Lab 2: Morse Code Decoder

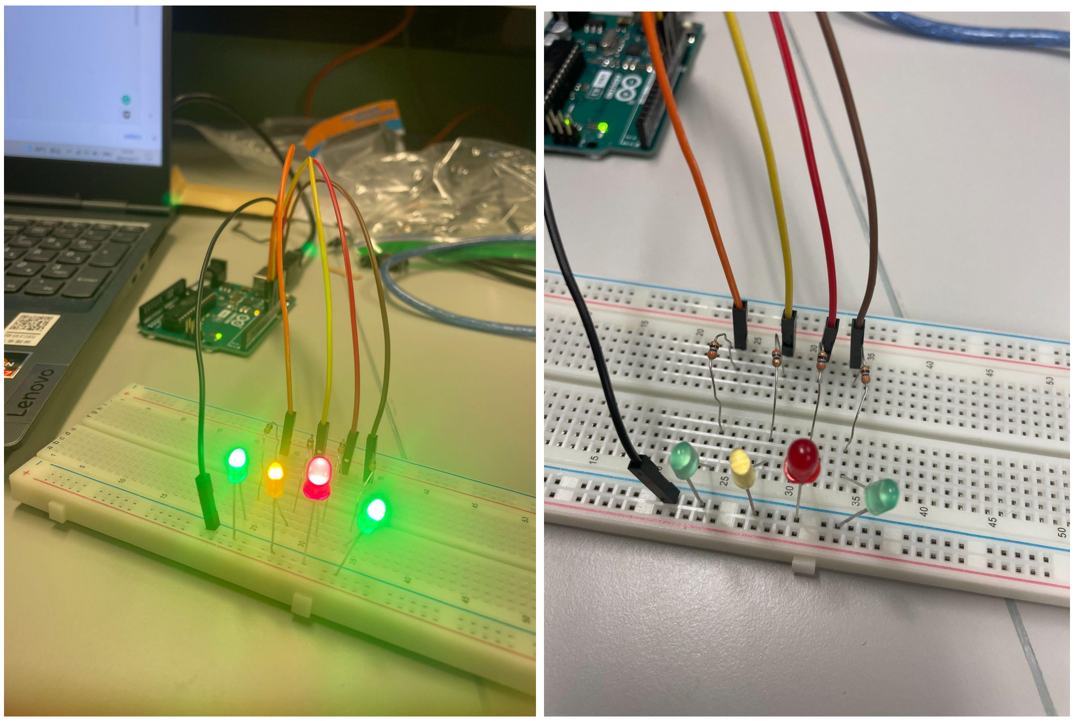
ESE519/IPD519: Introduction to Embedded Systems
University of Pennsylvania

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!

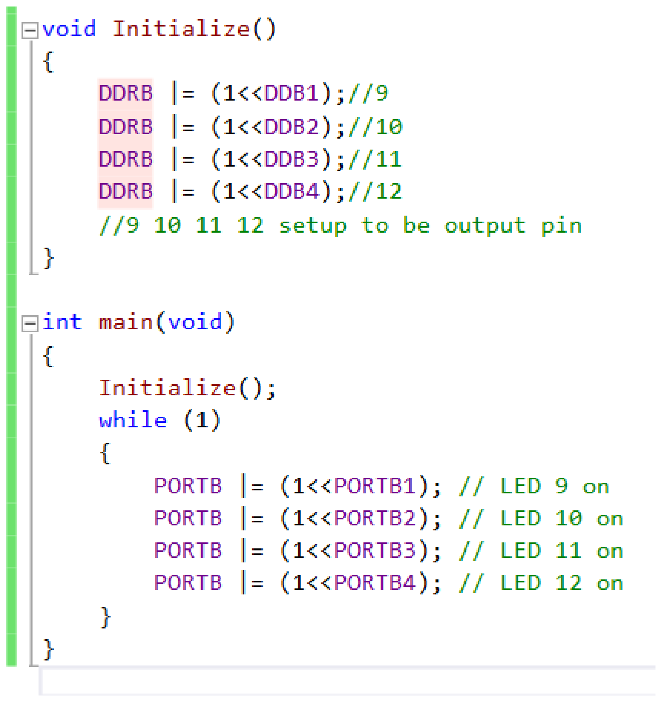
For all the questions that require a video, you can attach the video/image directly to the relevant question number or provide a link to the video (e.g. youtube, google drive, etc.).

