ENCM 511 Assignment1

Preamble

A computer screen with text and numbers

AI-generated content may be incorrect.

The above definitions have been added at the start of the code to improve readability. Their corresponding pin mapping may be referenced in the image above. LED2 is unused in this lab and is commented out for that reason. Global ISR priority and debounce time variables have also been created to allow for easy adjustment.

A computer screen with text and images

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.Peripherals

A computer screen with text on it

AI-generated content may be incorrect.Above is the IO\_init, timer\_init, and delay\_init functions to be called at the beginning of main to configure IO pins and timers on the microcontroller. Line functions are elucidated by their comments.

Datatypes and Initialization

A computer screen with green text

AI-generated content may be incorrect.

The state variable will be in the range of 0 to 3 and thus is an unsigned 8bit integer.

The pb\_event variable will hold 1 or 0 and thus is an unsigned 8 bit integer.

The bp2\_last variable will hold 1 or 0 and thus is an unsigned 8 bit integer.

The LED0\_last variable will hold 1 or 0 and thus is an unsigned 8 bit integer.

The LED1\_last variable will hold 1 or 0 and thus is an unsigned 8 bit integer.

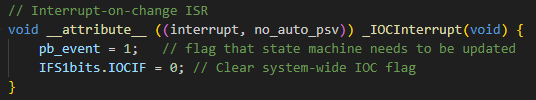
Operation

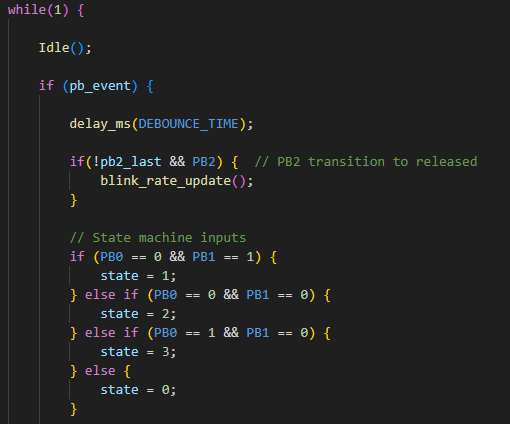
The code begins by calling all three initialization functions before entering its while loop and going into idle.

A black rectangle with white text

AI-generated content may be incorrect.

Assuming this is the first run of the program, i.e. no timers are going, the microcontroller will remain in idle until one of the push buttons triggers an IOC interrupt.

