

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!

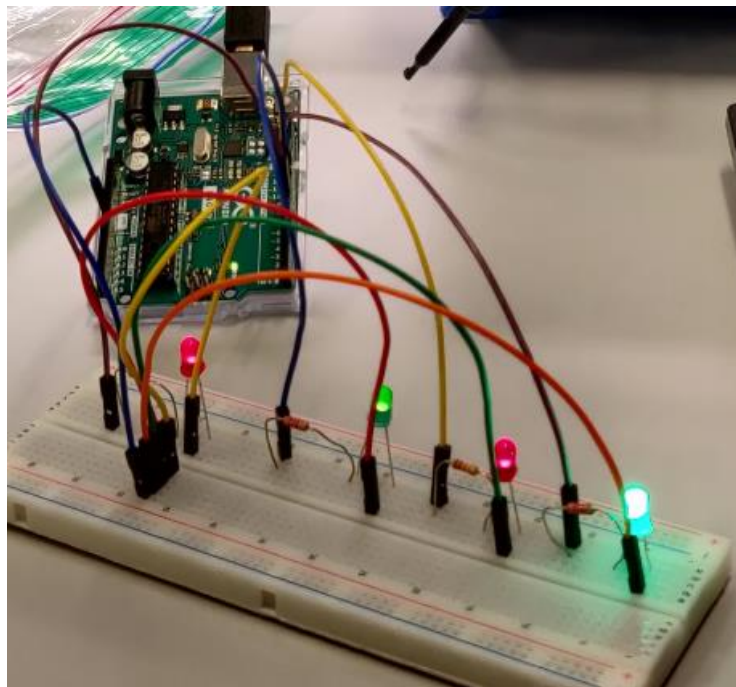
For all the questions that require a video, you can attach the video/image directly to the relevant question number or provide a link to the video (e.g. youtube, google drive, etc.).

Student Name: Adwayt Pradeep Nadkarni

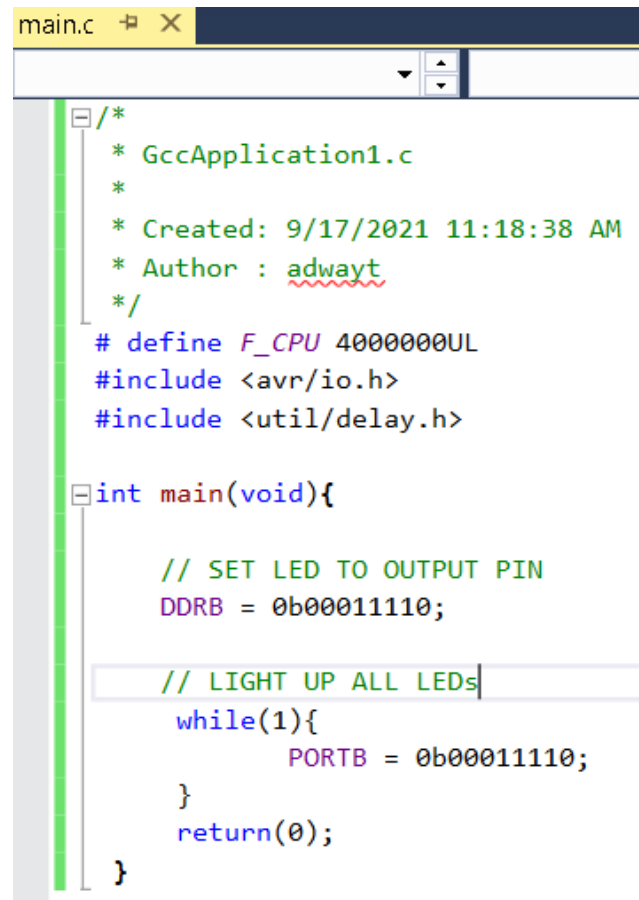
Pennkey: adwayt

GitHub Repository:

1. Hooray! You just wrote your first AVR C code! Take a picture of those fantastic LEDs with their lights on and attach it to your submission document.



2. Take a screenshot of your code. It should only be a few lines. No need to include the "include <avr/io.h>" or header file lines. Just the main function. (Refer to the examples in lecture.)



```
main.c
/*
 * GccApplication1.c
 *
 * Created: 9/17/2021 11:18:38 AM
 * Author : adwayt
 */
#define F_CPU 4000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void){

    // SET LED TO OUTPUT PIN
    DDRB = 0b00011110;

    // LIGHT UP ALL LEDs
    while(1){
        PORTB = 0b00011110;
    }
    return(0);
}
```

3. Does the LED turn on and off as expected when you press the button? If not, explain the behavior observed.

Yes. The LED turns ON and OFF as expected.

