg2017_main.h"

void REBThread1(void)
{
//REB Thread 1: Task that counts from 0 – 15 in ¼ second intervals and displays on
REB LEDS 4 – 7
//Use 4 – 7 as these are not hidden behind 50 pin cable as a 0 - 3
	static unsigned short Thread1_Counter = 0;

	if(Thread1_Control)
	{
		My_Write_GPIO_REB_OutputASM(Thread1_Counter);
		printf("Writing numbers 0-15\n");
	}

	Thread1_Counter++;

	if (Thread1_Counter == 16)
		Thread1_Counter = 0;
}


void REBThread2(void)
{
//Task that ON COMMAND will read REB switches 0 – 3 and store them in an array the
number of values
//must be changeable at demo time (no more than 100)
//Must be user and power friendly (meaning respond to request to store values in a
humanly useful time, but does
//not run so often that wastes batter power
//As in Lab 1 – FP SW5 and SW4 are available to help control this task
	unsigned short int switchPattern = My_Read_GPIO_REB_InputASM();

	//write to the array if switch 4 is pressed
	if((myReadFrontPanelSwitches() & FP_SW4_ON) == FP_SW4_ON)
	{
		hardWareArray[array_index] = switchPattern;
		array_index++;
		printf("\n read pattern \n");

	}

	//terminate thread if writing is done by pressing switch 5 or going over 100
values
```

```
        if((myReadFrontPanelSwitches() & FP_SW5_ON) == FP_SW5_ON ||array_index >= 100)
        {
            Thread1_Control = false;
            printf("\ngoing into thread 3\n");
        }
    }

    void REBThread3(void)
    {
        if(!Thread1_Control)
        {
            static int index = 0;
            printf("printing index %d\n", index);
            My_Write_GPIO_REB_OutputASM(hardWareArray[index]);
            index++;

            //Will exit this thread if it has gone through entire hardware array or
    if switch 2 is pressed
            if(array_index <= index || myReadFrontPanelSwitches() == FP_SW2_ON )
            {
                // reset
                index = 0;
                array_index = 0;
                Thread1_Control = true;
            }
        }
    }
```

REB_Threads.h

```
/*
 * REB_Threads.h
 *
 *  Created on: Nov 20, 2019
 *      Author: miche
 */

#ifndef REB_THREADS_H_
#define REB_THREADS_H_

#include "Lab2_BF609_Core0_uTTCOSg2017_main.h"
//All the REB ASM functions are in the main.h file

//Function Prototypes used in REB_Threads.cpp
void REBThread1(void);
void REBThread2(void);
void REBThread3(void);

//Variables used in the REB threads file
static bool Thread1_Control = true;
static unsigned short int hardWareArray [100];
static int array_index = 0;

//Switches
#define FP_SW5_ON 0x10
#define FP_SW_OFF 0x00
#define FP_SW4_ON 0x08
#define FP_SW3_ON 0x04
#define FP_SW2_ON 0x02
#define FP_SW1_ON 0x01


#endif /* REB_THREADS_H_ */
```

\

Lab1And0FrontPanelFunctions.cpp

```
/*
 * Lab1And0FrontPanelFunctions.cpp
 *
 *  Created on: Nov 21, 2019
 *      Author: aidan
 */
#include "Lab1And0FrontPanelFunctions.h"

void myWriteFrontPanelLEDs(unsigned char neededLEDValue) //This function is writing
the values to be displayed by the LEDs
{
        if (My_Init_LEDInterface_Done == false)
        {
                return;
        }

