# 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:** Emily Shen
**Pennkey:** shenyit
**GitHub Repository:** EmilyYShen

## Part A

1. Screenshot of code to light up 4 LEDs.

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

int main(void) {

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

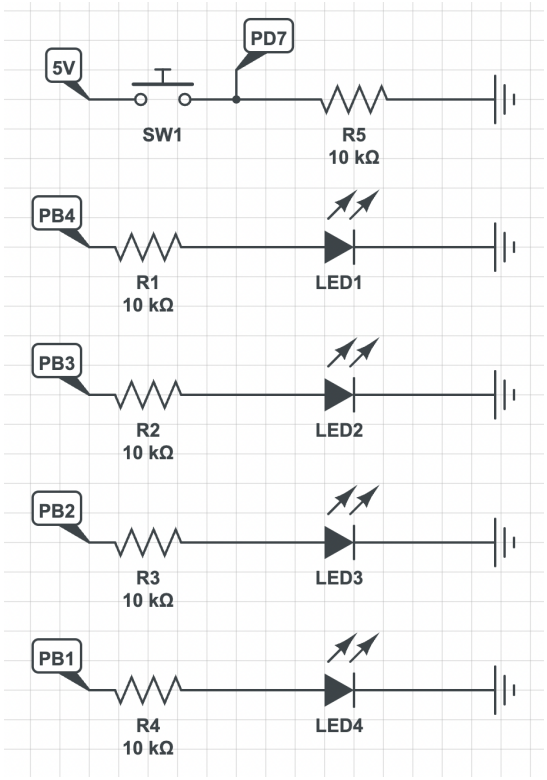
2. Screenshot of code to use a switch to control an LED.

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

int main() {

    DDRD &= ~(1<<DDD7);
    DDRB |= (1<<DDB1);

    while(1) {
        if (PIND & (1<<PIND7)) {
            PORTB |= (1<<PORTB1);
        } else {
            PORTB &= ~(1<<PORTB1);
        }
    }
}
```

3. Screenshot of code to light up 4 LEDs sequentially when a button is pressed.

```
int main() {
    // set up input and output pins
    DDRD &= ~(1<<DDD7);
    DDRB |= (1<<DDB1);
    DDRB |= (1<<DDB2);
    DDRB |= (1<<DDB3);
    DDRB |= (1<<DDB4);
    // set pin 9 as lit
    PORTB |= (1<<PORTB1);

    while(1) {
        if ((PIND & (1<<PIND7))) {
            if (PORTB & (1 << PORTB1)) {
                PORTB &= ~(1 << PORTB1);
                PORTB |= (1 << PORTB2);
                _delay_ms( ms: 1000);
            }
            else if (PORTB & (1 << PORTB2)) {
                PORTB &= ~(1 << PORTB2);
                PORTB |= (1 << PORTB3);
                _delay_ms( ms: 1000);
            }
            else if (PORTB & (1 << PORTB3)) {
                PORTB &= ~(1 << PORTB3);
                PORTB |= (1 << PORTB4);
                _delay_ms( ms: 1000);
            }
            else if (PORTB & (1 << PORTB4)) {
                PORTB &= ~(1 << PORTB4);
                PORTB |= (1 << PORTB1);
                _delay_ms( ms: 1000);
            }
        }
    }
}
```

4. Schematic of the circuit used in Question 3 using CircuitLab



## Part B

5.

*Advantage:* Interrupts are a simple way to make the system more responsive to time-sensitive tasks. Since pressing and releasing a button is time-sensitive depending on when you press or release the button, using an interrupt will light up and turn off the LED at a more responsive rate to your action than if you just used the while loop in the main function. Furthermore, the interrupt uses less CPU processing power than polling. This is because polling continuously checks to see if a condition is met whereas the interrupt only interrupts the CPU when something triggers it.

*Disadvantage:* Since this is a relatively simple task of turning on and off an LED, the code is simpler and easier to read if you just used the while loop in the main function rather than the extra code required for an interrupt, such as enabling the pin change interrupt to detect for the rising and falling edge, and enabling the trigger. Furthermore, this task was simple in that we did not have to account for overflow or more complex calculations that may require an interrupt.

Screenshot of code to light up LED when a button is pressed and turn off LED when button is released.

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

void Initialize() {

    // disable global interrupts
    cli();

    // flag
    TIFR1 |= (1<<ICF1);

    // set up input and output pins
    DDRB |= (1<<DDB5);
    DDRB &= ~(1<<DDB0);

    // enable pin change interrupt (rising)
    TCCR1B |= (1<<ICES1);

    // enable trigger
    TIMSK1 |= (1<<ICIE1);

    sei();
}

ISR(TIMER1_CAPT_vect) {
    TIFR1 |= (1<<ICF1);
    PORTB ^= (1<<PORTB5);
    TCCR1B ^= (1<<ICES1);
}

int main() {
    Initialize();
    while(1);
}
```

## Part C

6.