I set Button as PORTD7- input and LED as PORTB1-output. I set a condition that if my button is pressed, I give 5V to the LED.

The LED glows in response to the button being pressed ON, remains being ON when the button is ON and switches OFF immediately when the button is

OFF. No delay or dimming or any other aberrations found in the experiment. No issues were found while pushing buttons "politely".

4. Take a screenshot of your code. It should only be a few lines. No need to include the "include <avr/io.h>" lines. Just the main function. (Refer to the examples in lecture.)

```
main.c
main
int main(void)

#include <avr/io.h>

int main(void){

    // SET LED TO OUTPUT PIN
    DDRB = 0b00000010;

    DDRD &= ~(1<<DDRD);

    // LIGHT UP ALL LEDs
    while(1){

        if( PIND & (1<<PIND7) ){
            PORTB = 0b00000010; }

        else{
            PORTB = 0b00000000;
        }

    }

    return(0);
}
```

5. <MANUAL DOES NOT MENTION A QUESTION 5.>

6. Does the LED turn on and off as expected when you press the button? If not, explain the behaviour observed.

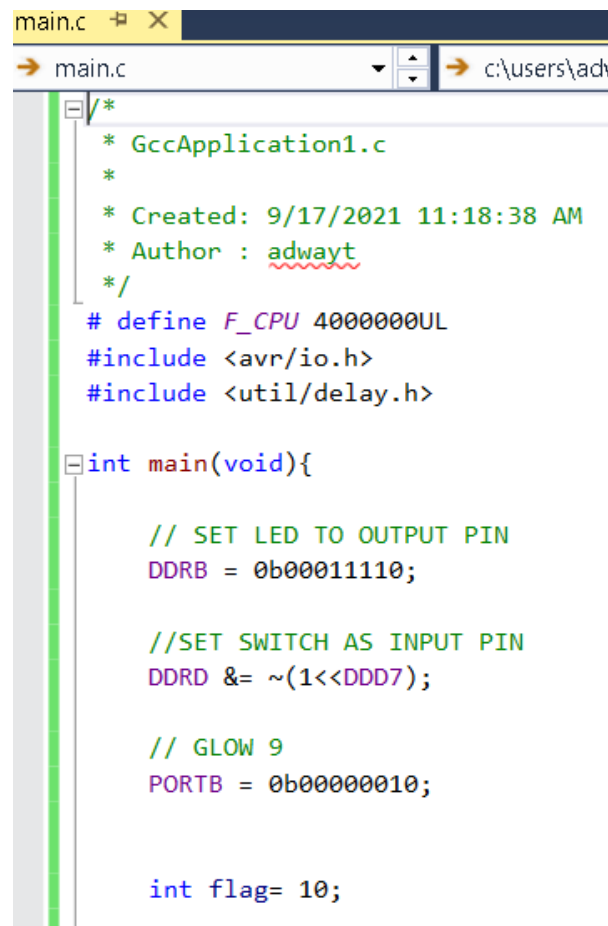
Yes. The LED turns ON and OFF as expected when I press the button.

The motion is cyclical. It goes from PIN 9-> PIN 10 -> PIN 11 ->PIN 12 and back to PIN 9 in arduino or (PORTB1, PORTB2, PORTB3, PORTB4 OUTPUT pins respectively). A button on PORTD7 is in input mode. I also have a variable that shows me which LED I'm with currently. I iterate this variable everytime I send 5V

to a pin. I also make provisions to ensure the variable does not over shoot and remains between 9 and 12, the numbered pin used in this experiment.

No delay or dimming or any other aberrations found in the experiment. The experiment was successful.

7. Take a screenshot of your code. It should only be a few lines. No need to include the "include <avr/io.h>" lines. Just the main function. (Refer to the examples in lecture.)



