ELEC 335 - Lab #4

Objective

The objective of this lab is to

- practice C language, and write basic programs,
- work with interrupts and utilize external interrupts,
- work with various timer peripherals such as SysTick, general purpose timers and watchdog timers,
- work with the debugging tools for analyzing program flow.

Setup

Install STM32CubeIDE and implement problems on the breadboard. For that, use the **`blinky`** project from the g0 project repo (https://github.com/fcayci/stm32g0) and follow https://youtu.be/UYk_NAnDJlw. Do not use **standalone** for any problem / lab. That is just for demonstration purposes. You can use one of the undergraduate laboratories for using the scope and measuring the necessary voltages if needed.

- **For each coding problem**, you are required to create a flowchart that shows software operation and a block/connection diagram that shows how things are connected and what modules are in use. You can use online tools to do that. One example is lucidchart (https://lucid.app)
- **All codes should be written in C.**
- If a specific pin number is not stated, you are free to pick any appropriate pin to connect your components. Make sure to include this in your block/connection diagram.

Submission

You should submit the following items organized in a folder:

- **Lab report** - Written in English. PDF file. Should include
  - a cover page
  - for each problem
    - a flow chart
    - a block/connection diagram
    - pictures/photos if any/requested
    - code listing
    - at least one paragraph explanation/comment about that problem, code listing
  - final lab conclusion about what you've learned.
- **Source files** - should have proper comments and pin connections if there are any external components.
- **Elf files** - generated elf file with debug information.

Problems

Problem 1 [20 pts]. Create a Board Support Package (BSP) that you will be using for the rest of the semester. This BSP should only consist of board related functions. Some of these functions include:

- Configure / turn on / turn off / toggle on-board LED.
- Configure / read on-board button.
- Initialize and configure the processor clock.
- Initialize and configure the interrupts / exceptions.
- Initialize and configure the SysTick timer. (Problem 2)
- Initialize and configure the watchdog timer. (Problem 4)
- Initialize and configure the timers if any.
- Initialize and configure the external interrupts.

Problem 2 [15 pts]. In this problem, you will work on creating an accurate delay function using the **SysTick** exception. Create a SysTick exception with 1 millisecond interrupt intervals. Then create a `delay_ms(..)` function that will accurately wait for (blocking) a given number of milliseconds.

- Demonstrate the operation using an oscilloscope.
- Compare this approach to the without timer approach, explain the differences.

Problem 3 [45 pts]. In this problem you are asked to implement a time counter using SSDs. Attach 4x SSDs and using **a state machine**, implement a time counter with different intervals. Assign each speed a mode and attach a button to cycle through the modes. (Each button press will cycle through these modes.)

- You should use a timer interrupt to display a given number on an SSD.
- Another timer should do the counting.
- This is the final implementation on how you should use the Seven Segment Display.

Modes:

- Mode 0 → No countdown (0 second interval)
- Mode 1 → Count down speed is $\times 1$, with 1 second intervals.
- Mode 2 → Count down speed is $\times 2$, with .5 second intervals
- Mode 3 → Count down speed is $\times 10$, with .1 second intervals
- Mode 4 → Count down speed is $\times 100$, with .01 second intervals
- Mode 5 → Count down speed is $\times 1000$, with .001 second intervals

Warnings:

- Define an enum for your states, and cycle through them in each button press.
- Depending on the state, define the delay.
- Your flowchart should be detailed enough and should overlap with your implementation.
- Make sure your code works when the optimization is defined as `-O2`.

Requirements:

- Brightness on the Seven Segments should be equal.
- No bouncing on the buttons.
- Start from `00:00`, `23:59`, `23:58`, ... and go down to `00:00` and keep counting again (free running time counter). The four SSDs must show the countdown as "**minute:second**" format without colon (**:**). For Mode 1, counting down must take 24 minutes to complete. For the other modes it should take less than 24 minutes, inversely proportional to the speed.

Questions:

- What is the difference in code size when the optimization is enabled / disabled? How about the actual counter speed? Is there any change? If so, what would be the difference?
- What is the code size percentage to the available ROM / RAM? How does optimization change this percentage?
- Is / Was there any brightness difference between the Seven Segments? If so, how did you fix it?
- Explain the difference between your implementation from the last lab? Which one is more convenient and scalable?

Problem 4 [20 pts]. In this problem, you will work with watchdog timers. Take Problem 3 as base, and implement a window or independent watchdog timer with appropriate delay. The handler routine should restore the stack pointer and reset back the state of the program to the beginning.

- How would you work with watchdog timers when there is nothing on the main routine?
- Create an external interrupt with a button that will turn off all the timer interrupts. (Not the exceptions and watchdog timer) This should fire up the watchdog timer since it will not be fed.