Ticks (30ms) = 16000000/1000 * 30 = 480,000
Ticks (200ms) = 16000000/1000 * 200 = 3,200,000
Ticks (400ms) = 16000000/1000 * 400 = 6,400,000

7. The prescaler divides down the clock signals that are used for the timer, so this reduces overflow rates. This is particularly useful for this Morse code lab, as there is a lot of potential for overflow since a 16MHz clock overflows every 4ms, whereas we want to calculate at least up to 400ms. The prescaler is used by reducing the frequency, which would essentially slow down the clock. This prescaler will work if the ticks that are counted up to are larger than the reduced frequency value. For example, if a 8MHz timer clock is set to a prescaler of 8, the timer clock now runs at 8MHz/8 = 1MHz. Thus, the timer runs at a slower rate so there will be less chance of overflow. Both the timer and the system clock can both be prescaled to slow down the clock frequency even further. We can calculate the Count and Prescaler by using the following equation:

$$Count = \frac{f(clk1/O)}{2 * N * f(desired)} - 1$$

$f_{desired}$ is the frequency outputted, $f_{clk/O}$ is system clock frequency after any system clock prescalers applied. N is the timer prescaler and count is the number of ticks we count up to. If N=1 (no prescaler applied), and the count exceeds the maximum possible value of the clock, we can set N to a different larger value so that the count is less than the maximum possible value of the clock.

## Part D

8. The link below is a demo video of the Morse code translations. The red LED blinking represents a dot and the green LED blinking represents a dash.

I am printing “EMILY” into the serial port:

E . <br>
M _ _ <br>
I . . <br>
L . _ . . <br>
Y _ . _ _

Other than the UART file in order to print to the serial port, no other files were used. The times for dashes, dots, and spaces were also not extended or changed from Part C.

I used a time prescaler value of 256. Though the times for dashes, dots, and spaces were not changed, the number of ticks calculated for 30ms, 200ms and 400ms were different due to the prescaler. For 30ms, the number of ticks was now 480000/256 = 1875. For 200ms, the number of ticks was now 12500. For 400ms, the number of ticks was now 25000.

The code is composed of three main sections. There is a “printLetter” function that checks what ASCII character the String array contains. After the function has checked what the ASCII character is, the String array is then reset to take in the next Morse code.

The interrupt checks to see if the switch has been pressed or released. When the switch has been pressed, TCNT1 is reset to 0 and an unsigned int i is also set to 0. This is done in order to prevent potential overflow. Once the switch has been released, an unsigned int j takes the current value of TCNT1.

Then the difference of j - i is calculated. If it is between 30ms to 200ms, the red LED is lit up and a dot is stored in the String array. If it is between 200ms to 400ms, the green LED is lit up and a dash is stored in the String array. If it is longer than 400ms or less than 30ms, then nothing will happen. Also, checkSpace, a variable that keeps track of whether the last value in the String array is a dot or dash, is set to 0.

The main function keeps track of the unsigned int light variable, and once this light variable increments to 5000, it turns off the red or green LED. This is to prevent the use

of delays in the code whilst also being able to turn off the LED after a small amount of time.

We also check that if after a certain amount of time no button is pressed, we print out one space. This is done in the main function. First, we must account for overflow. If j + 40000 is greater than 62500, the maximum value that the timer goes up to, as we have prescaled the timer by the value of 256, then we reset TCNT1 to 0 and take away 62500 from j to prevent overflow. Then we check if j + 40000 is less than TCNT1. If it is, we set checkSpace to 1 to indicate that we have gone to a space, and we go to the “printLetter” function.

Note: The video recording could not capture all the LED flashes as the video was compressed, even though they were lighting up in real-life.

https://drive.google.com/file/d/1P3QgMHXoYqEVa79Ag3l1B47OC1zRVJUw/view?usp=share_link

## Extra Credit

9. Translation: SOMEDAY I WILL RULE YOU ALL