Lab 2: Morse Code Decoder

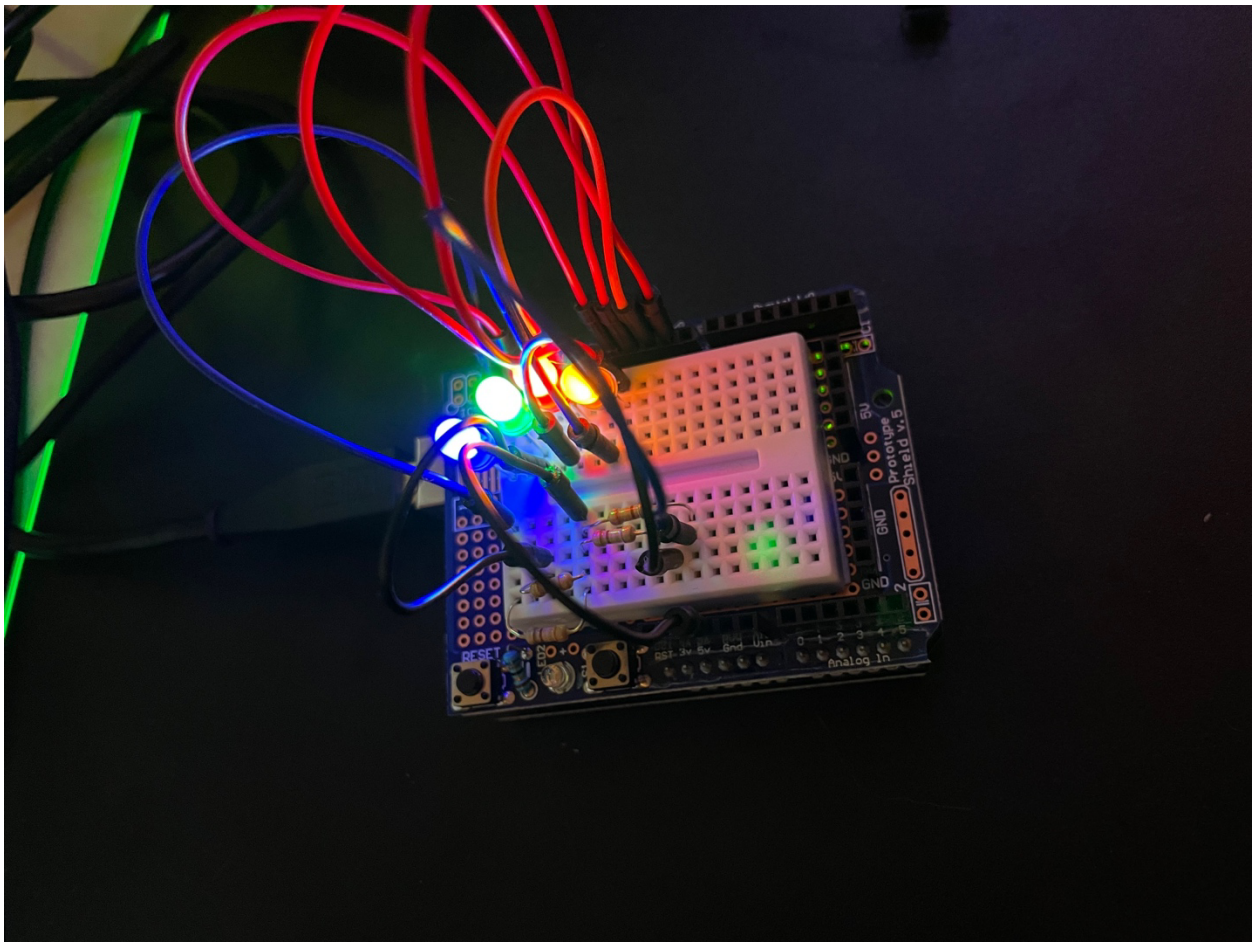
ESE519/IPD519: Introduction to Embedded Systems
University of Pennsylvania

In this document, you'll fill out your responses to the questions listed in the Lab 2 Manual. Please fill out your name and link your Github repository below to begin. Be sure that your code on the repo is up-to-date before submission!

For all the questions that require a video, you can attach the video/image directly to the relevant question number or provide a link to the video (e.g. youtube, google drive, etc.).

Student Name: Ziyi Yang
Pennkey: ziyiyang
GitHub Repository: https://github.com/Zakiyanggg/Intro-to-Embedded-system/tree/main/Lab2

1.



