ECE 361 Fall 2023
Homework #3b

SINCE WE ARE COMING INTO MIDTERM EXAM WEEKS FOR FALL TERM, I AM RELEASING HOMEWORK #3 IN TWO PARTS WITH TWO DUE DATES. PART B (THIS PART) WILL BE DUE TO CANVAS BY 11:59 PM ON SUN, 05-NOV. NO LATE SUBMISSIONS FOR PART B WILL BE ACCEPTED AFTER 8:00 AM ON WED, 08-NOV. I KNOW THAT MIDTERM SEASON AT PSU CAN BE STRESSFUL, HOPEFULLY THIS HELPS.

THIS ASSIGNMENT (PART B) SHOULD BE SUBMITTED TO CANVAS BY 11:59 PM ON SUN, 05-NOV-2023. NO LATE SUBMISSIONS FOR THIS ASSIGNMENT WILL BE ACCEPTED AFTER 8:00 AM ON WED, 08-NOV.

THE ASSIGNMENT IS WORTH 40 POINTS. IT IS EASIEST TO GRADE, AND I BELIEVE IT IS EASIEST FOR YOU TO SUBMIT, SOURCE CODE FILES AS TEXT FILES INSTEAD OF CUT/PASTE YOUR CODE INTO A `.doc` OR `.docx` FILE. SHORT ANSWER, TRUE/FALSE AND MULTIPLE-CHOICE QUESTIONS SHOULD BE SUBMITTED IN A SINGLE TEXT OR .PDF FILE FOR THE ASSIGNMENT. CLEARLY NAME THE FILES SO THAT WE KNOW WHICH QUESTION THE ANSWER REFERS TO. SUBMIT THE PACKAGE AS A SINGLE .ZIP, .7Z,.TGZ, OR .RAR FILE (EX: `ece361f23_rkravitz_hw3b.zip`) TO YOUR CANVAS HOMEWORK #3 DROPBOX

FOR THIS ASSIGNMENT, YOU WILL BE TURNING IN THESE FILES:

- .C SOURCE CODE FOR PROBLEM #2
- TRANSCRIPT OF YOUR COMMAND LINE SESSION SHOWING THAT YOUR SOLUTION COMPILES AND WORKS CORRECTLY. INCLUDE A MINIMUM OF THE 5 TEST CASES "HARDWIRED" INTO THE STARTER CODE BUT IT WOULD BE GOOD TO ADD A FEW MORE.

NOTE: THERE WAS CODE IN HOMEWORK #1, PROB3_STARTER.C THAT SHOULD DISPLAY YOUR WORKING DIRECTORY. THESE APPLICATIONS SHOULD INCLUDE THE CODE TO DISPLAY A GREETING AND YOUR WORKING DIRECTORY.

ACKNOWLEDGEMENT: THIS PROBLEM IS NEW – CONGRATULATION GUINEA PIGS*! THE IOM361 MEMORY-MAPPED I/O SYSTEM, API AND APPLICATION ARE ALL MINE.*

Problem 2 (45 pts) Embedded systems programming

IMPORTANT NOTE: It is important that you remember Dr Hall's "panic rule" for this problem. There is a lot to absorb to complete this problem. After you get over your initial panic attack read the iom361 documentation and study the iom361 test program and application starter program. Most of what you need to know about using iom361 is demonstrated in the iom361 test example. The rest is a basic application of struct and arrays and the iom361 API.

As promised, here is the first of what I hope will be several embedded systems-related programming assignments. This problem is typical of embedded systems programming – you will read and write peripheral registers in the **I**nput/**O**utput address space of your system. I have crafted an I/O module called `iom361` which simulates a few basic peripheral devices:

