Introduction

For this third assignment, we will finally get to play with our Nucleo-64 board. This assignment will focus on writing to the GPIO pins to control the User LED(LD2) and an external LED from the Nucleo-64 board, as well as reading from the user switch.

You will be joining a GitHub classroom with the link posted on Canvas, and use the repository created in this manner for the assignment, complete with code, documentation, and README. Details on submissions are a bit different than we've done to date and are given below. The assignment is due on Wednesday, October 2, 9:00 AM although there is an earlier deadline for September 30th readiness for code reviews (see below).

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. For your printfs, you terminal should be set to 9600 baud rate, 1 stop bit and no parity. You can use the terminal from within the STM32CubeIDE or an external program such as Putty or Tera Term.

Assignment Details

Your DancingLights program will be a bare-metal C executable that runs on the Nucleo-64. The board has User Led(ULED) on the board itself. We will also connect and external led(ELED) to the **PB3** pin on the board by using a breadboard and jumper cables which you can check out with the board. Make sure that the external LED **cathode** is connected to PB3 and **anode** is connected to the 3.3V. Doing so, writing a 0 to PB3 will light up the external LED. When fully complete, your code will exhibit the following functionality:

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

   ULED ON for 2000 msec, OFF for 1000 msec,
   ELED ON for 2000 msec, OFF for 1000 msec,
   ULED ON for 2000 msec, OFF for 1000 msec,

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

   ELED ON for 1000 msec then OFF for 500 msec. ULED is OFF during this sequence.

3) In Step 2, when the if the user presses the User Switch(LD2 on board), the blinking will change to following and continue in an infinite loop:

   ULED ON for 3000 msec then OFF for 1500 msec. ELED is OFF during this sequence.

   The User Switch must be pressed with a gentle touch and released quickly **otherwise it may register double clicks**.

4) If the board is running a Step 3 sequence, a switch press will put it in Step 2 sequence and when the board is running Step 2 sequence, a switch press will put it in Step 3 sequence.

   The DancingLights code should poll the User Switch at least once every 100 msec during Step 2 and Step 3 sequence above. The blinking should change immediately to Step 2 or 3 sequence as applicable when the switch press is detected.

5) Your code will have two build targets, DEBUG and RELEASE. The LED and User Switch 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 ULED(or ELED).
   - START TIMER 1000 for starting a 1000 msec delay.

The debug output should occur for every LED change and for every delay start.

The required timing for this project should be in the RELEASE build. The DEBUG will add extra delays due to printfs etc. and that is acceptable. Also, any sizes or starting addresses mentioned in the rubric below, should also pertain to RELEASE build. Also, confirm that you are running at zero optimization level(-O0) for timing as well as reporting addresses/sizes below.

This project is intended to be a bare metal implementation, so you may not use advanced SDK or RTOS routines or HAL functions. In fact, your application should be built on top of the github code provided which only supports output to the console with printfs etc. at 9600 baud rate, 1 stop bit and no parity.

When the board is reset, you may see momentary flash on both LEDs and that is acceptable. It happens because the GPIOs when initialized may be in an unknown state causing the LEDs to flash.

Implementation Notes

We will not use interrupts for this assignment. Reading the user switch 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 STM32CubeIDE, 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 has the potential to interfere with other interrupts on Nucleo-64. 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 add additional functionality. Get each level of functionality fully working, with no bugs, and save it aside[1], before progressing to the next step. As an example, when I developed my own version of DancingLights, I followed a strategy similar to the following:

- Get the program to run on the Nucleo-64 board and send a message (“Hello World” is fine) back across the serial port.

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

- Figure out how to turn on the user LED, and then turn it off
- Expand your LED implementation so that it can turn the LED.
- Implement the LED test sequence in step 1 of the Assignment Details. This will require creating a delay() function. Play with this sequence 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, you should put the specifics of the timing into an array or other data structure.
- Figure out how to read the User Switch, using print statements liberally.
- Add the polling of the User Switch 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 switch and seeing the LED color change. The final steps:

- Figure out how to modify your main loop so that you are polling the User Switch every 100 msec, and changing the light color immediately.
- Finally, add the RELEASE 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/

Information about accessing the user led is also available in Dean's GitHub account in the same location.

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 Dean's code that you choose to use.

Development Environment

For this project, you should use the STM32CubeIDE for development and debugging. The current version I have used is 1.15.1 and that will be used for grading as well.

Peer Code Review, and Submission Schedule

Your primary submission is due a bit earlier than usual: on Monday, September 30, at 9:00 am. At that time, you should submit the following:

- For this assignment, this repo should have an STM32CubeIDE project containing one or more .c and .h files.

  New for this assignment: You must tag your repo at this time with the tag "**ready-for-review**"

We will randomly assign you to a partner in the class, and we will announce assignments prior to 8 pm on Sunday, September 29. Between 9:00 am Monday and the start of class on Tuesday, you will need to spend time familiarizing yourself with your partner's code. Class time on Tuesday (October 1) will be spent performing peer code reviews; during this time, you will each be given about 20 minutes to review the other's code. Your participation grade for this class will be based on how prepared, insightful, and tactful you are in critiquing your partner's code. You will need to capture the Review comments (at least 5) and your response in the Readme.md file in your repository.

To be clear, your code should meet all the project requirements by the Monday 9:00 am deadline. The goal of the code review is to help you find stray bugs and improve the elegance and readability of your code.

After the code review, you should plan to make additional check-ins to your GitHub repo to address the feedback received during code review. These modifications are due by 9:00 am on Wednesday, October 2
at which point, you should tag your code with "**ready-for-grading**".

Note: The peer review process adds a hard deadline to the initial Monday due date. If your code is not available at this point, you are preventing your partner from reviewing your code, which will hurt their grade as well as yours. It is therefore especially important that your code is completed on time.

Grading

Points will be awarded as follows:

Points	Item
15	Overall elegance: Is your code professional, well documented, easy to follow, efficient, and elegant? Is code needlessly duplicated?
15	Initial LED test sequence from step 1 of the Assignment Details, with approximately correct timing
15	Main LED timing loop described in step 2(7.5) and step 3(7.5) of the Assignment Details, with approximately correct timing

15	Reading the user switch and changing the LED blinking from internal to external(8) or vice versa.(7)
10	Reading the switch and effecting changes to the LED color within 100 msec. Here the timing of change should be immediate.(All or nothing if the timing appears to be slow).
10	Quality of your code review and review feedback
10	Having the DEBUG target with appropriate serial output(5) and RELEASE(5) build targets.
10	Record in your README **address** of main function(2), **size** of delay function(2), **size** of text segment(2), **size** of data segment(2), initial **size** of stack allocated at program start(2) for RELEASE configuration with -O0 optimization level. The sizes and addresses are as reported by ARM version of objdump since the reporting varies by the tool.

As mentioned above, in addition to the assignment grading, you will earn participation points on October 1 based on your preparation, insight, and tactfulness in code review.

Good luck and happy coding!