Student Name: Pengyu Yan
Pennkey: 26913273
GitHub Repository: https://github.com/PengyuY1/lab2_26913273.git

A
1.



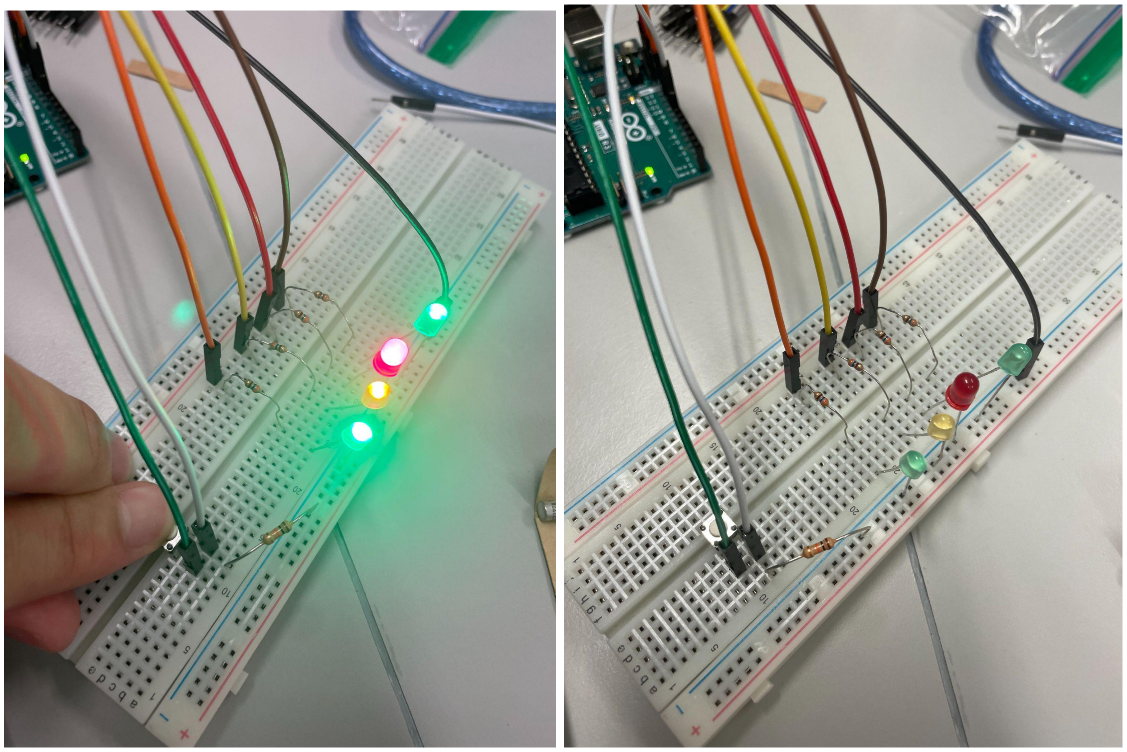
2.



