Introduction

For this third assignment, we will finally get to play with our FRDM-KL25Z. This assignment will focus on writing to the GPIO pins to control the tri-color LED on the Freedom board, as well as reading from the touch sensor register.

As before, you will develop a proper GitHub project for this assignment, complete with code, documentation, and README. The assignment is due on September 29 at 9:00 am Mountain time, and should be submitted on Canvas. Your submission will consist of two parts:

- Part 1 is a single private GitHub repo URL which will be accessed by SAs to review your code and documentation. For this assignment, this repo will have a README, multiple .c and .h files, test cases, and a Makefile. Details are below.

- Part 2 will be a PDF containing all C code and README documentation. As before, the PDF is used specifically for plagiarism checks: Your code should be yours alone, developed by you. You should provide a URL for any significant code taken from a third-party source, and you should not take code artifacts from other students from this or prior semesters.

You may consult with other students, the SAs, and the instructor in reviewing concepts and crafting solutions to problems (and may wish to credit them in documentation).

You may now use printf() and other standard C functions in your code.

Background

A computer lab I once worked in had the following sign posted in the window:

Achtung!

Das Machine ist nicht fur gefingerpoken und mittengrappen. Ist easy schnappen der Springenwerk, blowenfusen und poppencorken mit Spitzensparken. Ist nicht fur gewerken by das Dumköpfen! Das rubbernecken Sightseeren keepen hands in der Pockets, relaxen und watch der Blinkenlights.

In this assignment, we are going to give “Das rubbernecken Sightseeren” something to play with.

Assignment Details

Your Blinkenlights program will be a bare-metal C executable that runs on the KL25Z. When fully complete, your code will exhibit the following functionality:

1) At program startup, Blinkenlights will "test" the LED by blinking in the following pattern:

    RED for 500 msec, OFF for 100 msec,
    GREEN for 500 msec, OFF for 100 msec,
    BLUE for 500 msec, OFF for 100 msec
    WHITE (that is, RED, GREEN, and BLUE all on) for 100 msec, OFF for 100 msec
    WHITE for 100 msec, OFF for 100 MSEC

2) Blinkenlights will then enter an infinite loop where the LED will be flashed using the following pattern:

    ON for 500 msec, OFF for 500 msec,
    ON for 1000 msec, OFF for 500 msec,
    ON for 2000 msec, OFF for 500 msec,
    ON for 3000 msec, OFF for 500 msec
    Go back to the top (e.g., ON for 500 msec…)

3) During the infinite loop, the color when the LED is ON will initially be white. If the user touches the capacitive touch slider, the color will change as follows:

User touches	Color
Left side of slider	Red
Center of slider	Green
Right side of slider	Blue

   (There is no way to get back to white after touching the slider.)

   The Blinkenlights code should poll the touch slider once every 100 msec. If the LED is on when a touch is detected, the light color should change immediately.

4) Your code will have two build targets, DEBUG and RUN. The LED and touch behavior will be the same under both targets; however, the DEBUG target will additionally cause the following debug output (via printf) to a serial console:

    - CHANGE LED TO RED, etc. for changes to the LED color
    - SLIDER VALUE 87 for changes to the slider value including showing the value read[1]
    - START TIMER 2000 for starting a 2000 mSec delay

[1] No need to print a message if the slider is not currently sensing a touch event; otherwise, your debug output will be filled with no-touch events!

This project is intended to be a bare metal implementation, so you may not use advanced SDK or RTOS routines. You may use include files generated from MCUXpresso tools such as pin_mux.h. You may also use “board.h” and “MKL25Z4.h”. You may not use the “fsl_” include files that provide higher-level SDK-based functions for LED or slider control, with the exception that you *may* use “fsl_debug_console.h”.

Implementation Notes

We will not use interrupts for this assignment. Reading the touch slider will be done via periodic polling.

For timing delays, you should use a relatively simple “hard spin” loop similar to the following:

```
while (ctr-- != 0) {
      __asm volatile("NOP");
}
```

You will need to empirically determine an approximate adjustment to convert from milliseconds of delay into number of counter loops.

