EENG260 (Microcontrollers) Lab 5:

Build a 3-digit decimal display up/down timer controlled by a Tiva C evaluation board using interrupts, C, and the TivaWare library

Tasks for this lab

- Connect the provided 7-segment display hardware to your microcontroller evaluation board by plugging its header into the J1/J3 socket, such that the red wire connects to the VBUS pin.
    - VBUS (+5V from your computer's USB port) and GND supply power for the display module circuitry.
    - Port D pins 0-3 connect to the "GPIO Data" lines on the Display Module schematic.
    - Port E pins 1-3 connect to the "GPIO Control" lines on the Display Module schematic.
- Create a program in C that meets the following requirements:
    - The display module will show the current value of a count that starts at 0, without unnecessary leading zeroes, maintained in a uint8_t variable.
        - Use a general-purpose timer with an interrupt to switch between digits to produce a steady display.
        - I *strongly* suggest separating out your display driver code into its own .c/.h files, as you will be using this display again as part of your final project.
    - Once per second, the value of the count will change by 1.
        - Use a general-purpose timer with an interrupt to make the count change every second.
        - At startup, the count will be increasing by 1 each second.
        - Pressing SW1 on the Tiva C board will toggle between increasing and decreasing (use SysTick with an interrupt to handle switch debounce).

Reference Materials

- CortexM_InstructionSet.pdf
- Tiva TM4C123GH6PM Microcontroller spms376e.pdf
- Tiva C Series TM4C123G LaunchPad spmu296.pdf
- Display Module.pdf
- TivaWare_C_Series-2.2.0.295.zip
- TivaWare Peripheral Driver Library spmu298e.pdf

An overview of the display module

Ideally, when we want to add peripherals to a microcontroller, we want to drive them directly from the microcontroller to the greatest extent possible, since this gives us greater flexibility without additional cost. However, sometimes this simply isn't possible. The seven-segment displays on the display module are a prime example of this problem. Individually, each segment on the display draws at least 10mA of current, while the most we can supply from any of our microcontroller pins is 8mA, and that only for a limited number of pins. We need some sort of driver circuit for our microcontroller pins to control, to provide the necessary current without overloading our microcontroller. One simple option is the 74LS47, a driver IC specifically designed for driving common-anode seven-segment LED displays. Of course, it needs a 5V power supply, while our microcontroller takes 3.3V, but we have 5V available from the USB bus (it's labeled VBUS on the Tiva C board and schematic, as well as the display module schematic), and the high logic level voltage put out by the GPIO ports is high enough to be recognized by the inputs of the 74LS47, so it will all work together without issues.

In addition to power and ground, this module has four "GPIO Data" lines, which are used to provide the binary-coded decimal (BCD) number that needs to be displayed (connected to Port D pins 0-3), and three "GPIO Control" lines that turn on individual digits when a 1 is applied to them (connected to Port E pins 1-3, with pin 1 -> Ones, pin 2 -> Tens, pin 3 -> Hundreds). This setup will allow you to create a 3-digit decimal display, and even stay within the power limits of the Tiva C board if all the display segments are turned on at once, but the trade-off is that there is no way to display hex digits A-F or a negative sign. For our purposes, we can do without those features, so we will use this design.

To display a multi-digit number with this module, all you need to do is set up each digit you want to display on the GPIO Data lines with the appropriate GPIO Control line held high, hold it for a while, then switch to the next digit and repeat. If you hold each digit for a full second, you can watch the digits change at a regular rate (handy for verifying that your code does what you expect). Reduce the hold time enough, and you can produce the illusion that all the digits are lit (if somewhat more dimly than full held-on brightness). If all you needed to do was drive digits to display a number, you could do that using the sort of busy-wait programming we've been using up to now, but you will have multiple timed tasks occurring at the same time in this project, so you will be better served by using the microcontroller's onboard timers and interrupts.

An overview of the timers

You have already used the SysTick timer before in a previous lab (and will again in this lab!), so you have some familiarity with the basic concepts involved already – give the timer a clock to drive it, a value to countdown from, and wait for the countdown to reach zero before continuing on. However, the general-purpose timers (or just "Timer" in the TivaWare library manual) on our microcontroller can be a bit more complex, due to all of the different use-cases the manufacturer envisioned that a programmer might need it for.

Simply put, don't get distracted by all of the options and features that show up in the TivaWare library. You will need two timers (in addition to the SysTick timer), and the microcontroller has six, so there's no need to worry about setting up half-width timers. You just need them as simple timers, so no need to worry about features like ADC, PWM, synchronization, or anything like that. Just configure two timers to use the system clock, then set them up as periodic timers with the value you want them to count down from, and, as each timer reaches zero, it can automatically trigger an interrupt to handle its timed task.

An overview of interrupts

Programming with interrupts is a bit different from the types of programming you have done up to this point. Instead of having to set up a loop to obsessively poll your inputs and watch for changes, you can let the hardware do the work for you and let you know when something needs your attention, leaving you free to use those clock cycles for more important things (or even slow down your clock to make the microcontroller “sleep” while you are waiting!). On microcontrollers that support the feature, interrupts are a powerful tool.

Our microcontroller has excellent support for interrupts, courtesy of the nested-vector interrupt controller (NVIC), with multiple priority levels and dozens of interrupts that can be used. Again, don’t get distracted by all the various options and features. There will be slight differences from one peripheral to the next, but, generally, your initial setup for each will be to register an interrupt handler (a function with <u>void for parameters and return type</u> that you write) that you want to be called when your interrupt condition is met, configure for any options you want to use (like, say, triggering only on the falling edge when your switch is pushed), then enable the interrupt. Inside of your interrupt handler, you will also need to “clear” the interrupt as part of your handler code, so the microcontroller knows that you are dealing with it and doesn’t send you right back to the start of your handler as soon as your current call is finished (or produce other, harder-to-troubleshoot behavior like suppressing lower-priority interrupts). You might even “chain” your interrupts together to meet more complex needs (like, say, when the switch’s interrupt is triggered, its handler enables the interrupt on a timer for the switch debounce, then that timer’s interrupt handler re-enables the switch interrupt, along with whatever else it does).

One more thing: like I underlined previously, your interrupt handlers can take no parameters, and will return nothing. Simply put, since they’re being triggered by the hardware, there isn’t any opportunity to set up anything to provide input or receive results at the time the handler is called. That said, there is only one way to get data into or out of an interrupt handler that the rest of the system can make use of, and that’s global variables. The drawbacks of global variables should have been covered in your prior C course, so no need to go into great detail here. Simply put, use global variables as and when you must, but don’t rely on them where there are other options available.

Where to start

I suggest focusing on the display module to begin with, to work out the details of timer configuration and setting up interrupts and handlers (start with a constant count to test your display and work out a suitable refresh rate for the seven-segment displays). Once you have that much, adding an incrementing count should be easy, and you will have less trouble setting up the count direction toggle thereafter.

To turn in

Make a .zip file with the following contents and upload it to Canvas:

- A brief lab report, in Word or PDF format. What issues did you run into? What did you learn? Keep it under a page, if you can.
- A copy of your main.c file. You can export your file to the file system by right-clicking on it in the Project Explorer pane and selecting Export.
- Any other .c and .h files you added to your project (if any)