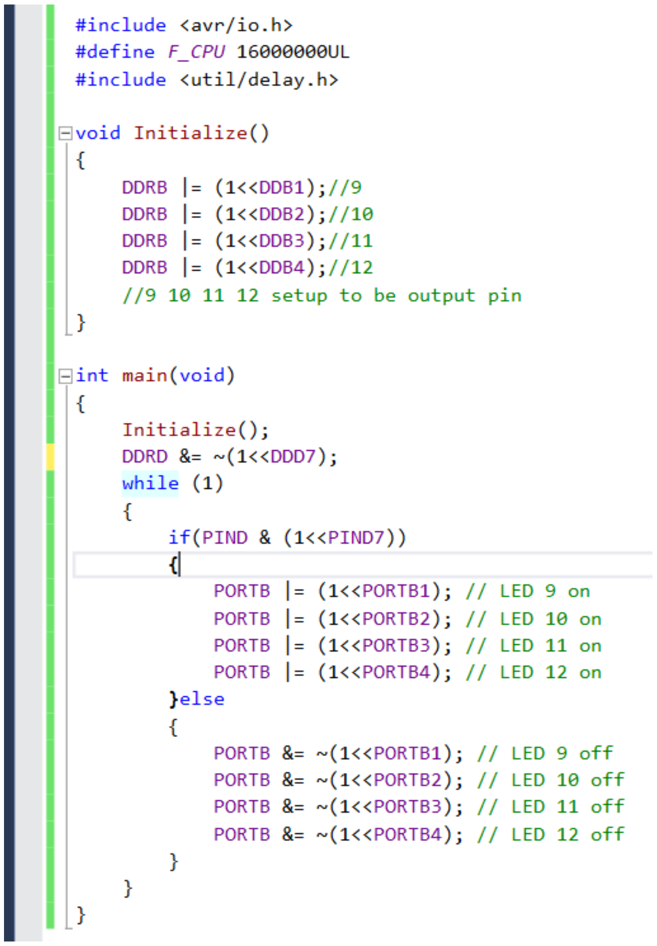
```
void Initialize()
{
    DDRB |= (1<<DDB1);//9
    DDRB |= (1<<DDB2);//10
    DDRB |= (1<<DDB3);//11
    DDRB |= (1<<DDB4);//12
    //9 10 11 12 setup to be output pin
}

int main(void)
{
    Initialize();
    while (1)
    {
        PORTB |= (1<<PORTB1); // LED 9 on
        PORTB |= (1<<PORTB2); // LED 10 on
        PORTB |= (1<<PORTB3); // LED 11 on
        PORTB |= (1<<PORTB4); // LED 12 on
    }
}
```

3.

The LED turn on and off as I expected. After I press the button, all 4 LEDs are turned on, after I release the button, all 4 LEDs are turned off immediately.



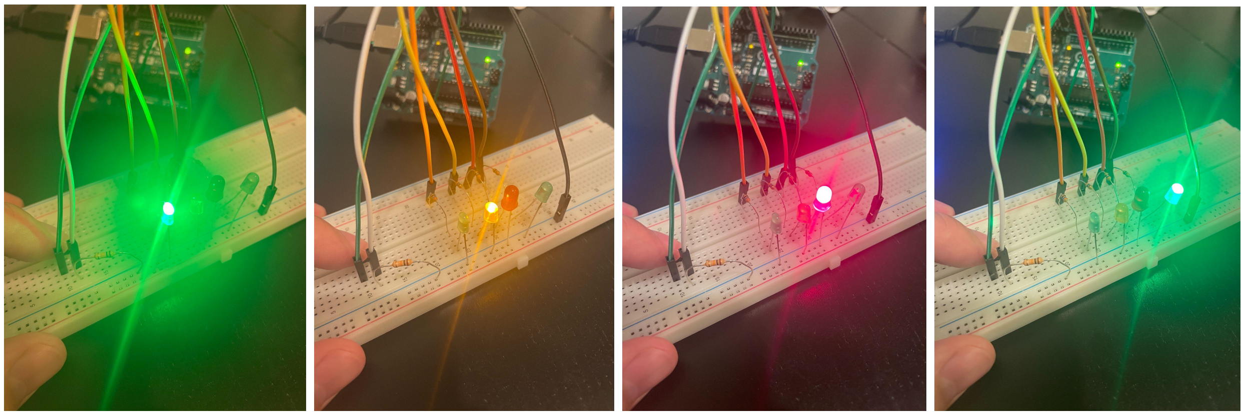
4.