For serial output, you may use either the “semihost” solution from MCUXpresso, or you may direct the output over a UART and display that in a separate window. Note that we will *not* use the semihost debugging solution after this assignment, because it requires a debugger to be present and also has the potential to interfere with other interrupts on the KL25Z. For that reason, there is some value to taking the time now to get your debugging working over the UART.

Your code should follow the ESE C Style Guide (posted on Canvas) as closely as possible.

As with all code you ever write, I recommend that you simplify the program to its most basic possible behavior, and then slowly accrete additional functionality. Get each level of functionality fully working, with no bugs, and save it aside[2], before progressing to the next step. As an example, when I developed my own version of Blinkenlights, I followed a strategy similar to the following:

- Get the program to run on the KL25Z and send a message (“Hello World” is fine) back across the serial port.
- Figure out how to turn on the red LED, and then turn it off
- Expand your LED implementation so that it can turn the LED on for an arbitrary color.
- Implement the LED test sequence in step 1 of the Assignment Details. This will require creating a delay() function. Play with this sequence in order to determine the conversion factor between msec and counter loops.
- Implement the LED blinking sequence from step 2 of the Assignment Details. You probably want to make this code data-driven in some manner—in other words, put the specifics of the timing into an array or other data structure.

[2] A local git repository serves this purpose nicely.

- Figure out how to read the touch slider, using print statements liberally. There is a calibration needed to map between the value read from the TSI0->DATA register and the left/center/right position of the touch. As with the delay loop, approximate values are fine.
- Add the polling of the touch slider to the natural loop cadence in your LED blinking sequence.

At this point, you have the system *almost* done. The only problem will be that there is too much latency between pressing the slider and seeing the LED color change. The final steps:

- Figure out how to modify your main loop so that you are polling the touch slider every 100 msec, and changing the light color immediately.
- Finally, add the RUN build target. Using #ifdefs and #defines, modify your debug output so that it only appears under the DEBUG target.

Code References

A good example of bare metal LED control can be found in the Dean book in Chapter 2. The example LED code from the book is available at https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter_2/Source/main.c

Information about accessing the touch slider is also available in Dr. Dean's GitHub account, under https://github.com/alexander-g-dean/ESF/tree/master/NXP/Misc/Touch%20Sense.

Remember that you must be able to explain how all the code in your project works, so be sure you both cite the source of and can entirely explain any small amounts of Dr. Dean's code that you choose to use.

Development Environment

For this project, you should use the MCUXpresso IDE.

Grading

Points will be awarded as follows:

- 20 points for overall elegance: Is your code professional, well documented, easy to follow, efficient, and elegant? Does it compile without warnings? Does it follow the ESE style guide? Is code needlessly duplicated?
- 10 points for the initial LED test sequence from step 1 of the Assignment Details, with approximately correct timing
- 20 points for the main LED timing loop described in step 2 of the Assignment Details, with approximately correct timing.
- 20 points for reading the touch sensor and changing the color of the LED (with any amount of latency)
- 20 points for reading the touch sensor and effecting changes to the LED color within 100 msec

- 10 points for having the DEBUG and RUN build targets, and appropriate serial output in the DEBUG case (note you need to have both targets to get the 10 points; just having one or the other results in no points).

Extra Credit

After you have finished the development portion of the assignment, you may wish to answer the following questions for extra credit. Please add the answers to a section of your README with the heading "Extra Credit".

- What is the address of your main() function? [+1 point]
- What is the size in bytes of your delay() function, as shown by objdump? [+1 point]
- Show the full disassembly of your delay() function, adding comments to each line to explain the functionality. [+8 points]

All the above information is available via the objdump command[3].

You are only eligible for extra credit if you get at least 70 points on the main assignment. I make this rule because I want you to focus on eating dinner before dessert: You should only attempt the extra credit after you have the Blinkenlights functionality working.

Good luck and happy coding!

[3] You might need to poke around your system to find the cross-compilation version of objdump. On Mac, mine is located at `/Applications/MCUXpressoIDE_11.2.0_4120/ide/tools/bin/arm-none-eabi-objdump`. Note the executable name on my system (and likely on yours) is "arm-none-eabi-objdump"; this is done by MCUXpresso in order to support cross-platform development.