Lab 2: Morse Code Decoder

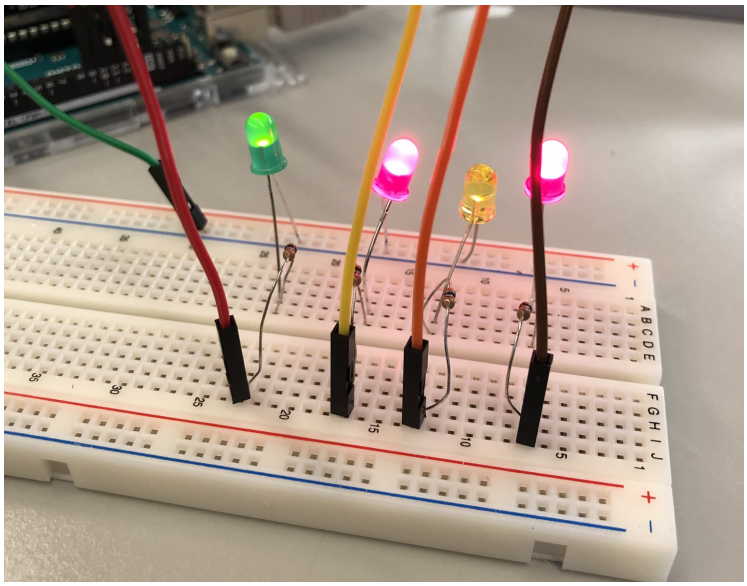
ESE519/IPD519: Introduction to Embedded Systems
University of Pennsylvania

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: Jonathan Zur
Pennkey: jzur
GitHub Repository: https://github.com/Jonathanzur1/lab2-jzur.git

1. The yellow light appears dim, it is a dim LED bulb.



2.

```
#include <asf.h>

#include <avr/io.h>

int main (void)
{
    /* Insert system clock initialization code
 DDRB |= (1<<DDB1);
 DDRB |= (1<<DDB2);
 DDRB |= (1<<DDB3);
 DDRB |= (1<<DDB4);
 //PORTB |= (1 << PORTB1);
 //PORTB |= (1 << PORTB2);
 //PORTB |= (1 << PORTB3);
 //PORTB |= (1 << PORTB4);

 //PORTB |= 0x1E;
 PORTB |= 0b11110;
    board_init();

    /* Insert application code here, after the
}
```

3. All four external LEDs and the internal LED turn on when the button is pressed. This is the expected behavior observed, even when no debouncing is used.

4.

```
while(1){
 DDRD &= ~(1<<DDD7); //Set PD7 as an input pin
 if (PIND & (1<<PIND7)) //Check if value at port is HIGH
 {
    PORTB |= (1 << PORTB5);
    PORTB |= 0b11110;
 }
 else
 {
     PORTB &= ~(1<<PORTB5);
     PORTB &= ~(1 << PORTB1);
     PORTB &= ~(1 << PORTB2);
     PORTB &= ~(1 << PORTB3);
     PORTB &= ~(1 << PORTB4);
 }
}
```

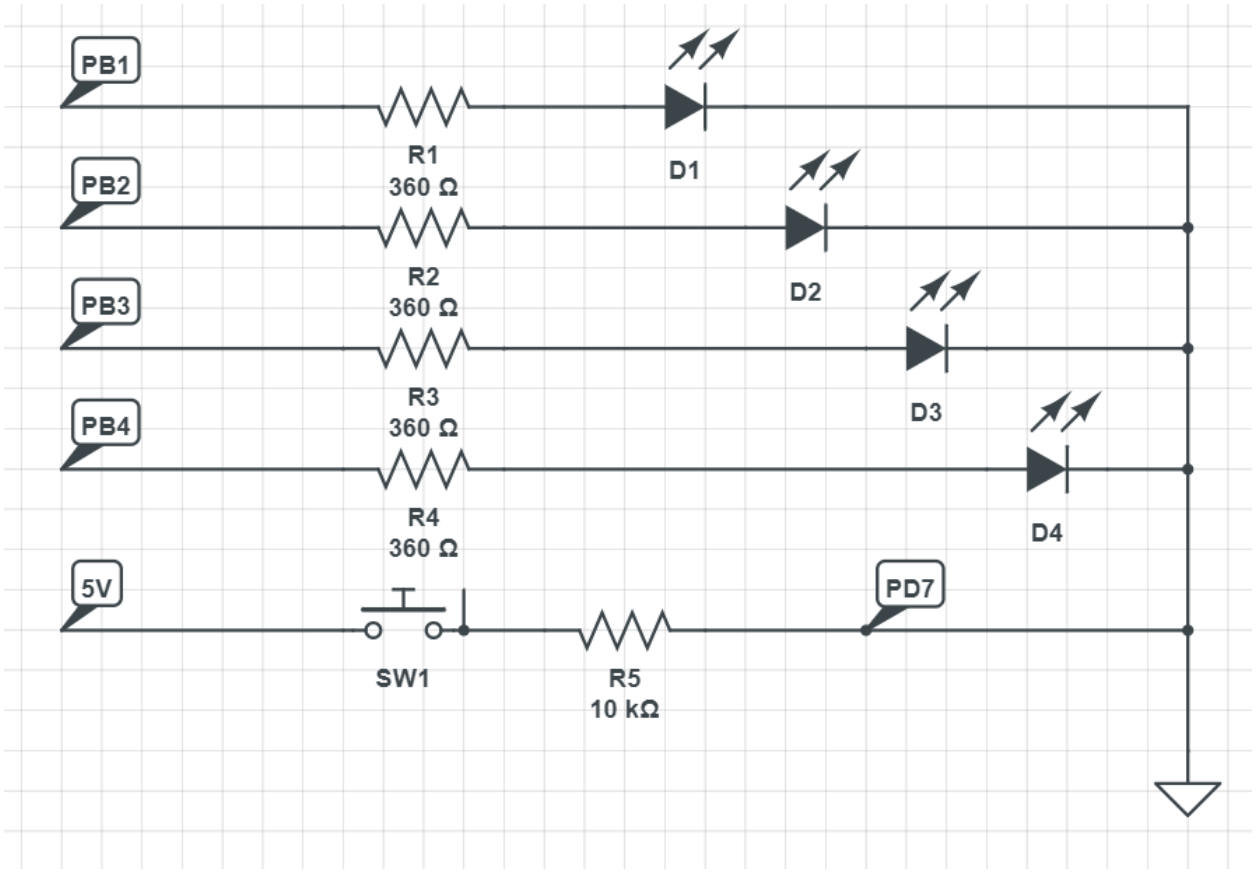
5. -
6. The lights turn on sequentially, however without the delay the cycle was happening much too quickly. A delay needed to be included to visualize the LEDs changing, and even so it was difficult to time the button press properly so that each LED would light up on each button press.