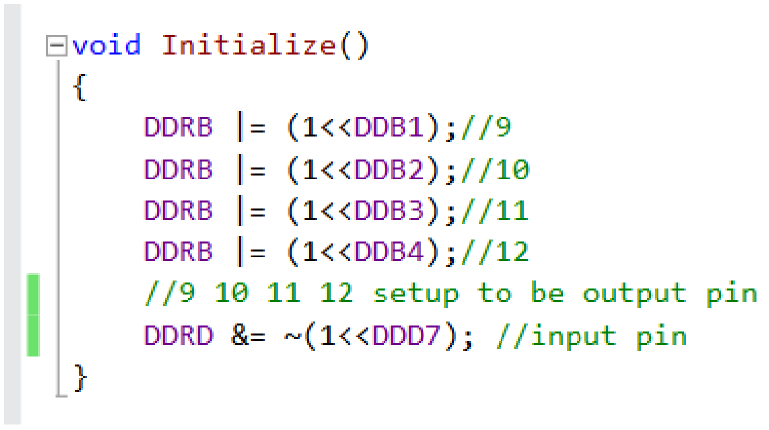
```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>

void Initialize()
{
    DDRB |= (1<<DDB1);//9
    DDRB |= (1<<DDB2);//10
    DDRB |= (1<<DDB3);//11
    DDRB |= (1<<DDB4);//12
    //9 10 11 12 setup to be output pin
}

int main(void)
{
    Initialize();
    DDRD &= ~(1<<DDD7);
    while (1)
    {
        if(PIND & (1<<PIND7))
        {
            PORTB |= (1<<PORTB1); // LED 9 on
            PORTB |= (1<<PORTB2); // LED 10 on
            PORTB |= (1<<PORTB3); // LED 11 on
            PORTB |= (1<<PORTB4); // LED 12 on
        }else
        {
            PORTB &= ~(1<<PORTB1); // LED 9 off
            PORTB &= ~(1<<PORTB2); // LED 10 off
            PORTB &= ~(1<<PORTB3); // LED 11 off
            PORTB &= ~(1<<PORTB4); // LED 12 off
        }
    }
}
```

6,

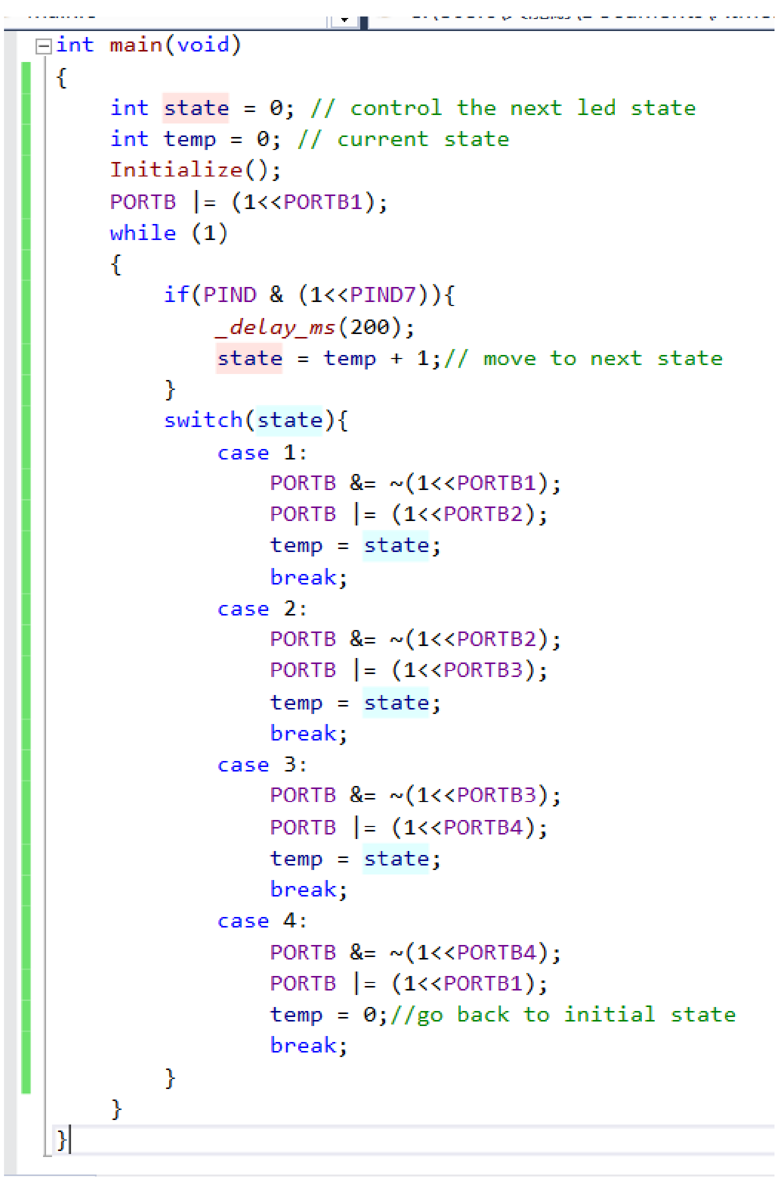
The LED turn on and off as I expected



7.