- Slide switches – these are an input to your program. A slide switch that is "on" has a value of 1. A slide switch that is "off" has a value of 0. The peripheral register has 1 bit for each switch.
- LEDs – these are an output from your program. An LED is "lit" when it has a value of 1. It is "dark" when it has a value of 0. The peripheral register has 1 bit for each switch. The iom361 module will display the LED values ('o' for lit, '_' for dark) whenever you write a value to the LEDS peripheral register.
- RGB LED – An RGB LED has 3 segments (RED, GREEN, and BLUE). Rather than being driven by a 1 or a 0, each segment is attached to a PWM (**P**ulse **W**idth **M**odulation) circuit. Simply put, you set the brightness of each of the segments to a value of 0 (0%, segment is dark) to 255 (100%, segment is very bright). The iom361 module has 8-bit channels for each of the three segments giving a total of $2^8 \times 2^8 \times 2^8$ (24-bit color) possible colors even though your eyes would not be able to discriminate any near that many colors. The iom361 module displays the duty cycles for each of the segments and whether the PWM outputs to the RGB LED segments are enabled whenever you write a value to the RGB LED peripheral register.

The `iom361` module has other peripheral registers that are not used in this problem. Most notably, `iom361` models an AHT20 Temperature/Humidity sensor (data sheet is included in /docs).

The `iom361` module is provided as iom361.o. I have not made the source code available but the API and register format is documented using my source code comments and Doxygen. **I**ntellectual **P**roperty like `iom361` is often provided as a "black box." You can use it by making API calls, but you do not have details on how it works.

Specification:

Write an application that reads switches and RGB LED duty cycles from `a set of hardwired values` and writes the values to the `iom361` LED and RGB LED peripheral registers. The

iom361 API provides a function named _iom361_setSwitches() that emulates the user setting the switches on "real" hardware.

As a bonus (well, maybe you'd pick a different word) we are going to use the colors API that we developed in class to store the red, green, and blue duty cycle values received to the RGB LED segments. The source code we developed in class and that was contributed by our volunteer teams is included in the starters folder.

Steps:

I recommend the following process to develop this app. This was the way I developed my application. You are welcome to use it or not:

1. Add a function to the colors API that encodes the red, green, and blue variables in the colors_t struct for the RGB LED peripheral register. I defined this function as:

   ```
   uint32_t makeRGBLedReg(color_t s, bool enable)
   ```

   where s is the color_t struct holding the red, green, and blue values (0..255). enable is a bool variable that is true if we want the RGB PWM output driven to the simulated RGB LED. For this problem enable will always be true. The function returns a uint32_t value that can be written to the RGB LED peripheral register. The format of the register is defined in the documentation, and I also reviewed it in class.

2. Initialize iom361 by calling the iom361_initialize() function. Specify 16 switches and 16 LEDs. The iom361_initialize() function returns a pointer to the IO space. The iom361 test program shows how this is done and how to use the base pointer and offset to address the desired peripheral register.
3. Review the "hardwired" test cases in the application so you know what the expected output is. These data will be processed one at a time in your main loop. The main loop in a simple embedded application is where much of the work is done and is often an infinite loop. This application, however, has a fixed number of test cases and should terminate when they are all processed. Many embedded apps make use of interrupts and/or multithreading but they're not needed for this application.
4. Loop on each valid data entry to write the switches to iom361 using _iom361_setSwitches() to put the value to the switches. Read the Switches peripheral register and write the value to the LEDs peripheral register. The iom361 test program has examples of how to do this.
5. In the same main loop use your makeRGBLedReg() function to extract the red, green, and blue values from your color_t variable and encode the values for the iom361 RGB LED peripheral register. Write the encoded value to the RGB LED peripheral register.
6. Repeat your main loop after a delay of a few seconds (use the sleep() function in windows.h or unistd.h for Linux (and hopefully MAC). The sleep() function suspends the process your program is running in for sleep(seconds). Waiting around for something

"interesting" to happen is standard fare for embedded apps that do work at "human speeds" (glacial in comparison to the speed of your CPU and peripherals).

a. (35 pts) Write a program that meets the specification. A "skeleton" application template is included in the starters folder. Make sure to claim the source code as your own; at this point the code is yours, not mine.

b. (10 pts) Compile, debug, and execute your application using the command line. Include at least 10 words to show that your application works correctly. Submit your source code, in including changes to `color.h` and `colors.c` and a transcript showing that your application runs successfully