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: Gene Liu
Pennkey: geneliu
GitHub Repository: https://github.com/ese3500/lab-2-geneliu

1.

```
int main(void)
{
    DDRB |= (1 << DDB1);
    DDRB |= (1 << DDB2);
    DDRB |= (1 << DDB3);
    DDRB |= (1 << DDB4);
    while (1)
    {
        PORTB |= (1<<PORTB1);
        PORTB |= (1<<PORTB2);
        PORTB |= (1<<PORTB3);
        PORTB |= (1<<PORTB4);
    }
}
```

2.

```
int main(void)
{
    DDRB |= (1 << DDB1);
    DDRB |= (1 << DDB2);
    DDRB |= (1 << DDB3);
    DDRB |= (1 << DDB4);
    DDRD &= ~(1 << DDD7);
    while (1)
    {
        PORTB &= ~(1<<PORTB1);
        PORTB &= ~(1<<PORTB2);
        PORTB &= ~(1<<PORTB3);
        PORTB &= ~(1<<PORTB4);
        if (PIND & (1<<PIND7)) {
            PORTB |= (1<<PORTB1);
            PORTB |= (1<<PORTB2);
            PORTB |= (1<<PORTB3);
            PORTB |= (1<<PORTB4);
        }
    }
}
```

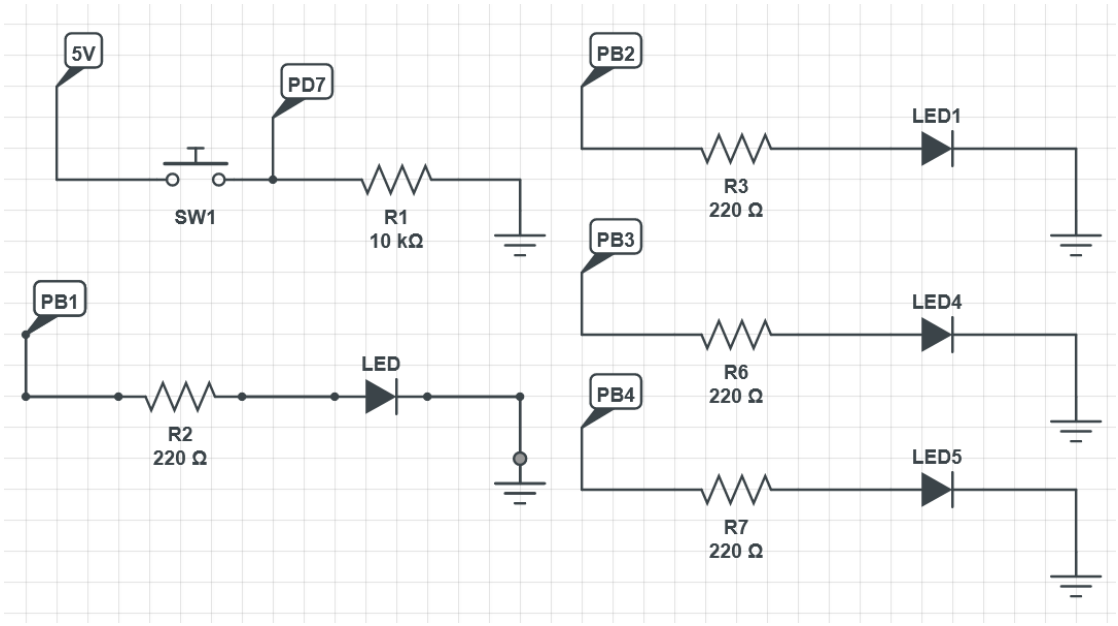
3.

```
void init()
{
    DDRB |= (1 << DDB1);
    DDRB |= (1 << DDB2);
    DDRB |= (1 << DDB3);
    DDRB |= (1 << DDB4);
    DDRD &= ~(1 << DDD7);

    PORTB |= (1<<PORTB1);
}

int main(void)
{
    init();
    int offset = 1;
    while (1)
    {
        if (PIND & (1<<PIND7)) {
            PORTB = 0;
            PORTB |= (1<<(PORTB1 + offset));
            offset = (offset + 1)%4;
            _delay_ms(2000);
        }
    }
}
```

4.



5. Interrupts are more power and compute efficient than continuous polling, as we do not have to constantly loop and look for the relevant signals and instead are notified whenever it happens to deal with it. However, interrupts may take more time to execute the callback, as it can only happen in line with the underlying interrupt cycles while polling can immediately pick up and deal with the signal as it is constantly looking for it.
6. A 16MHz clock ticks 16 million times a second, and so each tick corresponds to (1/16000000) = 6.25 * 10^-8 seconds. Thus, 30 ms takes (0.03/6.25e-8) = 480000

ticks, 200ms takes (0.2/6.25e-8) = 3200000 ticks, and 400ms takes (0.4/6.25e-8) = 6400000 ticks.

7. A prescaler allows you to slow the system or timer clocks down so that it takes longer to overflow the clock registers. This corresponds to it taking longer to do so, and so allows us to work with a wider range of frequencies before overflow occurs.
8. https://drive.google.com/file/d/15pii-wKJXzJlrP6BIGtftIFD-c1agpfu/view?usp=sharing
9. Someday i will rule you all
10.