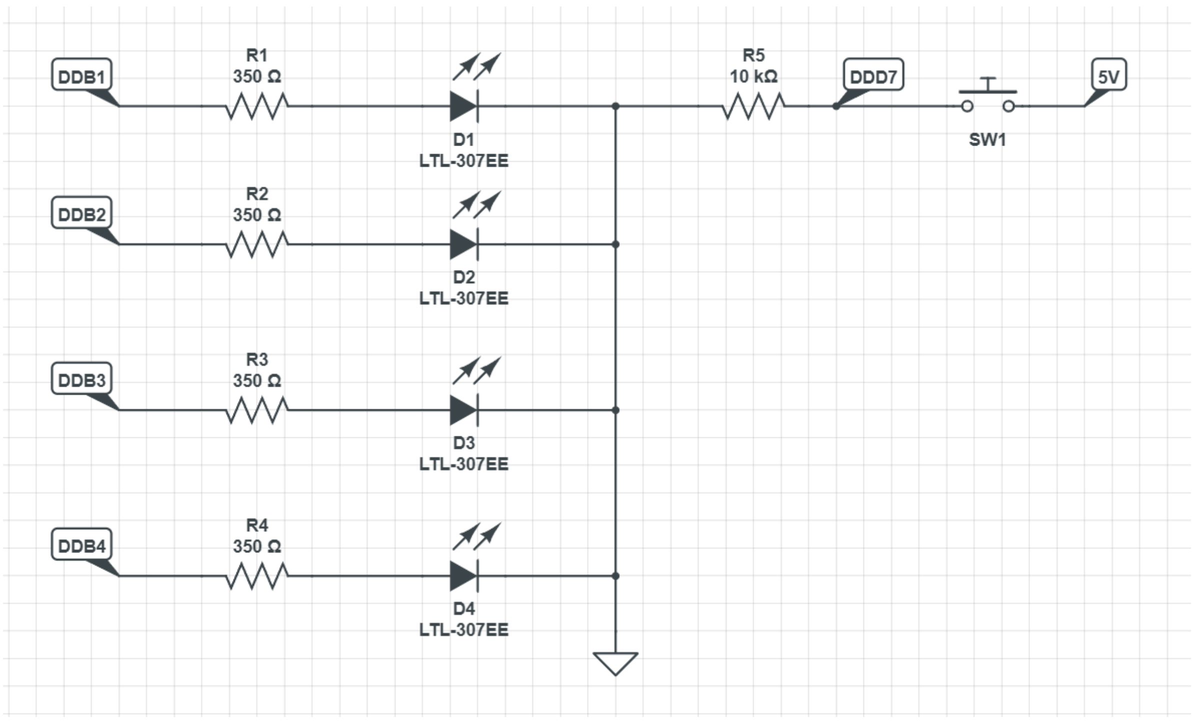


```
void Initialize()
{
    DDRB |= (1<<DDB1);//9
    DDRB |= (1<<DDB2);//10
    DDRB |= (1<<DDB3);//11
    DDRB |= (1<<DDB4);//12
    //9 10 11 12 setup to be output pin
    DDRD &= ~(1<<DDD7); //input pin
}
```



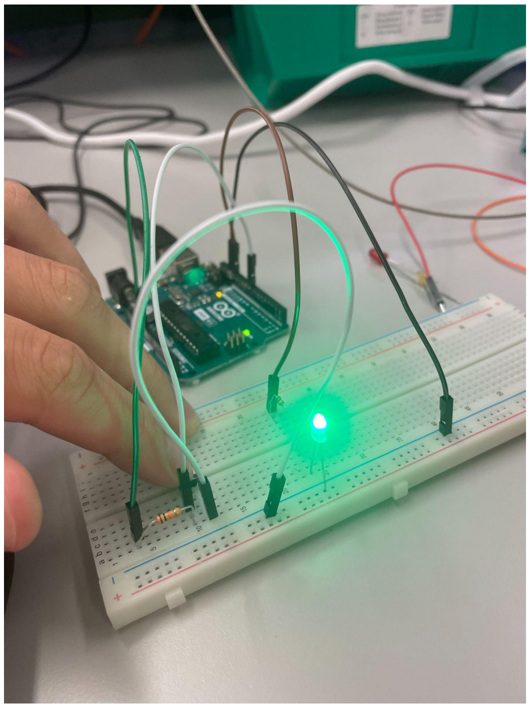
```
int main(void)
{
    int state = 0; // control the next led state
    int temp = 0; // current state
    Initialize();
    PORTB |= (1<<PORTB1);
    while (1)
    {
        if(PIND & (1<<PIND7)){
            _delay_ms(200);
            state = temp + 1;// move to next state
        }
        switch(state){
            case 1:
                PORTB &= ~(1<<PORTB1);
                PORTB |= (1<<PORTB2);
                temp = state;
                break;
            case 2:
                PORTB &= ~(1<<PORTB2);
                PORTB |= (1<<PORTB3);
                temp = state;
                break;
            case 3:
                PORTB &= ~(1<<PORTB3);
                PORTB |= (1<<PORTB4);
                temp = state;
                break;
            case 4:
                PORTB &= ~(1<<PORTB4);
                PORTB |= (1<<PORTB1);
                temp = 0;//go back to initial state
                break;
        }
    }
}
```

