Final ECE2560 Spring 2022

Collaboration with other students is not allowed

Use the word template and instructions contained on our web site to submit your work to Carmen. Do not email directly to your TA or me. Files emailed to the TA or me will not be accepted.

In this program we will use interrupts generated by a Timer on your microcontroller. Timers were introduced in Screencast22. Based on that screencast you configure TimerB0 on your MCU by using the following code in your main program.

```
; Configer timer B0, connected to ACLK, to raise interrupts
  bis.w #TBCLR, &TB0CTL
  bis.w #TBSSEL__ACLK, &TB0CTL
 bis.w #MC__CONTINUOUS, &TB0CTL
 bis.w #TBIE, &TB0CTL
 bic.w #TBIFG, &TB0CTL
```

This code will configure TimerB0 and the timer will start generating interrupts periodically. You will handle these interrupts in an Interrupt Service Routine (ISR). Call this ISR Timer_Overflow just as we did in Screencast22. You will need to figure out the Interrupt Vector associated with timer interrupts before you can use this ISR. Screencast22 at 29:11 shows you what Interrupt Vector to use so that your ISR will be invoked every time the timer generates an interrupt.

We will also use interrupts generated by button SW1. The aim of our program is to implement a **counter** which counts the number of timer generated interrupts between two presses of the button SW1. Pressing button SW1 once will start our counter, and it will start counting the interrupts being generated by the timer. Pressing SW1 again will stop the timer and the number of timer interrupts between the time the counter was started and the time when the counter was stopped will be written to a variable named result. Pushing SW1 again will start the counter again.

Use the following variables in the .data region

```
count:          .space 2                            ; count of timer
                                    ; interupts since the program started

count0:         .space 2                            ; Value of count
                                  ; variable at the time counter is started

count1:         .space 2                            ; Value of count
                                  ; variable at the time counter is stopped

result:         .space 2        ; result = count1 - count0

cstatus:        .space 1                            ; Zeroth bit of
                                ; cstaus stores your counter's status bit.
                       ; 0 means counter stopped, 1 means counter running
```

Note that by "Timer" we mean TimerB0 on yoour MCU which is generating the timer interrupts. By "counter" we mean the counter you are implementing which counts the number of interrupts being enerated by the Timer.

Timer ISR (name it Timer_Overflow) Should increase the value of the count variable every time it is invoked. It should also clear the interrupt flags (see Screencast22) since this interrupt is multisourced. These are the only responsibilities of this ISR.

The ISR servicing SW1 (name it PORT1_ISR) should do the rest, its functionality should be such that

i) When it is invoked and your counter is in the stopped state then it should turn the counter to be in the running state, and if it is in the running state then it should put it in the stopped state, while also handling the variables count0, count1 and result appropriately based on the table above. The ISR should handle **toggling the 0th bit of cstatus**. It should not change the value of the count variable.
ii) The red LED should indicate whether your counter is running or stopped. When the 0th bit of cstatus is equal to 1 then the red LED on your launchpad should turn on. When the 0th bit of cstatus is 0 then the red LED should be turned off.

Start the program with the red LED in the turned off state and the 0th bit of scount set to 0 in your main program.

Important Note: It turns out that for some reason a SW1 interrupt is automatically generated by the system at program startup even though you have not pressed the button SW1 yourself. This means that at program start up, the ISR for SW1 will already have been invoked once. Keep this in mind. For example, if in your main program you put the counter in the stopped state (and you toggle the counter's state in your SW1 ISR), and you start your program, the ISR would already have been invoked once and your counter would be in a running state at startup even though you have not pressed SW1. This should not pose a problem if your program is written correctly, just press the button again to put the counter in the stopped state. This happens only once at program startup.

Low Power Mode: Put your MCU is LPM3 Low Power Mode while it is waiting for interrupts.

Attach the following to your solution:

i) Assembly language source code (Word format only). Make sure to attach every line of the source code, with nothing missing so that we can run the code on our computers

Ii) Run your program. As mentioned above, the program will start with SW1 interrupt service routine already invoked once, so your counter will be in the running state (if your program is written correctly). Press SW1 to put the counter in the stopped state. Now wait about 15 seconds (this time is not critical).

Iii) Press SW1 again (to put the counter in running state)

Iv) Wait about 10 seconds and then press SW1 to stop the counter.

v) Press the suspend/pause button in the debugger
vi) While your program is suspended go to the .data region of your memory browser and take a screenshot of the memory browser showing the values of the variables count, count0, count1, result and scount.
vii) Include this screenshot in your solution