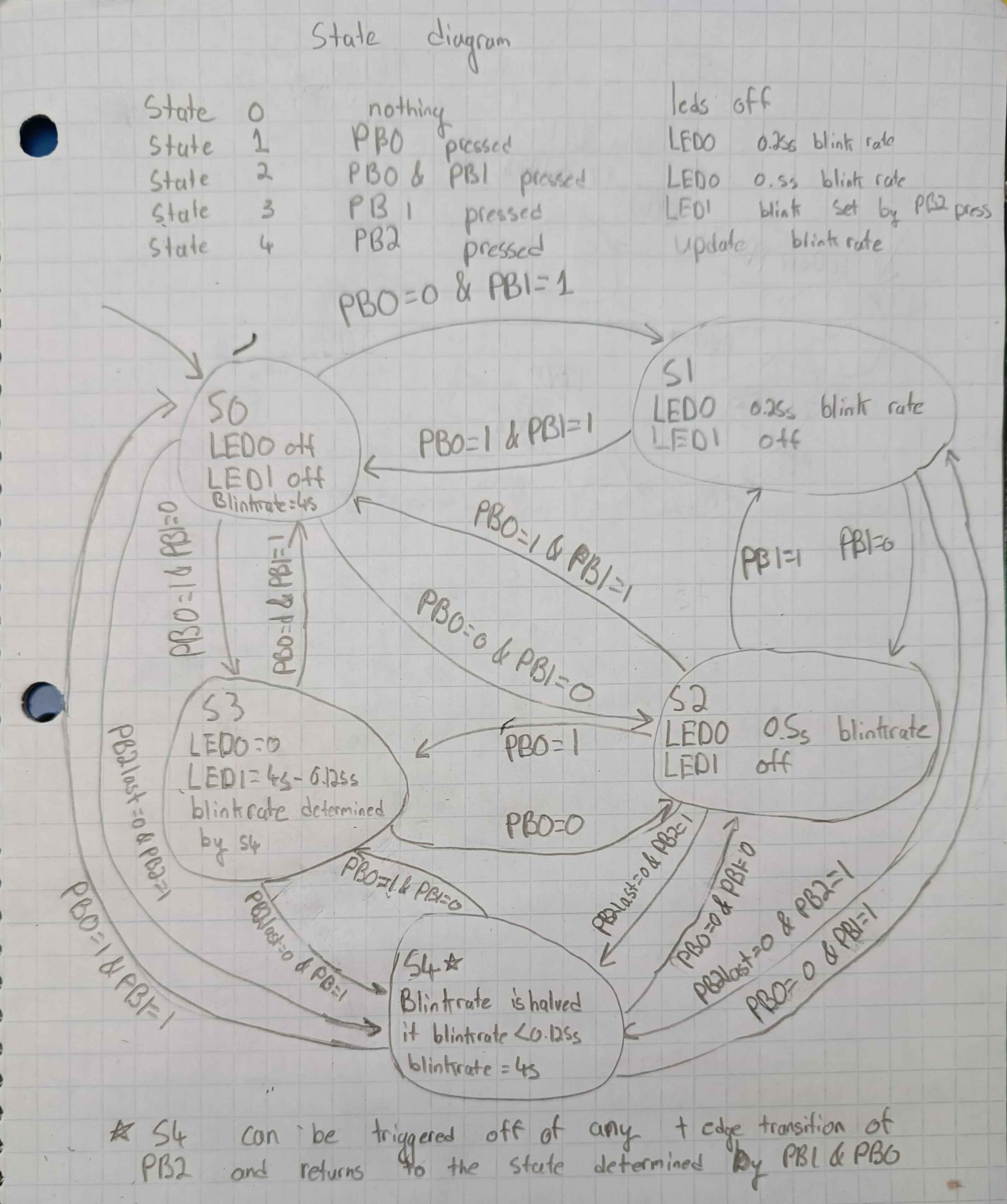


The IOC ISR will then cause the pb\_event variable to be set. The flag is then cleared, and the program returns to main.

After getting out of idle, the programs checks to see if the pb\_event flag was asserted. This is to verify that the reason for getting out of idle was a pb event before doing the logic rather than a timer or other interrupt.

After verify a pb event occurred, a debounce delay is incurred using the delay\_ms function, this helps to avoid multiple high and low transitions on a button press.

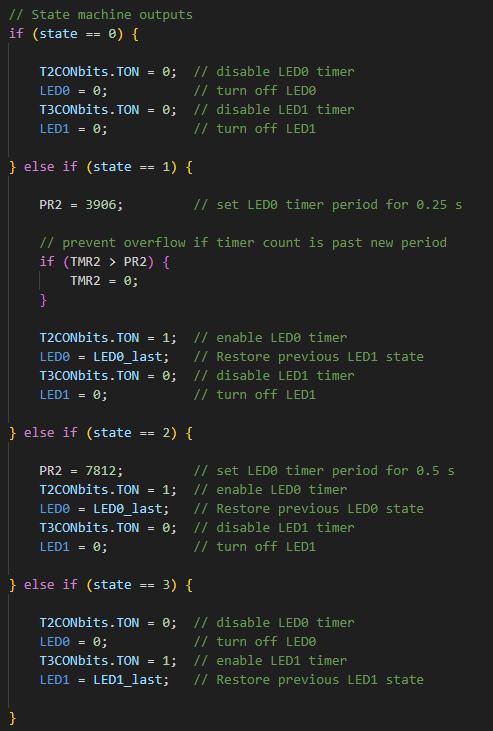
After the debounce the program reevaluates which state it should be in based on the buttons, this is illustrated in the state diagram below.