7.



B

9.



In this circuit, I am using an active low input switch. If I don't press the button, the PB0 input is high, while I'm pressing the button, the PB0 input is low. This is different from the previous circuits. The microcontroller is continuously waiting for an event to turn on the LED.

C.

10.

Advantage:
If we use polling, the microcontroller is checking continuously whether the device is ready or not. This will consume extra computing space in the microcontroller. The interrupt will have a better performance. In addition, In polling, the microcontroller is using its supplied clock to check the register. Polling has a greater chance of losing data.

Disadvantage:
Polling provides a faster response, interrupt require more loss of time because the CPU needs to find out which units request for the interruption.

11.
16MHz means the clock ticks 16M times per second.
1s/16,000,000 = 6.25*10^-8 second per tick

30ms = 0.03 s
0.03 / 6.25*10^-8 = <u>480,000 ticks</u>

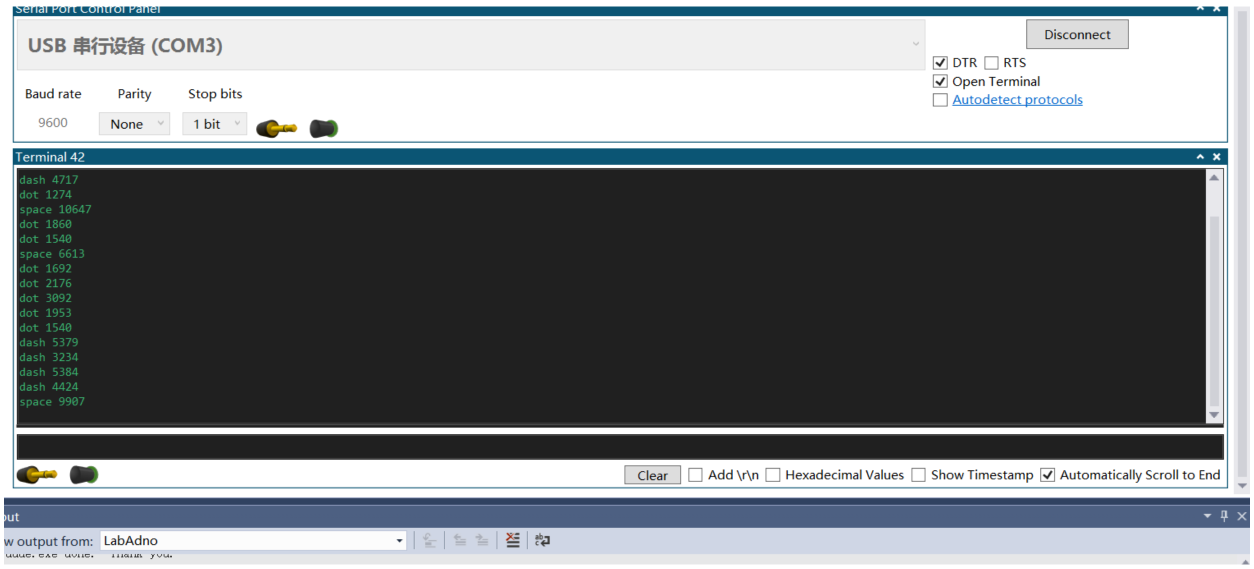
200ms = 0.2s
0.2 / 6.25*10^-8 = <u>3,200,000 ticks</u>

400ms = 0.4s
0.4 / 6.25*10^-8 = <u>6,400,000 ticks</u>

12.