        Write_GPIO_FrontPanelLEDS(neededLEDValue); //Writing the value to the panel of
LEDs
}

unsigned char myReadFrontPanelSwitches(void) //This function is reading the switches
from the panel
{
                if (My_Init_SwitchInterface_Done == false)
                {
                        return GARBAGEVALUE;
                }
                FRONTPANEL_LED_8BIT_VALUE activeLowValues =
Read_GPIO_FrontPanelSwitches();
                FRONTPANEL_LED_8BIT_VALUE activeHighValues = ~activeLowValues;
                FRONTPANEL_LED_8BIT_VALUE wantedSwitchValueActiveHigh = activeHighValues
& MASK_KEEP_LOWER_FIVE_BITS;

                return wantedSwitchValueActiveHigh;
}

unsigned char myReadFrontPanelLEDs(void)
{
        if (My_Init_LEDInterface_Done == false)
        {
                return GARBAGEVALUE;
        }

        return Read_GPIO_FrontPanelLEDS();
}
```

Lab0And1InitFunctions.cpp

```
/*
 * Lab0And1InitFunctions.cpp
 *
 *  Created on: Nov 21, 2019
 *      Author: aidan
 */
#include "Lab2_BF609_Core0_uTTCOSg2017_main.h"

void My_Init_SwitchInterface(void) //This function is initializing the Switches on
the Front Panel
{
        My_Init_SwitchInterface_Done = true;

        #ifdef __ADSPBF609__
                Init_GPIO_FrontPanelSwitches();
        #endif
}

void My_Init_LEDInterface(void) //This function is initializing the LEDs on the Front
Panel
{
        My_Init_LEDInterface_Done = true;

        #ifdef __ADSPBF609__
                Init_GPIO_FrontPanelLEDS();
        #endif

}

void My_Init_GPIO_REB_Input(void) //This function is initializing the switches on the
REB
{
        My_Init_GPIO_REB_Input_Done = true;

        #ifdef __ADSPBF609__
                        My_Init_GPIO_REB_InputASM();
        #endif
}

void My_Init_GPIO_REB_Output(void) //This function is initializing the LEDs on the
REB
{
        My_Init_GPIO_REB_Output_Done = true;

        #ifdef __ADSPBF609__
                        My_Init_GPIO_REB_OutputASM();
        #endif
}
```

Lab1And0FrontPanelFunctions.h

```
/*
 * Lab1And0FrontPanelFunctions.h
 *
 *  Created on: Nov 21, 2019
 *      Author: aidan
 */

#ifndef LAB1AND0FRONTPANELFUNCTIONS_H_
#define LAB1AND0FRONTPANELFUNCTIONS_H_

#include "Lab2_BF609_Core0_uTTCOSg2017_main.h"

#define GARBAGEVALUE static_cast<unsigned char>(-1)

//Function Prototypes used in Lab 1 for FrontPanel
void myWriteFrontPanelLEDs(unsigned char);
unsigned char myReadFrontPanelSwitches(void);
unsigned char myReadFrontPanelLEDs(void);




#endif /* LAB1AND0FRONTPANELFUNCTIONS_H_ */
```

Idle_WaitForInterrupts_ASM.asm

```
/*
 * Idle_WaitForInterrupts_ASM.asm
 *
 *  Created on: Oct 7, 2018
 *      Author: smithmr
 */

	.section program;
	.global _Idle_WaitForInterrupts_ASM;
_Idle_WaitForInterrupts_ASM:
	nop; nop; nop; nop;  // Stop assembler warning messages

	idle;
_Idle_WaitForInterrupts_ASM.END:
	RTS;
```

ReadCycles_ASM.asm

```
/*
 * ReadCycles_ASM.asm
 *
 *  Created on: Oct 7, 2018
 *      Author: smithmr
 */

       .section program;
       .global _ReadCycles_ASM;
 _ReadCycles_ASM:
       nop; nop; nop; nop;  // Stop assembler warning messages

       R0 = CYCLES;
 _ReadCycles_ASM.END:
       RTS;
```

ReadProcessorCyclesASM.asm

```
/*
 * ReadProcessorCyclesASM.asm
 *
 *  Created on: Sep 26, 2019
 *      Author: aidan
 */
       .section L1_data;

       .section program;
       .global _ReadProcessorCyclesASM;

       #define returnValue_R0 R0
       #define returnValue_R1 R1

_ReadProcessorCyclesASM:
       LINK 20;

       returnValue_R0 = CYCLES;
       returnValue_R1 = CYCLES2;

       UNLINK;

