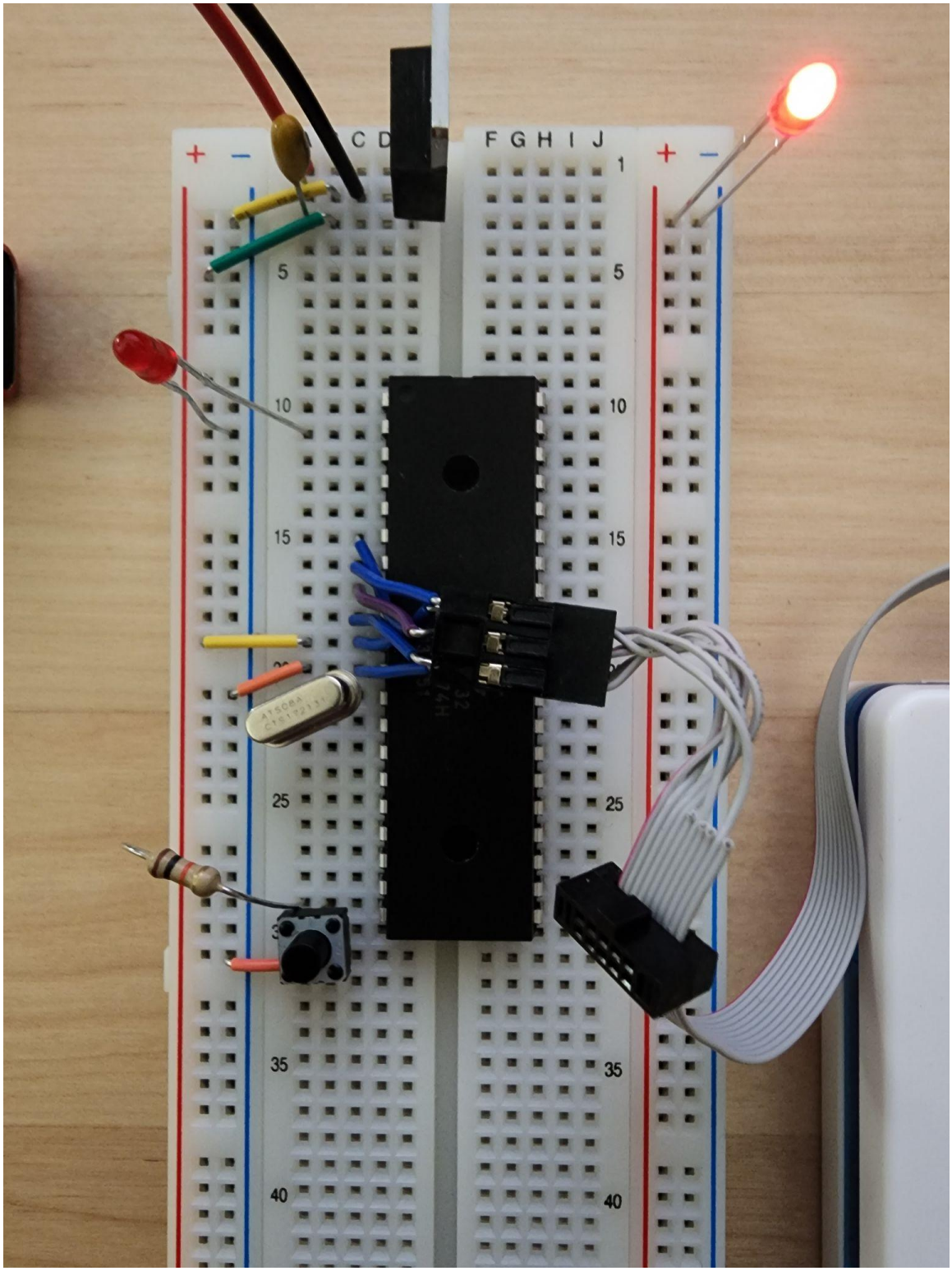
CSE 145 Lab #1

Steven Tabilisma
16326339

4/20/22

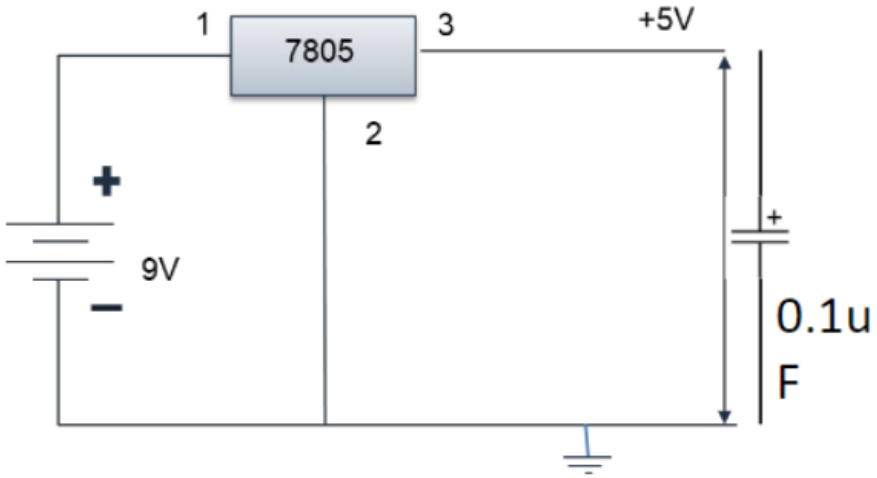


Introduction

The objective of this lab was to use the ATMEGA32 microcontroller to create a circuit that blinks a singular LED in half second intervals when input is received from pressing a button. To break this lab down into digestible parts I would divide it into three sections: Power, ATMEGA32 setup, and I/O setup.

