Lab 2: Morse Code Decoder

ESE3500: Embedded Systems & Microcontroller Laboratory
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, provide a link to the video (e.g. youtube, google drive, etc.).

Student Name: Katie Zhang
Pennkey: zhangkat
GitHub Repository: https://github.com/ese3500/lab-2-morse-zhangkat48

1.

```
int main(void)
{
    DDRB |= (1<<DDB1); //PB1 set to output pin
    PORTB |= (1<<PORTB1); //Drive output high to turn on LED

    DDRB |= (1<<DDB2); //PB2 set to output pin
    PORTB |= (1<<PORTB2); //Drive output high to turn on LED

    DDRB |= (1<<DDB3); //PB3 set to output pin
    PORTB |= (1<<PORTB3); //Drive output high to turn on LED

    DDRB |= (1<<DDB4); //PB4 set to output pin
    PORTB |= (1<<PORTB4); //Drive output high to turn on LED

    /* Replace with your application code */
    while (1)
    {
    }
}
```

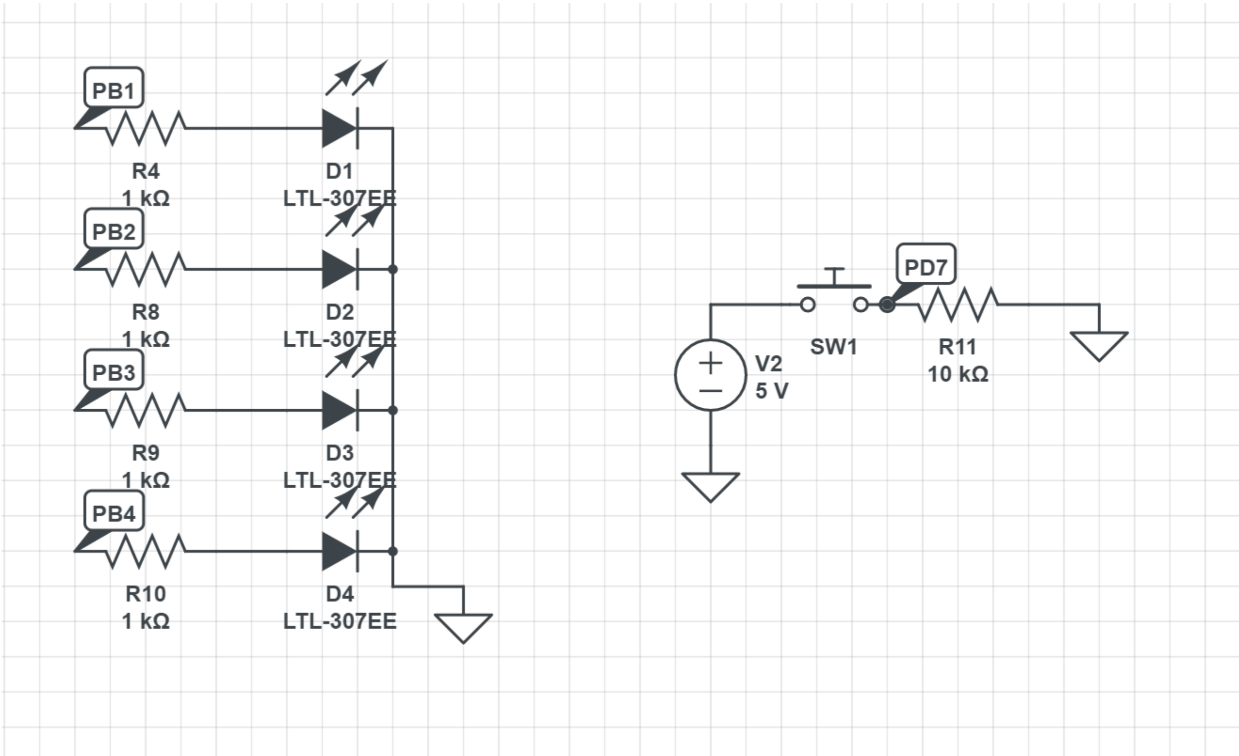
2.

```c
int main(void)
{
    DDRB |= (1<<DDB1); //PB1 set to output pin
    PORTB |= (1<<PORTB1); //Drive output high to turn on LED

    DDRB |= (1<<DDB2); //PB2 set to output pin
    PORTB |= (1<<PORTB2); //Drive output high to turn on LED

    DDRB |= (1<<DDB3); //PB3 set to output pin
    PORTB |= (1<<PORTB3); //Drive output high to turn on LED

    DDRB |= (1<<DDB4); //PB4 set to output pin
    PORTB |= (1<<PORTB4); //Drive output high to turn on LED

    DDRD &= ~(1<<DDD7); //set PB7 as input pin


    while (1)
    {
        if (PIND & (1<<PORTD7))
        {
            PORTB |= (1<<PORTB1); //set to high
        } else {
            PORTB &= ~(1<<PORTB1); //set to low
        }
    }
}
```