_ReadProcessorCyclesASM.END:
       RTS;
```

My_Init_GPIO_REB_InputASM.asm

```
/*
 * My_Init_GPIO_REB_InputASM.asm
 *
 *  Created on: Oct 12, 2019
 *      Author: aidan
 */
#include <blackfin.h>

        .section L1_data;

        .section program;
        .global _My_Init_GPIO_REB_InputASM;

        #define returnValue_R0 R0
        #define MASK_KEEP_BITS_11_TO_8 0x0f00
        #define SETTING_TO_ALL_ZEROS 0x0000
        #define MASK_KEEP_BITS_15_TO_12_AND_7_TO_0 0xf0ff
        #define SETTING_BITS_11_TO_8_ALL_ONES 0x0f00

_My_Init_GPIO_REB_InputASM:
        LINK 20;

        //This code is storing the value in the port F data register into the pointer
register P0
        P0.L = lo(REG_PORTF_DATA);
        P0.H = hi(REG_PORTF_DATA);

        R1 = SETTING_TO_ALL_ZEROS;
        [P0] = R1; //This is intializing the data register with all zeros to begin
with

        R2 = MASK_KEEP_BITS_15_TO_12_AND_7_TO_0(Z);

        //This code is storing the value in the port F enabled register into the
pointer register P0
        P0.L = lo(REG_PORTF_INEN);
        P0.H = hi(REG_PORTF_INEN);

        R3 = W[P0](Z);
        R0 = R3 & R2; //This is making sure we only zero the bits 11-8
        [P0] = R0;

        R3 = W[P0](Z);

        R1 = SETTING_BITS_11_TO_8_ALL_ONES;

        R0 = R3 | R1; //This putting in the correct enabled values into the enabled
bits part
        [P0] = R0;
	//This code is storing the value in the port F polarity register into the
pointer register P0
	P0.L = lo(REG_PORTF_POL);
	P0.H = hi(REG_PORTF_POL);

	R1 = SETTING_TO_ALL_ZEROS;
	[P0] = R1;

	UNLINK;

_My_Init_GPIO_REB_InputASM.END:
	RTS;
```

My_Init_GPIO_REB_OutputASM.asm

```
/*
 * My_Init_GPIO_REB_OutputASM.asm
 *
 *  Created on: Oct 12, 2019
 *      Author: aidan
 */
#include <blackfin.h>

      .section L1_data;

      .section program;
      .global _My_Init_GPIO_REB_OutputASM;

      #define returnValue_R0 R0
      #define MASK_KEEP_BITS_11_TO_0 0x0fff
      #define MAKING_DIRECTION_ALL_ONES 0xf000

_My_Init_GPIO_REB_OutputASM:
      LINK 20;

      //This code is storing the value in the port F data register into the pointer
register P0
      P0.L = lo(REG_PORTF_DIR);
      P0.H = hi(REG_PORTF_DIR);

      R0 = W[P0](Z);
      R1 = MASK_KEEP_BITS_11_TO_0;
      R2 = MAKING_DIRECTION_ALL_ONES(Z);

      R0 = R0 & R1;
      R0 = R0 | R2;

      [P0] = R0;

      UNLINK;

_My_Init_GPIO_REB_OutputASM.END:
      RTS;
```

My_Read_GPIO_REB_InputASM.asm

```
/*
 * My_Read_GPIO_REB_InputASM.asm
 *
 *  Created on: Oct 12, 2019
 *      Author: aidan
 */
#include <blackfin.h>

	.section L1_data;

	.section program;
	.global _My_Read_GPIO_REB_InputASM;

	#define returnValue_R0 R0
	#define MASK_KEEP_BITS_11_TO_8 0x0f00

_My_Read_GPIO_REB_InputASM:
	LINK 20;

	R1 = MASK_KEEP_BITS_11_TO_8; //Putting the masks into the registers

	//This code is storing the value in the port F register into the pointer
register P0
	P0.L = lo(REG_PORTF_DATA);
	P0.H = hi(REG_PORTF_DATA);

	returnValue_R0 = W[P0](Z); //Putting the value for the switches into the R0
register (this is reading the values)

	returnValue_R0 = returnValue_R0 & R1; //This is selecting only bits 11-8 which
are the input pins
	returnValue_R0 = returnValue_R0 >> 8; //Shifting the 4bit input down to the
bottom to be able to read as a switche value