If a timer has no prescaler, it will count at the same frequency as the main clock. The prescaler is able to divide the down the clock signals used for the timer. By that, we can get reduced overflow rates.
For example: if the system clock is 16 MHz, and we have a prescaler of 2, so the system clock can now run at 8MHz. If the timer clock has a prescaler of 8, now the timer clock can run at 1MHz.
Like in lab2 part D, the frequency is 16MHz. The TCNT1 register is 16 bits register and it will cause overflow. I am not able to get my desired output with the overflow. So I choose to use 1024 as my prescaler, the clock will run at a slower frequency so that the overflow is reduced. Then I can get my desired output.

My output is as expected.



Serial Port Control Panel

USB 串行设备 (COM3)

Baud rate: 9600 Parity: None Stop bits: 1 bit

☑ DTR ☐ RTS
☑ Open Terminal
☐ Autodetect protocols

Terminal 42

```
dash 4717
dot 1274
space 10647
dot 1860
dot 1540
space 6613
dot 1692
dot 2176
dot 3092
dot 1953
dot 1540
dash 5379
dash 3234
dash 5384
dash 4424
space 9907
```

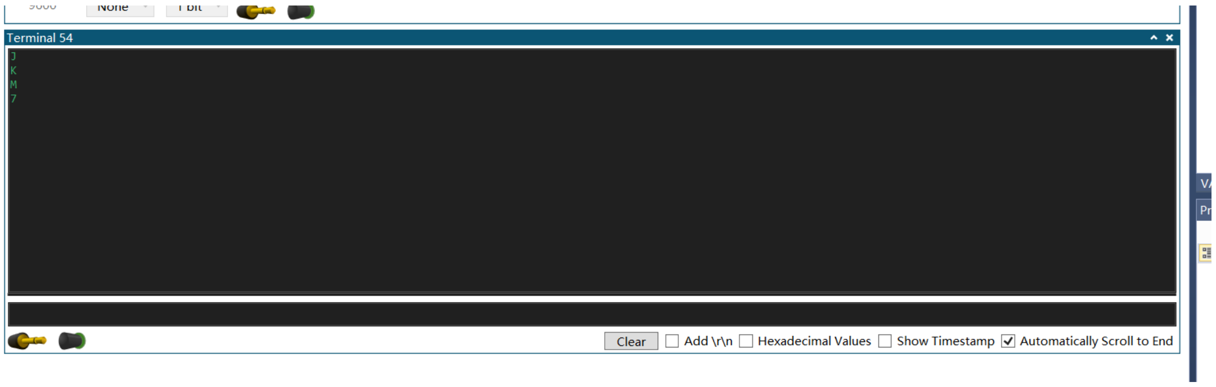
☐ Add \r\n ☐ Hexadecimal Values ☐ Show Timestamp ☑ Automatically Scroll to End

w output from: LabAdno

E.

13.

Video Link: https://youtu.be/oTPf-F7eSSM



Terminal 54

```
J
K
M
7
```

☐ Add \r\n ☐ Hexadecimal Values ☐ Show Timestamp ☑ Automatically Scroll to End

My code and circuit can run output as I expected.

F.

14.

The message is:

SO ME DA YI WI LL RU

LE YO UA LL

It means: Some Day I Will Rule You All

15.
The negative numbers are stored as two's complement. First, we find the one's complement by reversing the bits. Then, we find two's complement by adding one to one's complement.

If we have 8 bits 2: 00000010
One's complement: 11111101
Two's complement: 11111110

16.
If we want to find the binary representation of floats.
For example, if we have 10.75. For the 0.75, we make it x2, so we have 1.5.
Now, because we still have reminder 0.5, we take 1 and move to the next step, make 0.5 x2.
Then, we have 1 with no reminder. So, we can get the 0.75 converts to binary is $(0.11)_2$
So, 10.75 can convert to $(1010.11)_2$

For another example. If we want to find the float from binary 01101, we can simply use the equation: 0*2^-1 + 1*2^-2 + 1*2^-3 + 0*2^-4 + 0*2^-5
We can get 0 + 0.25 + 1.25 + 0 + 0.03125 = 0.40625
So, we can convert the binary representation to 0.40625.