\

A computer screen shot of a program code

AI-generated content may be incorrect.

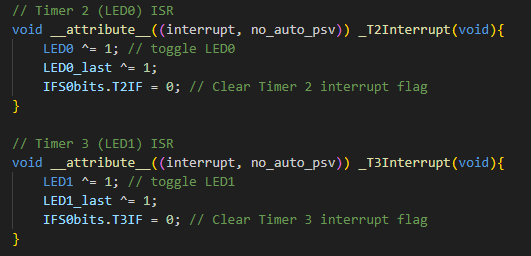
In the case that the program detects that PB2 has been released, it enters this function which updates the period of timer which is the blink rate of LED1. First it checks to see if the blink rate is less than or equal to the lowest blink rate of 0.125s, if that’s true it resets the blink rate to 4s. Otherwise it simply divides the blink rate by 2. We also added a TMR3 reset if it’s greater than PR3, this was meant to catch the case when you switch to a lower blink rate or PR3, but TMR3 has gone over PR3 and thus would count till it overflows.



The program then updates the appropriate timers and LEDs as per the assignment specifications. See state diagram for more details.



Finally the pb\_event is reset so it can wait for the next button input and the state of PB2 is saved in pb2\_last so it can be compared in the future to determine in a press-and-release has occurred. The microcontroller then returns to idle.



Idle may now be exited by one of Timer 2 or Timer 3 (depending on the current state) as well as by a button input. Upon a timer interrupt the ISR toggles one of LED0 or LED1 and saves its state in LEDx\_last before clearing its interrupt flag and returning to main.

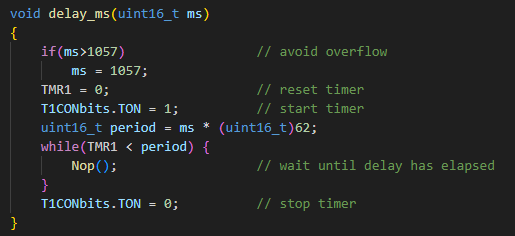
Answers to Lab Questions

*In the original provided program, why does the LED appear to “flash” every time you “click” a button (quickly)?*

Because multiple interrupts are occurring as the button “bounces” which causes multiple high-low and low-high transitions.

*If you press a button and hold it there, does the IOC interrupt service routine keep executing repeatedly? If so, why? If not, why not?*

No, the IOC does not repeatedly trigger on a button being held down as it only interrupts on a change (i.e. low to high or high to low transition).

A computer screen with text on it

AI-generated content may be incorrect.*Explain your implementation of void delay\_ms(uint16\_t ms). How do you guarantee that the delay works across the range of possible input values? What combination of clock/prescaler/PRx values do you use? What is the maximum delay you can use for the FOSC and timer prescaler setting?*

The maximum delay for the delay\_ms function is 1057ms as any number larger will cause in overflow in the calculation. A prescaler of 64 was used to maximize resolution while still allowing up to 1 sec of delay.

*How many timers do you use? What do you use each timer for?*

Three timers are used, Timer 1 is used for the delay\_ms function (which is used to debounce), Timer 2 and Timer 3 are used for blinking LEDs and specified rates.

*Explain your implementation of the IOC interrupt service routine, and your choice of IOC settings. How do you distinguish between a button press and a release? How about the different buttons?*

The IOC register is configured to trigger on high and low going transitions on all three buttons, this allows the detection of any change of state in the pressed buttons. To detect a press and release on push button 2 the previous state is stored. Iif the button was pressed in the previous state and is now high it is considered a press and release.