3.

```c
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    char count = 1;

    DDRB |= (1<<DDB1); //PB1 set to output pin
    PORTB |= (1<<PORTB1); //Drive output high to turn on LED

    DDRB |= (1<<DDB2); //PB2 set to output pin
    PORTB &= ~(1<<PORTB2); //output set to low to be initially off

    DDRB |= (1<<DDB3); //PB3 set to output pin
    PORTB &= ~(1<<PORTB3); //output set to low to be initially off

    DDRB |= (1<<DDB4); //PB4 set to output pin
    PORTB &= ~(1<<PORTB4); //output set to low to be initially off

    DDRD &= ~(1<<DDD7); //set PB7 as input pin


     while (1)
     {
         if (PIND & (1<<PORTD7) && count == 1) {
            PORTB &= ~(1<<PORTB4); //turn off LED 4 (LOW)
            PORTB |= (1<<PORTB1); //Turn on LED 1 (HIGH)
            _delay_ms(10000); //added delay so count does not increment when button is held
            count++; //increment count to light up next LED
         } if (PIND & (1<<PORTD7) && count == 2) {
            PORTB &= ~(1<<PORTB1); //turn off LED 1 (LOW)
            PORTB |= (1<<PORTB2); //Turn on LED 2 (HIGH)
            _delay_ms(10000);
            count++;
         } if (PIND & (1<<PORTD7) && count == 3) {
            PORTB &= ~(1<<PORTB2); //turn off LED 2 (LOW)
            PORTB |= (1<<PORTB3); //Turn on LED 3 (HIGH)
            _delay_ms(10000);
            count++;
         } if (PIND & (1<<PORTD7) && count == 4) {
            PORTB &= ~(1<<PORTB3); //turn off LED 3 (LOW)
            PORTB |= (1<<PORTB4); //Turn on LED 4 (HIGH)
            _delay_ms(10000);
            count++;
            }
        if (count > 4) {
            count = 1; //after lighting up the 4th LED, the cycle return to start with LED 1
        }
     }
   }
```

4.



5. Advantage: The main advantage of using interrupt instead of polling is that it uses less CPU processing power. Unlike polling which continuously checks to see if a condition is met, interrupts interrupt the CPU only when an event is triggered. Disadvantage: Polling is simpler to implement and easier to understand than interrupts. Since interrupts are not explicitly scheduled in the code and can happen at any time, it requires careful coding to ensure that other parts of the program do not get broken. Using interrupts has a risk of being asynchronous with the rest of the program.

6. For 30ms: $\frac{16\,x\,10^6}{1s} = \frac{X}{30x10^{-3}}$ => $(16\,x\,10^{6})(30\,x\,10^{-3}) = 480000$ ticks

For 200ms: $\frac{16\,x\,10^6}{1s} = \frac{X}{200x10^{-3}}$ => $(16\,x\,10^{6})(200\,x\,10^{-3})$ 3200000 ticks

For 400ms: $\frac{16\,x\,10^6}{1s} = \frac{X}{400x10^{-3}}$ => $(16\,x\,10^{6})(400\,x\,10^{-3})$ 6400000 ticks

7. Prescaler allows you to reduce the frequency, which, depending on the prescaled value, would allow the 16 bit timer, or 8 bit timer, or both to work if the ticks they count up to (i.e. 2^{16} for 16-bit timer) is larger than the reduced frequency value. To further slow down the clock frequency for us to be able to use either the 8-bit or 16 bit timer, both the system clock and the timer can be prescaled (reduced). Whether or not a

16-bit or 8-bit timer would function can be determined by $\frac{\frac{clock\ frequency}{clock\ prescale\ value}}{timer\ prescale\ value}$. Using a prescaler also reduces the number of times a timer overflows which could be useful for certain applications (i.e. makes the morse code lab calculations easier).

8. https://drive.google.com/file/d/11pzIFlTXqqRpWQ-1sHOesmnAIw2BfMiB/view?usp=sharing

   (video was compressed so may have to increase the quality of the video to 720p)

9. Someday I will rule you all
10. https://drive.google.com/file/d/1TeFTEWOaPCrxhr-Kd1v7ZGVVDGXtyA8F/view?usp=share_link