	UNLINK;

_My_Read_GPIO_REB_InputASM.END:
	RTS;
```

My_Write_GPIO_REB_OutputASM.asm

```
/*
 * My_Write_GPIO_REB_OutputASM.asm
 *
 *  Created on: Oct 11, 2019
 *      Author: aidan
 */

#include <blackfin.h>

	.section L1_data;

	.section program;
	.global _My_Write_GPIO_REB_OutputASM;

	#define returnValue_R0 R0
	#define MaskBitValues11to0 0x0fff

_My_Write_GPIO_REB_OutputASM:
	LINK 20;

	R1 = MaskBitValues11to0; //Storing the mask value into R1 register

	//This code is storing the value in the port F register into the pointer register P0
	P0.L = lo(REG_PORTF_DATA);
	P0.H = hi(REG_PORTF_DATA);

	R2 = W[P0](Z); //Putting the value for the port F register into the R2 register (this is reading the values)

	returnValue_R0 = returnValue_R0 << 12; //Shifting the value for the LEDs up to the output pins

	R2 = R2 & R1; //Masking the port F register

	returnValue_R0 = returnValue_R0 | R2; //Oring the port F register with the correct value for the LED outputs

	[P0] = returnValue_R0; //Storing theses new output values into the port F register so it will display the correct LED orientation

	UNLINK;

_My_Write_GPIO_REB_OutputASM.END:
	RTS;
```

EUNIT FILES

Lab2_BF609EUNIT_Core0_EUNIT2017_main.cpp

```
/*************************************************************************************
*
*   AUTOMATICALLY GENERATED COMMENT -- DO NOT MODIFY
* Author: aidan
* Date: Thu 2019/11/21 at 10:55:00 AM
* File Type: EUNIT Main File
**************************************************************************************
/

#include <EmbeddedUnit2017/EmbeddedUnit2017.h>
#include "Lab2EUNITTesting.h"
void UpdateEunitGui(void);
extern volatile int useLongFileFormat;

extern void AutomatedTestLevelControl(void);

void RestartEunitGui(void);
void UpdateEunitGui(void);

int main(void)
{
      int failureCount;

      RestartEunitGui( );
      UpdateEunitGui();

      UnitTest::ProcessorSpecificStartup();

      AutomatedTestLevelControl();

      UnitTest::Test::GetTestList().ReverseListDirection();

      bool showFail = true;      bool showXFail = true;
      bool showSuccesses = true;

// TODO You can adjust UnitTest::RunAllTests( ) parameters to show only failures --
Wed 2018/09/26 at 08:14:10 PM
// TODO          by setting bool showSuccesses = false;;
      failureCount = UnitTest::RunAllTests(showFail, showXFail, showSuccesses);

      UpdateEunitGui();
      return failureCount;
}
```

Lab2EUNITTesting.cpp

```
/*************************************************************************************
*
*   AUTOMATICALLY GENERATED COMMENT -- DO NOT MODIFY
* Author: aidan
* Date: Thu 2019/11/21 at 10:55:00 AM
* File Type: EUNIT Test File
**************************************************************************************
/

#define EMBEDDEDUNIT_LITE
#include <EmbeddedUnit2017/EmbeddedUnit2017.h>
#include "Lab2EUNITTesting.h"


TEST_CONTROL(Lab2EUNITTesting_cpp);

#if 1
void UpdateEunitGui(void);
TEST(Lab2EUNITTesting_cpp_GUIUpdate) {
        UpdateEunitGui();  // Conditionally compile this line (use #if 0) to stop an
GUI update based on last completed test
}
#endif


unsigned short int TestBitwiseAND(unsigned short int bitPattern, unsigned short int
bitMask);
unsigned short int TestBitwiseOR(unsigned short int bitPattern, unsigned short int
bitMask);