The screenshot shows a code editor window titled 'main.c' with a file path 'c:\users\ad...'. The code is as follows:

```
main.c
main.c c:\users\ad...

/*
 * GccApplication1.c
 *
 * Created: 9/17/2021 11:18:38 AM
 * Author : adwayt
 */
#define F_CPU 4000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void){

    // SET LED TO OUTPUT PIN
    DDRB = 0b00011110;

    //SET SWITCH AS INPUT PIN
    DDRD &= ~(1<<DDD7);

    // GLOW 9
    PORTB = 0b00000010;

    int flag= 10;
```

```

while(1){

    if (flag>12){flag = 9;}

    if ( (PIND & (1<<PIND7)) && flag==9){

        // 9 ON AND 12 OFF
        _delay_ms(2000);
        PORTB = 0b00000010;
        _delay_ms(2000); // Delays 2000ms (2s)
        flag = flag+1;

    }

    if ( (PIND & (1<<PIND7)) && flag==10){

        // 10 ON AND 9 OFF
        _delay_ms(2000);
        PORTB = 0b00000100;
        _delay_ms(2000); // Delays 2000ms (2s)
        flag = flag+1;

    }

    if ( (PIND & (1<<PIND7)) && flag==11){

        // 11 ON AND 10 OFF
        _delay_ms(2000);
        PORTB = 0b00001000;
        _delay_ms(2000); // Delays 2000ms (2s)
        flag = flag+1;

    }

    if ( (PIND & (1<<PIND7)) && flag==12){

        // 12 ON AND 11 OFF
        _delay_ms(2000);
        PORTB = 0b00010000;
        _delay_ms(2000); // Delays 2000ms (2s)
        flag = flag+1;

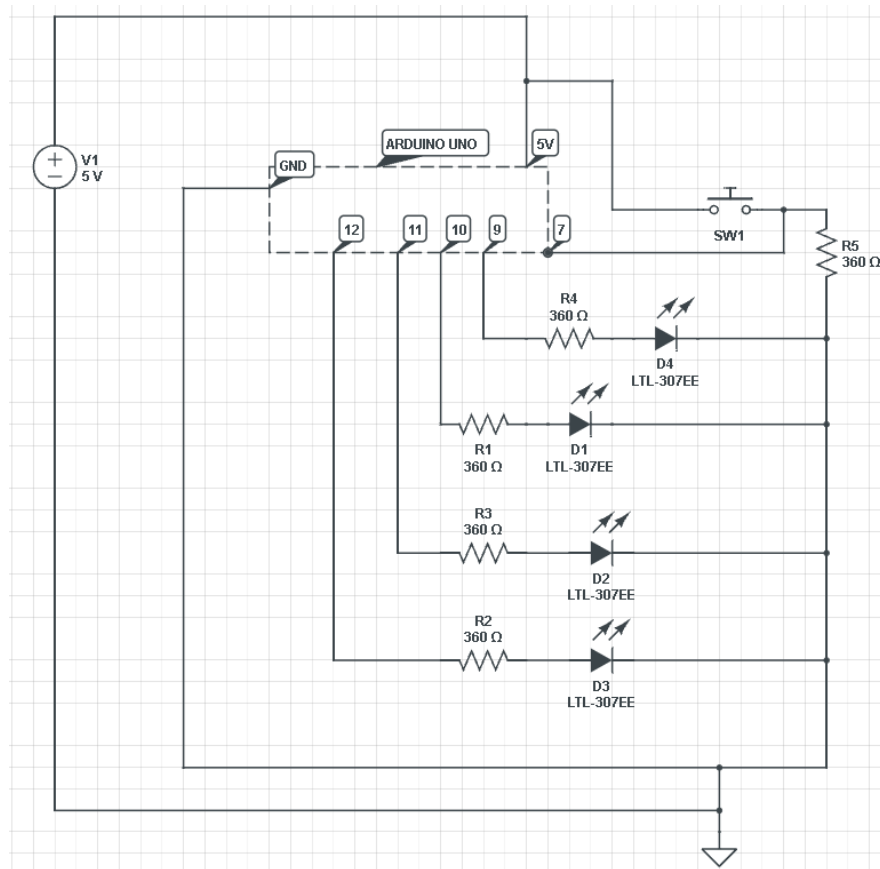
    }

}

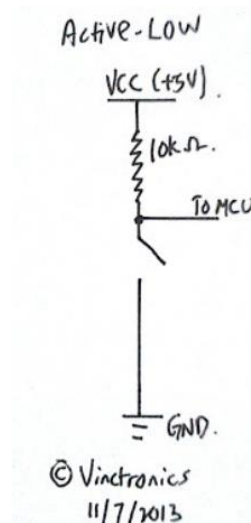
return(0);
}

```

8. Use [CircuitLab](#) to draw your schematic. Attach an image to your answer to this question.



9. Describe your approach to implementing polling and how this differs from the approach of simply reading the pin in the previous part.



Source: [Vinctronics: Active-HIGH Vs Active-LOW Button \(vinduino.blogspot.com\)](http://vinctronics.blogspot.com)

Polling is the process in which the compiler waits for external device to check it can perform a process or not. In our situation, we wait for an I/O operation to trigger a change in LED. The button in this experiment is now connected as active LOW (as per question) (source provided to the diagram above. It does not belong to me). The button will remain in 5V when open, when we close the button, it forms a closed circuit with ground and voltage falls to 0V. We see here that it is exactly the opposite of we performed for the previous question. (We waited for the button to be HIGH or have 5V to perform operation) We wait for button to be 0V to perform operation.