2.

```c
int main(void)
{
    /* Replace with your application code */
    DDRB |= (1<<DDB1);//set PB1 as output DIGITAL PIN9
    DDRB |= (1<<DDB2);//2 PIN10
    DDRB |= (1<<DDB3);//3 PIN11
    DDRB |= (1<<DDB4);//4 PIN12

    while (1)
    {
        PORTB |=(1<<PORTB1);//set PB1 to HIGH
        PORTB |=(1<<PORTB2);//2
        PORTB |=(1<<PORTB3);//3
        PORTB |=(1<<PORTB4);//4

    }
}
```

3.
Yes, it does. But I didn't know that printf doesn't work here, please ignore them.
4.

```c
int main(void)
{
    /* Replace with your application code */
    DDRD &= ~(1<<DDD7);//PD7 setup as input pin
    DDRB |= (1<<DDB1);//PB1 setup as output pin

    while (1)
    {

        if (PIND &(1<<PIND7)){
            printf("ON");//if the value at port is HIGH
            PORTB |= (1<<PORTB1);//drive output high to turn LED on
        }
        else{
            printf("OFF");//if the value at port is LOW
            PORTB &= ~(1<<PORTB1);//drive output low to turn LED off
        }

    }
}
```

5.

6.
Most of the time it works, but sometimes it jumps over a LED and light up the one after.

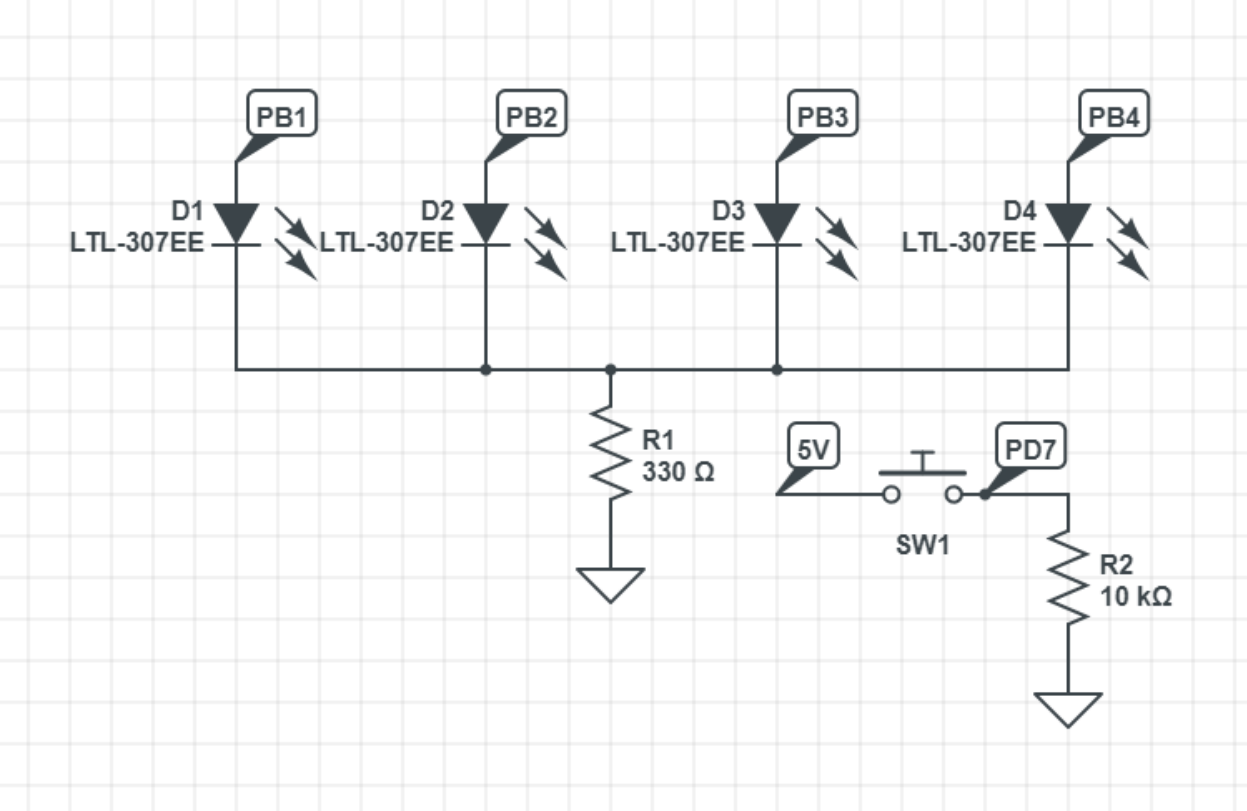
7.

```c
int main(void)
{
    /* Replace with your application code */
    DDRD &= ~(1<<DDD7);//PD7 setup as input pin
    DDRB |= (1<<DDB1);//PB1 setup as output pin
    DDRB |= (1<<DDB2);//PB2 setup as output pin
    DDRB |= (1<<DDB3);//PB3 setup as output pin
    DDRB |= (1<<DDB4);//PB4 setup as output pin
    int count = 0;//set a counter
    int i = 4;
    while (1)
    {

        if(PIND &(1<<PIND7)){//If input is HIGH then add 1 to count
            count = count + 1;
            _delay_ms(1000);//delay to avoid multiple addition for one press
            i = i + count;
        }
        else{
            i = i %4;
            switch (i){//the reminder of i is the control of which led to light up
                case 0://1st LED on, rest off
                    PORTB |= (1<<PORTB1);
                    PORTB &= ~(1<<PORTB2);
                    PORTB &= ~(1<<PORTB3);
                    PORTB &= ~(1<<PORTB4);
                    break;
                case 1://2nd LED on, rest off
                    PORTB |= (1<<PORTB2);
                    PORTB &= ~(1<<PORTB1);
                    PORTB &= ~(1<<PORTB3);
                    PORTB &= ~(1<<PORTB4);
                    break;
                case 2://3rd LED on, rest off
                    PORTB |= (1<<PORTB3);
                    PORTB &= ~(1<<PORTB2);
                    PORTB &= ~(1<<PORTB1);
                    PORTB &= ~(1<<PORTB4);
                    break;
                case 3://4th LED no, rest off
                    PORTB |= (1<<PORTB4);
                    PORTB &= ~(1<<PORTB2);
                    PORTB &= ~(1<<PORTB3);
                    PORTB &= ~(1<<PORTB1);
                    break;
            }
        }

    }
}
```