#if 0
TEST(Thread1to3_MoreComplexTest)
{
        #warning 'Dummy test has been inserted -- replace with your own -- Thu
2019/11/21 at 10:55:00 AM '
        // TODO -- 'Dummy test has been inserted  -- replace with your own -- Thu
2019/11/21 at 10:55:00 AM '
        printf("Dummy test has been inserted -- replace with your own -- Thu
2019/11/21 at 10:55:00 AM \n");

        unsigned long int value                 = 0x01FF01FF;
        unsigned long int ORmask                = 0x0F000F0F;
        unsigned long int expectedORResult      = 0x0100010F;
        unsigned long int resultOR   = TestBitwiseOR(value, ORmask);
        CHECK(expectedORResult == resultOR);
        CHECK_EQUAL(expectedORResult, resultOR);

        #error("You insert the 'wrong' test for TestBitwiseAND";
}
```

```
unsigned short int TestBitwiseAND(unsigned short int bitPattern, unsigned short int
bitMask) {
	return bitPattern && bitMask;
}

unsigned short int TestBitwiseOR(unsigned short int bitPattern, unsigned short int
bitMask) {
	return bitPattern || bitMask;
}

TEST(Thread1to3_Successes)
{
	#warning 'Dummy test has been inserted -- replace with your own -- Thu
2019/11/21 at 10:55:00 AM '
	// TODO -- 'Dummy test has been inserted  -- replace with your own -- Thu
2019/11/21 at 10:55:00 AM '
	printf("Dummy test has been inserted -- replace with your own -- Thu
2019/11/21 at 10:55:00 AM \n");

	CHECK(false == false);
	CHECK_EQUAL(false, false);

	XF_CHECK(false == true);    // Expected failure occurs
	XF_CHECK_EQUAL(false, true); // Expected failure occurs
	XF_CHECK(false == false);    // Expected failure does not occur
	XF_CHECK_EQUAL(false, false); // Expected failure does not occur

	#error("You insert the 'wrong' test for TestBitwiseAND";
}

#endif

bool My_Init_LEDInterface_Done = false;
bool My_Init_SwitchInterface_Done = false;
bool My_Init_GPIO_REB_Input_Done = false;
bool My_Init_GPIO_REB_Output_Done = false;
bool My_Init_GPIO_REB_Done = false;

TEST(Thread1to3)
{
	printf("EUNIT Test for Threads 1 to 3 \n");

	My_Init_SwitchInterface();  //This function is initiating the switches on the
panel
	My_Init_LEDInterface();  //This function is initiating the LEDS on the panel

	unsigned char expectedValue = 0x00;
	unsigned char value = 0;

	//Time 0.25 seconds
	//TODO thread 1 check
	//should be off

	frontPanelThread1();
	value = myReadFrontPanelLEDs();
value = value & LED8VALUE;

CHECK_EQUAL(expectedValue, value);

//Time 0.4 seconds
//TODO thread 2 check
//should be off

frontPanelThread2();
value = myReadFrontPanelLEDs();
value = value & LED7VALUE;

expectedValue = 0x00;
CHECK_EQUAL(expectedValue, value);

//Time 0.5 seconds
//TODO thread 1 and 3 check
//thread 1 should be on and thread as a 0
frontPanelThread1();
frontPanelThread3();
value = myReadFrontPanelLEDs();
value = value & LED8AND1AND2VALUE;

expectedValue = 0x80;
CHECK_EQUAL(expectedValue, value);

//Time 0.75 seconds
//TODO thread 1 check
//Thread 1 should be off
frontPanelThread1();
value = myReadFrontPanelLEDs();
value = value & LED8VALUE;

expectedValue = 0x00;
CHECK_EQUAL(expectedValue, value);

//Time 0.8 seconds
//TODO thread 2 check
//Thread 2 should be on
frontPanelThread2();
value = myReadFrontPanelLEDs();
value = value & LED7VALUE;

expectedValue = 0x40;
CHECK_EQUAL(expectedValue, value);