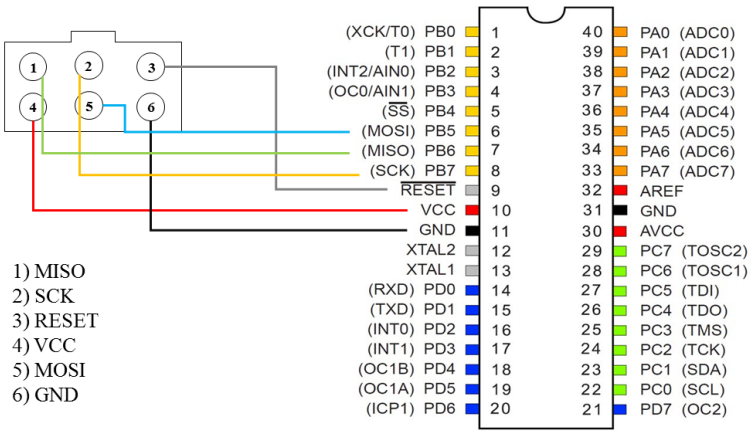
Power

Powering the microcontroller and components required us to regulate the power of a 9V battery into the operating voltage range of 2.4V-5V for the microcontroller. The location of this implementation on my breadboard is in the top left corner. As you can see, this is simply done by using the linear regulator. Using the wires from the battery, positive goes to pin 1 of the regulator, negative goes to point 2, and pin 3 connects to the positive section of the power rail. A capacitor is connected to the same row as pins 2 and 3, and the column of pin 2 is directly connected to the negative section of the power rail. With this we have provided 5V of power to the left rail. Below is a circuit showing how the battery and linear regulator are connected.



ATMEGA32 Setup

In order to control the LED using the button, we needed to install the ATMEGA32 and set it up to be programmed. After plugging the programmer into a PC using the provided type-A to type-B cable and then connecting the 10 pin to 6 pin cable to the AVR port, we simply need to reference the data sheet of the 6 pin connector and the ATMEGA32.



After this is wired the last step is to power the microcontroller. As seen in the diagram pin 10 is the voltage connection and pin 11 is ground. After wiring these to the correct sections of the power rail the IDE should properly ID the device and receive a proper 5V of power.

I/O Setup

The most intuitive progression for this section is to start by turning on the LED by addressing the GPIO of the microcontroller, make the LED blink, and then install the button and use it to start the blinking. Once this works we simply go into the fuse settings and switch to enable the use of the external crystal. Below is the final code I ended up writing.

```
int main(void)
{

    DDRB = 0xff;
    DDRD = 0x00;
    while (1)
    {
        if (GET_BIT(PIND,6) == 0) {
            avr_wait(500);
            PORTB = 0xff;
            avr_wait(500);
            PORTB = 0x00;
        }
        //PORTB = 0xff;
        //_delay_ms(100);
        //PORTB = 0x00;
        //_delay_ms(100);
    }
}
```

Summarized simply, I enable the entirety of Port B as input and set all of Port D to input. Port B is the top 7 pins on the left hand side and Port D is the bottom 7. After declaring the directions of these two ports, we enter an infinite loop that polls for input from the button. While this input is being received the output of Port B cycles between on and off with a delay of 500ms in between.

LED

Since we set all of the pins on Port B to be output it does not matter which of the open 5 we connect to the positive side of the LED. After choosing one, the negative side of the LED simply needs to connect to the ground of the power rail.

Button

In order to connect the button, one pin of the button must be connected to a decided pin of Port D. In this image I have chosen PIND6. In the same row, a 10k Ohm resistor must connect the positive section of the power rail. On the other pin of the button, simply we simply wire the row to the ground rail. Now when the button is pressed the power being supplied to PIND6 is shorted resulting in zero voltage being received by the pin. Hence why we check for equivalence to zero in the if conditional.

Timer Crystal

The timer crystal is very simple as it is not a polarized component. In the two consecutive rows directly after the pins for the programmer we connect the two pins of the timer crystal. In the fuse settings of Microchip studio we go to the drop down the very last item in the list and choose the setting that corresponds with the timer crystal we installed. In this case it is the very last option, “Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms”.

NOTE:

Although I included avr.c and avr.h files, they are unnecessary to run the main.c file. Rather than using an include statement I have copied the necessary code from the avr.c/h files and pasted it inside of the main.c file. This is because I was unable to properly implement the avr files into the solution hierarchy.