EECS 388

LABORATORY EXERCISE IV

LED

October 22, 2018

Benjamin Streit

GTA: Ibikunle Oluwanisola

LABORATORY OVERVIEW

This laboratory involves the writing, compiling, and demonstrating of a new task on the Tiva TM4C1294 evaluation board with the EECS 388 custom designed BoosterPack. This new task makes use of the 10-segment LED display on the custom BoosterPack to display a variety of patterns.

BACKGROUND

Strong understanding of bit-wise operations and binary arithmetic is necessary for this laboratory exercise. The patterns generated make use of (i) bit-shifting, (ii) bit-wise operator `OR`, and (iii) binary addition. Simple examples of each are listed below:

(i) x = y < < 2;
Assigns x to the result of shifting y to the left by two bits, which is equivalent to a multiplication by four.

(ii) 0101 `OR` 0011 = 0111
Takes two bit patterns of equal length and performs the logical inclusive `OR` operation on each pair of corresponding bits.

(iii) 1010 + 11 = 1101
Addition that carries on a value of 2 rather than 10.

PROCEDURE & RESULTS

Design of the task `Task_LED_Test.c` is in two parts: setting the `LED_InitialState`, and entering the while loop where a given pattern will be generated and written to the LED. Part one and part two vary for each of the patterns. The procedure for said patterns is as follows:

(i) `LED_InitialState` is set to `0x0001`. Then, while `LED_WhichBit` doesn't equal `LED_Bits_Nbr`, `LED_Data` is shifted using < <, and `LED_WhichBit` is incremented. If `LED_WhichBit` does equal `LED_Bits_Nbr`, then `LED_Data` is set back to `LED_InitialState` and `LED_WhichBit` is set back to `0x0`.

This procedure lights up one LED after another across the LED display, and loops.

(ii) LED_Data is set to 0x2AA. Then, while LED_WhichBit doesn't equal LED_Bits_Nbr, LED_Data is set to 0x2AA and LED_WhichBit is set to 0xA. If LED_WhichBit does equal LED_Bits_Nbr, then LED_Data is set to 0x155 and LED_WhichBit is set back to 0x0.

This procedure toggles the LED display between two different states to create the flashing light effect.

(iii) LED_InitialState is set to 0x0001. Then, while LED_WhichBit doesn't equal LED_Bits_Nbr, LED_Data is set to LED_Data | (LED_Data + 0x1), and LED_WhichBit is incremented. If LED_WhichBit does equal LED_Bits_Nbr, then LED_Data is set back to LED_InitialState and LED_WhichBit is set back to 0x0.

This procedure generates the loading bar effect, and loops.

(iv) LED_Data is set to 0x200. Then, while LED_WhichBit doesn't equal LED_Bits_Nbr, LED_Data is shifted using > >, and LED_WhichBit is incremented. If LED_WhichBit does equal LED_Bits_Nbr, then LED_Data is set back to 0x200 and LED_WhichBit is set back to 0x0.

This procedure generates the same effect as pattern (i), but in reverse.

(v) LED_InitialState is set to 0x0001. Then, while LED_WhichBit doesn't equal LED_Bits_Nbr, LED_Data is set to rand(), and LED_WhichBit is incremented. If LED_WhichBit does equal LED_Bits_Nbr, then LED_Data is set back to LED_InitialState and LED_WhichBit is set back to 0x0.

This procedure generates a random pattern, and loops. Note: The rand() function may generate a value greater than the 10-bits the LED display can accommodate, any extra bits will be ignored in this case.

ANALYSIS

The use of bit-wise operations within Task_LED_Test.c allows for quick, efficient, and readable code. An alternative to this approach would be to use the base 10 system, and convert to binary before writing to the LED display. However, this would quickly become rather convoluted as there would be numerous conversions between radices, as well as a significant number of additional lines of code required to generate such patterns using the base 10 system.

Conclusions

The logic written for this laboratory exercise is sensible and effective, as all desired outcomes were achieved and in an appropriate manner. However, improvements could be made. Currently, the various patterns are hard-coded and lines of code must be commented or un-commented to switch between patters. This is workable, but not ideal as it inhibits code modularity, extensibility, and readability. Should this exercise be attempted a second time, these hindrances would be mediated by assigning each pattern to a switch on the evaluation board. This way, a user would only have to press a switch to see a new pattern looping on the LED display.

Code

```
/**************************************************************
* --Task_EECS388_LED_Test.c
*
* Author: Gary J. Minden
* Organization: KU/EECS/EECS 388
* Date: July 27, 2018 (B80727)
*
* Description: Test driver for the LED shift register on the
* EECS 388 BoosterPack.
*
*/

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include "FreeRTOS.h"
#include "task.h"

#include "Drivers/EECS388_LED.h"

```

```
#define LED_Bits_Nbr 10
#define LED_InitialState 0x0001

extern void Task_LED_Test( void *pvParameters ) {

    int32_t LED_Data = LED_InitialState;
    int32_t LED_WhichBit = 0x0;

    //
    // Initialize the EECS_388 LED interface.
    //
    EECS388_LED_Initialization();

    //
    // For Pattern 2
    //
    LED_Data = 0x2AA;

    //
    // For Pattern 4
    //
    LED_Data = 0x200;

    while ( 1 ) {

        //
        // Set the current LED data pattern.
        //
        EECS388_SetLEDs( LED_Data );

        if (LED_WhichBit == LED_Bits_Nbr) {
            //
            // Reset LED_Data
            //
            LED_Data = LED_InitialState;

            //
            // For Pattern 2
            //
            LED_Data = 0x155;

            //
```

```
            // For Pattern 4
            //
            LED_Data = 0x200;

            //
            // Reset Bit Count
            //
            LED_WhichBit = 0x0;
        } else {
            //
            // Pattern 1
            //
            LED_Data = LED_Data << 1;

            //
            // Pattern 2
            //
            LED_Data = 0x2AA;

            //
            // Pattern 3
            //
            LED_Data = LED_Data | (LED_Data + 0x1);

            //
            // Pattern 4
            //
            LED_Data = LED_Data >> 1;


            //
            // Pattern 5
            //
            LED_Data = rand();

            //
            // For Pattern 2
            //
            LED_WhichBit = 0xA;


            //
```

```
            // Increment which bit counter
            //
            LED_WhichBit++;
        }


        //
        // Delay
        //
        vTaskDelay( ( 100 * configTICK_RATE_HZ ) / 1000 );

    }
}
```