8.



9.

I set PB8 as input pin and enabled the internal resistor so ht input pin is default HIGH, when press it turns to LOW. No need external resistor, easier to build.

10.

The interrupt only needs the CPU when triggered by an event, which can save the power of CPU for other tasks. Compare to interrupts, polling has a faster reaction time to minimize the delay between push button and led.

11.

30ms = 0.03s→0.03/ 1/16M = 480000ticks
200ms → 3200000ticks
270ms → 6400000ticks

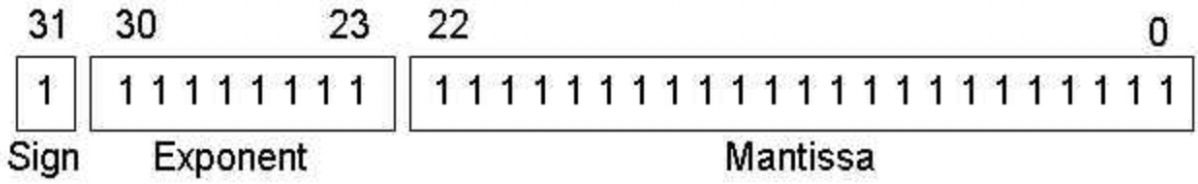
12.

Prescaler can be used to slow down the clock signals for timer. In our case, we have a 16Mhz clock and we can use prescaler to scale down the frequency to by 2/4/8/16/32/64/128/256. And we can prescale system clock or timer clock independently to get even more desire frequencies.

13. Video attached to Github

14. SOMEDAYIWILLRULEYOUALL
15. For unsigned integers, all bit are used to represent the corresponding value of 2^n(nth bit starts with 0). But a signed integer, the most left bit is known as sign bit. 0 represents positive and 1 for negative, no matter it's 8 bit, 16 bit or 32bit. Normally, if we wanna represents 8 in binary, it's 00001000, and as said above, -8 should be 10001000. But this method can only be used to solve addition problem. While subtract two values with this.
    Let's do +12+(-8)
    +12: 00001100
    -8: 10001000
    10010100→-20

    What it we use +4 – (+12) to get -8?
    +4 00000100
    +12 00001100(minus here)
    11111000
    Seems like we didn't get -8 but it's -8 in its two's compliment form.
    So we have two different method to represent negative number in binary. The way to transform a number to 2's compliment is first to inverse each digit(0 to 1, 1 to 0) and then plus 1. And then we can use normal way to compute positive numbers and negative numbers.
16. To represent float number in binary we use a method called single precision



    Sign bit is just like integer. 1 for negative and 0 for positive.
    Exponent = Scaling factor + bias (127 for single precision)
    Let's see an example. 1259.125 to single precision format.

    First, we convert decimal part into binary
    1259: 10011101011
    0.125 = 0.001. (1/(2^n)) n is the position after dot, since 0.125 = 1/8 = 1/(2^3)

    Then we normalize the number by forwarding the dot to the right of the most left 1.

    10011101011.001 → 1.0011101011001 x2 ^10
    Here we scale the number with exponent 10 because we are moving the dot 10 digits forward.

    E' = 10+127 = 137 → 10001001
    So we combine them toghther:
    0 10001001 00111010110010000000000