In my code. I have retained the nomenclature. I just added a not operation on my conditional statements. The experiment worked successfully.

10. Give an advantage and a disadvantage to using interrupts over polling for this task.

Advantage of Interrupt over Polling:

- Performance of microcontroller for interrupt is better than polling.
- Data loss are greater in polling than interrupts.
- In polling device needs to ask every loop if device is ready.

Disadvantage of Interrupt over Polling:

- Difficult to debug interrupt program
- Software does not plan for interrupt like polling. CPU may ignore other processes to take up interrupts.
- Interrupts are more complex
- Interrupts are at a risk of being non-synchronous with rest of the program.

11. For a 16MHz clock, how many "ticks"/steps are in 30ms, 200ms, and 400ms?

SUBMITTED BY: ADWAYT PRADEEP NADKARNI

(11) Given:
System Clock = 16MHz

Req:-
30ms, 200ms, 400ms

Ans - I have taken $N=256$ in my code.

Formula:

$$\text{Count (Ticks)} = \left(\frac{\text{CLK}_{\text{sys}}}{N} \times \frac{1}{f_{\text{off}}} \times \frac{1}{2} \right) - 1$$

① 30ms -
 $f_{\text{off}} = 1/30\text{ms}$

$$\text{Count} = \text{Ticks} = \left(\frac{16\text{M}}{256} \times \frac{1}{1/30\text{ms}} \times \frac{1}{2} \right) - 1$$

$$= \left(\frac{16\text{M} \times 30\text{ms} \times 1000}{256 \times 2} \right) - 1$$

$$= 936.5 \approx \underline{\underline{937 \text{ ticks}}}$$

② 200ms -

$$\text{Ticks} = \left(\frac{16 \times 200 \times 1000}{256 \times 2} \right) - 1$$

$$= \underline{\underline{6249 \text{ ticks}}}$$

PENNY KEY: adwayt

③ 400ms -

$$\text{Count} = \left(\frac{16 \times 400 \times 1000}{256 \times 2} \right) - 1$$

$$= 12499 \text{ ticks}$$

12. Describe how a prescaler allows us to work with a wider range of frequencies on our microcontroller.

A prescaler allows us to work with wider range of frequencies and dictates the speed of the timer also. 8-bit timers can store maximum counter of 255 and 16 bit timer, 65535. Even if we set the register to maximum value, interrupts will happen at 16us for 8 bit and 4ms for 16 bit counter. So we can use prescaler vary and create different frequencies depending on prescaler value.

Interrupt Frequency (Hz) = (Arduino Clock or System clock) / (Prescalar)

Once System clock is modified the modified Frequency will now be given to all peripherals connected the system clock. An individual timer can further be prescaled, so the Operating frequency can be reduced even further.

Hence we can work with wider range of frequencies on our microcontroller.

13. Take a short video demonstrating your code or do a live demo to a member of the teaching team. If you demoed it lived to a member of the teaching team then no need to attach a video.

Codes

c - <https://drive.google.com/file/d/1LNtbcUeXaN1uVWbghV1so-7z913BE6dn/view?usp=sharing>

d -

<https://drive.google.com/file/d/13oxaj0nyzyi9I3XAs1dKLWzkrfWm2LEQ/view?usp=sharing>

e -

https://drive.google.com/file/d/1EW4pKiP_3Dbe6O_ghRsacCVXcI2VR1Zh/view?usp=sharing

b -

<https://drive.google.com/file/d/1KlcyWzo4cbTLfqMTHzRwA7lBy99aooG1/view?usp=sharing>

MORSE CODE VIDEO:

YouTube Channel : Adwayt Nadkarni

14. What message did Jason in the comic on the first page tap out?

SOMEDAY I WILL RULE YOU ALL

15. Signed and unsigned integers are represented a bit differently in binary. Explain how negative numbers are represented in binary.

Signed integers are those data types which allow their integers to have signs. If the number is represented by "n" bits, then the most significant bit (MSB) is used to represent a sign of the number. They can be used to represent numbers where signs are absolutely necessary like temperatures, etc. By default int in C is a signed integer. Signed int is represented as 1's compliment and 2's complement.

The integer is represented by an unsigned binary number whose MSB is 0. It can store from 0 to positive values only. If all you care is the magnitude, unsigned int is better than signed int. Unsigned int is used in embedded systems. They are represented by as is without special provision for signage.

16. Explain how floats are represented in binary.

Floating point arithmetic is arithmetic that uses real numbers for representation. In floating-point computation, a fixed number of significant digits are used and scaled using an exponent in a fixed base.