7.

```
//TURN ON LEDS SEQUENTIALLY
int i = 0;
while(1){
    PORTB &= ~(1 << PORTB1);
    PORTB &= ~(1 << PORTB2);
    PORTB &= ~(1 << PORTB3);
    PORTB &= ~(1 << PORTB4);
    PORTB &= ~(1 << PORTB5);

    if (PIND & (1<<PIND7) && i == 0){
       PORTB = 0;
       PORTB |= (1 << PORTB1);
       i++;
    }
    else if (PIND & (1<<PIND7) && i == 1){
       PORTB = 0;
       PORTB |= (1 << PORTB2);
       i++;
    }
  else if (PIND & (1<<PIND7) && i == 2){
      PORTB = 0;
      PORTB |= (1 << PORTB3);
      i++;
  }
  else if (PIND & (1<<PIND7) && i == 3){
      PORTB = 0;
      PORTB |= (1 << PORTB4);
      i++;
  }
  else if (PIND & (1<<PIND7) && i == 4){
      PORTB = 0;
      PORTB |= (1 << PORTB1);
      i = 1;
  }
  _delay_ms(500);
```

8.



9. In the first part of part A, I searched for when the pin was active high. When the circuit was closed and five volts was read, the LEDs would all light up. In the second part of part A, I used a similar approach but used the button as a toggle and not a press-and-hold button. In part B, I polled for when the LED was low. To do this, I switched the positions of the switch and the resistor in the circuit.

10. An advantage of using interrupts instead of polling here is saving CPU capability. In polling, the system is constantly looking for the event (e.g. pressing the button switch) whereas when interrupts are used, the CPU is free to focus on other tasks while the search for the interrupt happens in the background. In this particular case, no other tasks are required so the result is the same, however there are many circuits where lighting an LED is only one small part of the overall function.
A disadvantage of using interrupts in this case is that the function of the code is not easy to discern without referencing external documents. This is a disadvantage in learning purposes only, not for functionality of the overall system. In polling, the while loop implemented is relatively straightforward whereas when using interrupts, more specific pieces of code need to be looked up in the ATmega328P datasheet.

11.

(16 X 10^6) [1/s] X (30 X 10^-3) [s] = **4.8 X 10^5 Steps**
(16 X 10^6) [1/s] X (200 X 10^-3) [s] = **3.2 X 10^6 Steps**
(16 X 10^6) [1/s] X (400 X 10^-3) [s] = **6.4 X 10^6 Steps**

12. Without any prescaler, a timer will run at the same frequency as the system clock. The system clock often runs at very high frequencies as some applications such as flight control, airbag deployment, and many others require extremely fast reaction time from the computer. The disadvantage associated with this is that many timers used do not have enough memory to count to the maximum frequency of the system clock. In our case, while the system clock counts at a frequency of 16,000,000 ticks per second, our 16-bit timer can only count up to 65535 and so it will overflow rapidly and extremely often. By using a prescaler, the rate of the timer overflow can be slowed and the timer can capture the full count. In this case, a prescaler of 256 is being used so that the timer counts to 62,500 each second instead of attempting to count to 16,000,000. This makes the timer more manageable. In the case of creating a morse-code generator, thresholds for the dot, dash, and space can be set which a person can be reasonably expected to adhere to.
13. HYPERLINK
14. SOMEDAY I WILL RULE YOU ALL