//Time 1 second
//TODO check thread 1 and 3
//Thread 1 should be on and thread 3 should display a 1 with thread 2 still on
frontPanelThread1();
frontPanelThread3();
value = myReadFrontPanelLEDs();
value = value & LED871AND2VALUE;

expectedValue = 0xc1;
```

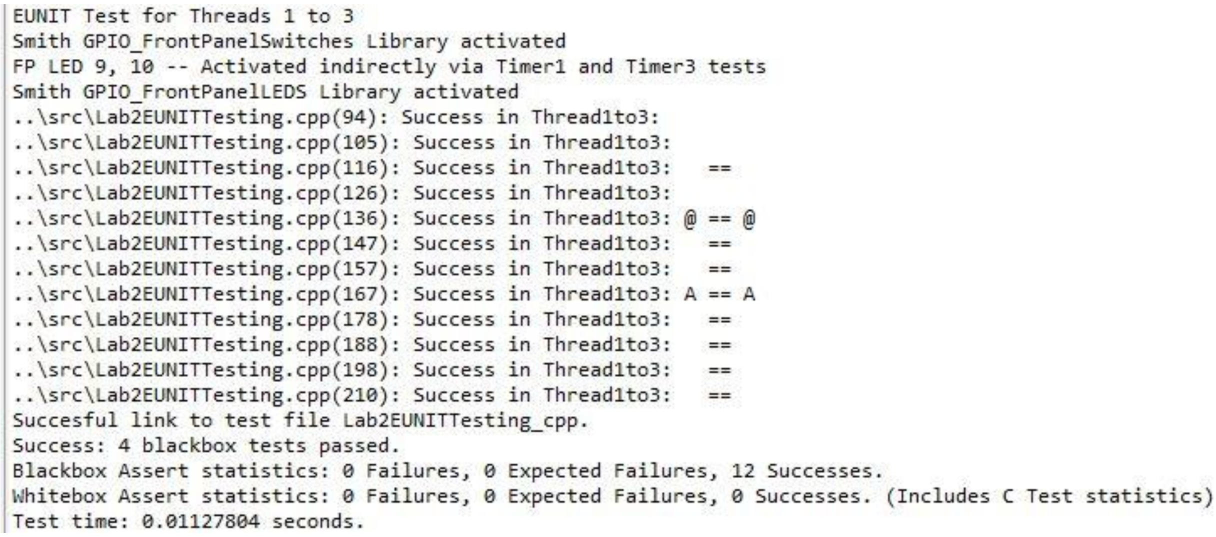
```
CHECK_EQUAL(expectedValue, value);

//Time 1.2 seconds
//TODO check thread 2
//Thread 2 should be on, with thread 1 on and thread 3 displaying a 1
frontPanelThread2();
value = myReadFrontPanelLEDs();
value = value & LED871AND2VALUE;

expectedValue = 0xc1;
CHECK_EQUAL(expectedValue, value);

//Time 1.25 seconds
//TODO check thread 1
//Thread 1 should be off, with thread 2 on and thread 3 displaying a 1
frontPanelThread1();
value = myReadFrontPanelLEDs();
value = value & LED871AND2VALUE;

expectedValue = 0x41;
CHECK_EQUAL(expectedValue, value);

//Time 1.50 seconds
//TODO check thread 1 and 3
//Thread 1 should be on and thread 3 should display a 2, with thread 2 on
frontPanelThread1();
frontPanelThread3();
value = myReadFrontPanelLEDs();
value = value & LED871AND2VALUE;

expectedValue = 0xc2;
CHECK_EQUAL(expectedValue, value);

//Time 1.6 seconds
//TODO check thread 2
//Thread 2 should be off, with thread 1 on and thread 3 displaying a 2
frontPanelThread2();
value = myReadFrontPanelLEDs();
value = value & LED871AND2VALUE;

expectedValue = 0x82;
CHECK_EQUAL(expectedValue, value);

//Time 1.75 seconds
//TODO check thread 1
//Thread 1 should be off, with thread 2 off and thread 3 displaying a 2
frontPanelThread1();
value = myReadFrontPanelLEDs();
value = value & LED871AND2VALUE;

expectedValue = 0x02;
CHECK_EQUAL(expectedValue, value);

