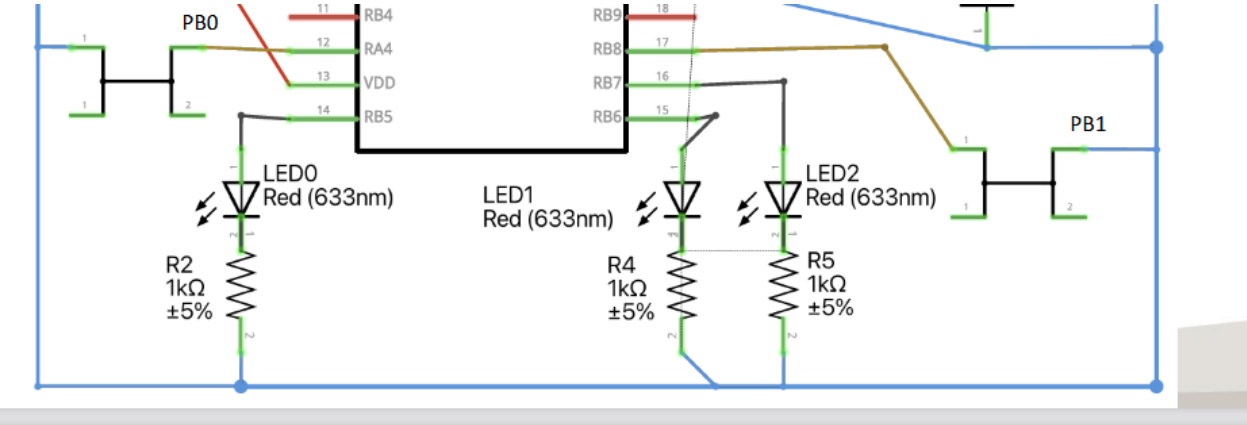
ENCM 511 Assignment1

Preamble

```
#define LEFT_LED_CTRL LATBbits.LATB5
#define MIDDLE_LED_CTRL LATBbits.LATB6
#define RIGHT_LED_CTRL LATBbits.LATB7

#define LEFT_BTN_STAT PORTAbits.RA4
#define RIGHT_BTN_STAT PORTBbits.RB8
```

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.



Left, Middle, and Right LED map to LED0, LED1, and LED2 respectively, matching the map above. PB0 and PB1 map to left and right button in the same manner.

Peripherals

```
TRISBbits.TRISB5 = 0;    // Set to output (left LED)
TRISBbits.TRISB6 = 0;    // Set to output (middle LED)
TRISBbits.TRISB7 = 0;    // Set to output (right LED)

TRISAbits.TRISA4 = 1;    // Set to input (left button)
TRISBbits.TRISB8 = 1;    // Set to input (right button)

IOCPUAbits.IOCPA4 = 1;   // Enable pull-up (left button)
IOCPUBbits.IOCPB8 = 1;   // Enable pull-up (right button)
```

The three LEDs have had their corresponding register bits set to 0 (output) as noted in the comments above.

The two buttons have had their corresponding register bits set to 1 (input) as noted in the comments above.

The two buttons have also had the pull-ups on their pins enabled to ensure a valid state when the button is not pressed.

Datatypes and Initialization

```
uint8_t state = 0;

uint16_t milliseconds = 0;

uint16_t i = 0;
```

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

The milliseconds variable will be in the range of 0 to 4000 and thus is an unsigned 16bit integer.

The i variable will be in the range of 0 to 444 and thus is an unsigned 16bit integer.

Operation

The code was architected with the goal of allowing the button state (i.e. which light is flashing) to be changed quickly and not be forced to wait for the expiration of the last state which could be quite long in the case of the 2 second blink.

This was achieved by first determining the sate of the buttons at the beginning of the main loop.

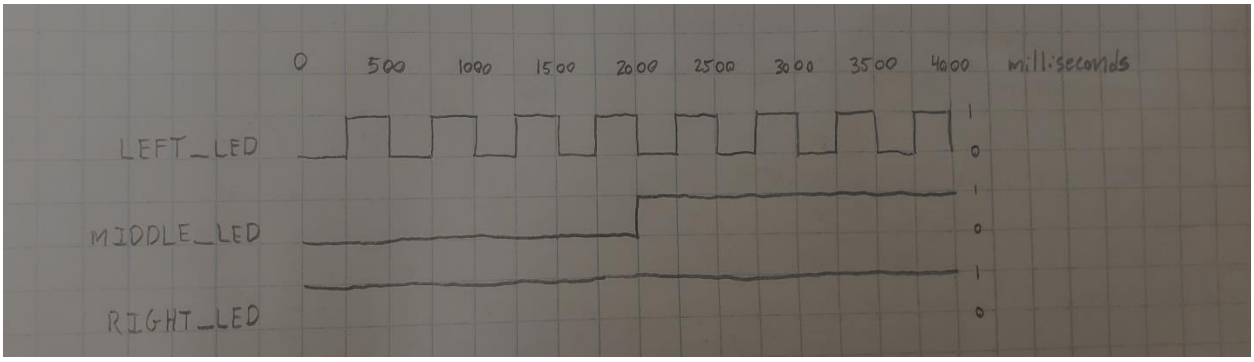
```
while(1){
    if(LEFT_BTN_STAT == 0 && RIGHT_BTN_STAT == 1) {
        state = 1;
    } else if(LEFT_BTN_STAT == 1 && RIGHT_BTN_STAT == 0) {
        state = 2;
    } else if (LEFT_BTN_STAT == 0 && RIGHT_BTN_STAT == 0) {
        state = 3;
    } else {
        state = 0;
    }
```

This gives 4 possible states and is stored in the state variable.

The LEDs are then updated depending on the current value of the milliseconds variable and the updated button state.

```
    if(state == 1 && (milliseconds % 500) > 250) {
        LEFT_LED_CTRL = 1;
        MIDDLE_LED_CTRL = 0;
        RIGHT_LED_CTRL = 0;
    } else if (state == 2 && milliseconds > 2000) {
        LEFT_LED_CTRL = 0;
        MIDDLE_LED_CTRL = 1;
        RIGHT_LED_CTRL = 0;
    } else if (state == 3) {
        LEFT_LED_CTRL = 0;
        MIDDLE_LED_CTRL = 0;
        RIGHT_LED_CTRL = 1;
    } else {
        LEFT_LED_CTRL = 0;
        MIDDLE_LED_CTRL = 0;
        RIGHT_LED_CTRL = 0;
    }
```

This logic results in the following diagram for each LED vs the milliseconds variable. Each LED only receives its counting pattern if the appropriate state is active.



After the LED update NOP instructions are added to increase the time of the loop to approximately 1 millisecond.

```
i = 0;
while (i < 444) {    // approx 1 millisecond delay
    Nop();
    i++;
}
```

Then the milliseconds counter is incremented and checked for a periodic reset at 4000ms as this is the longest period required for any of the LEDs and can be looped after this point.

```
milliseconds++;

if (milliseconds >= 4000) {  // Only ever need up to 4000ms period. Avoids overflow
    milliseconds = 0;
}
```

This means the longest response time to a change in the button inputs is approximately 1ms.