//Time 2 seconds
//TODO check thread 1, 2 and 3
```

```
    //Thread 1 should be on, thread 2 should be on and thread 3 should display a 3
    frontPanelThread1();
    frontPanelThread2();
    frontPanelThread3();
    value = myReadFrontPanelLEDs();
    value = value & LED871AND2VALUE;

    expectedValue = 0xc3;
    CHECK_EQUAL(expectedValue, value);

}



TEST_FILE_RUN_NOTIFICATION(Lab2EUNITTesting_cpp);
```

Lab2EUNITTesting.h

```
/************************************************************************************
*
* AUTO-GENERATED COMMENT - DO NOT MODIFY
* Author: aidan
* Date: Thu 2019/11/21 at 10:55:00 AM
* File Type: EUNIT Test Header File
*************************************************************************************
/

#ifndef LAB2EUNITTESTING_H
#define LAB2EUNITTESTING_H

#include "../../ENCM511_SpecificFiles/ENCM511_include/FrontPanel_LED_Switches.h"
#include "Front_Panel_Threads.h"
#include "Lab1And0FrontPanelFunctions.h"
#include "stdio.h"

//These variables are all declared and initialized to be false to begin until the
respective function is called
//to initialize the respective equipment
extern bool My_Init_SwitchInterface_Done;
extern bool My_Init_LEDInterface_Done;
extern bool My_Init_GPIO_REB_Input_Done;
extern bool My_Init_GPIO_REB_Output_Done;
extern bool My_Init_GPIO_REB_Done;

//Extern Function Prototypes
extern "C" unsigned long long int ReadProcessorCyclesASM(void);
extern "C" void My_Write_GPIO_REB_OutputASM(unsigned short int);
extern "C" unsigned short int My_Read_GPIO_REB_InputASM(void);
extern "C" void My_Init_GPIO_REB_InputASM(void);
extern "C" void My_Init_GPIO_REB_OutputASM(void);

//Function Prototypes
void My_Init_SwitchInterface(void);
void My_Init_LEDInterface(void);
void My_Init_GPIO_REB_Input(void);
void My_Init_GPIO_REB_Output(void);

void UpdateEunitGui(void);// Update EUNIT GUI with results from previous test

#endif
```

EUNIT CONSOLE SCREENSHOTS



```
EUNIT Test for Threads 1 to 3
Smith GPIO_FrontPanelSwitches Library activated
FP LED 9, 10 -- Activated indirectly via Timer1 and Timer3 tests
Smith GPIO_FrontPanelLEDS Library activated
..\src\Lab2EUNITTesting.cpp(94): Success in Thread1to3:
..\src\Lab2EUNITTesting.cpp(105): Success in Thread1to3:
..\src\Lab2EUNITTesting.cpp(116): Success in Thread1to3:   ==
..\src\Lab2EUNITTesting.cpp(126): Success in Thread1to3:
..\src\Lab2EUNITTesting.cpp(136): Success in Thread1to3: @ == @
..\src\Lab2EUNITTesting.cpp(147): Success in Thread1to3:   ==
..\src\Lab2EUNITTesting.cpp(157): Success in Thread1to3:   ==
..\src\Lab2EUNITTesting.cpp(167): Success in Thread1to3: A == A
..\src\Lab2EUNITTesting.cpp(178): Success in Thread1to3:   ==
..\src\Lab2EUNITTesting.cpp(188): Success in Thread1to3:   ==
..\src\Lab2EUNITTesting.cpp(198): Success in Thread1to3:   ==
..\src\Lab2EUNITTesting.cpp(210): Success in Thread1to3:   ==
Succesful link to test file Lab2EUNITTesting_cpp.
Success: 4 blackbox tests passed.
Blackbox Assert statistics: 0 Failures, 0 Expected Failures, 12 Successes.
Whitebox Assert statistics: 0 Failures, 0 Expected Failures, 0 Successes. (Includes C Test statistics)
Test time: 0.01127804 seconds.
```