Jared Rice

EE 333

Lab 1 Microcontroller Basic Operation

Overview:

In this lab we will build a breadboard-based microcontroller using the ATmega328p and configuring the Arduino IDE to program our microcontroller. To test if our microcontroller is operating as intended, we will run a basic blink sketch on our breadboard microcontroller and on an Arduino Uno. We will run our blink sketch at different frequencies and use an oscilloscope to confirm our LED blinks at the intended rate. In addition, the extra credit section below demonstrates implementing the blink sketch written purely in C code, where compilation, linking, and downloading were all performed via command line.

Procedure:

The ATmega328p microcontroller was built onto a breadboard following the Jameco Build your own Arduino Circuit guide. Below in figure 1 is the completed microcontroller running the blink sketch.

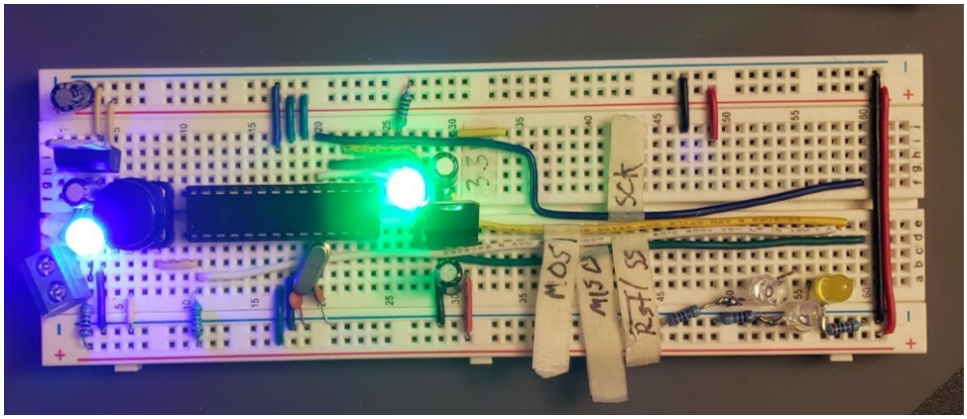

Figure 1: Breadboard Microcontroller

Programming for the ATmega328p was performed with an Arduino Nano as an ISP rather than the FTDL programmer. The Nano was connected to the ATmega328p via SPI, as shown in figure 2, and programmed with the ArduinoISP example sketch.

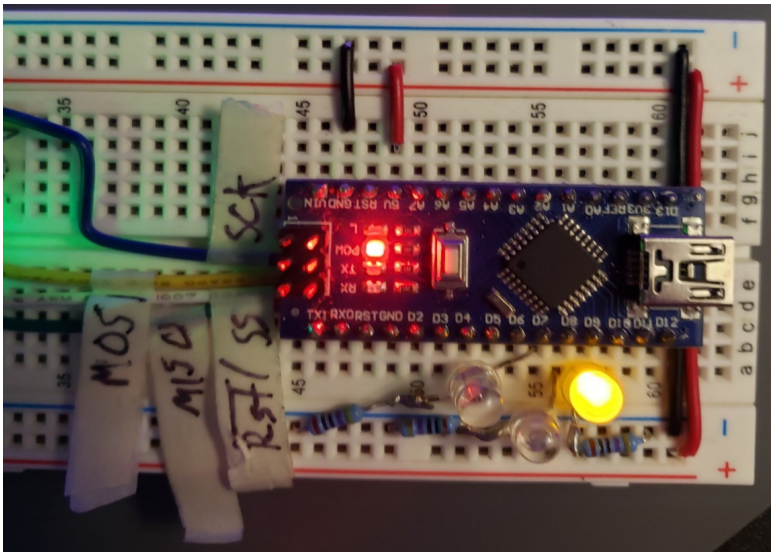

Figure 2: Arduino Nano functioning as an ISP

To test the ATmega328p, after programming the Nano was removed from the breadboard and the ATmega328p was externally powered as shown above in figure 1. The reset button was also tested and functioned as intended. Per the lab instructions, a #define was added as shown in code excerpt 1 below. Figure 3 below shows the output of pin 19 of the ATmega328p toggling every second, figure 4 shows it toggling every 100 ms and figure 5 shows it toggling every 10 ms.

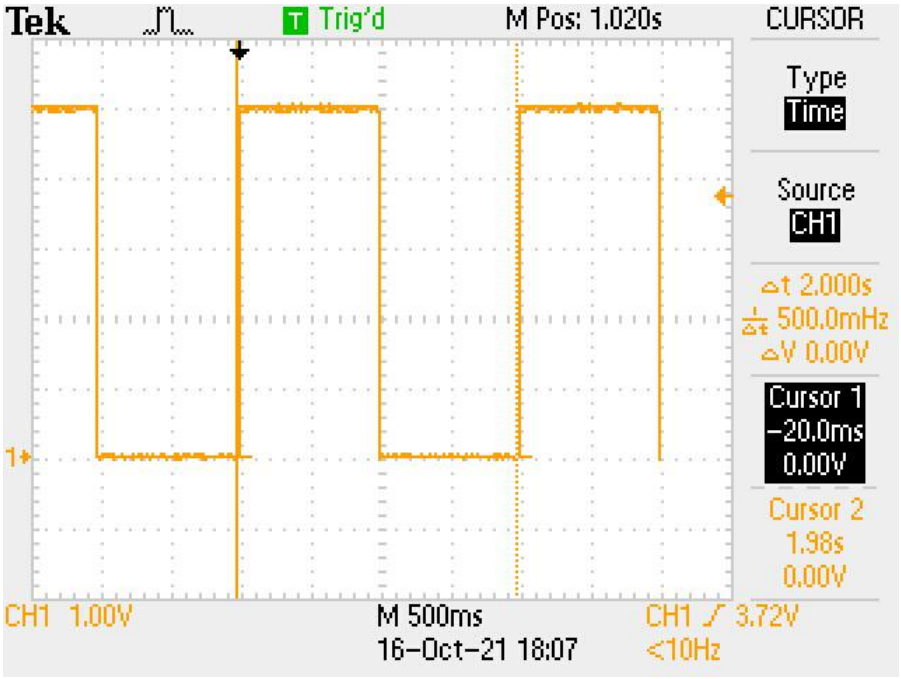

Figure 3: Breadboard ATmega328p Toggling Every Second

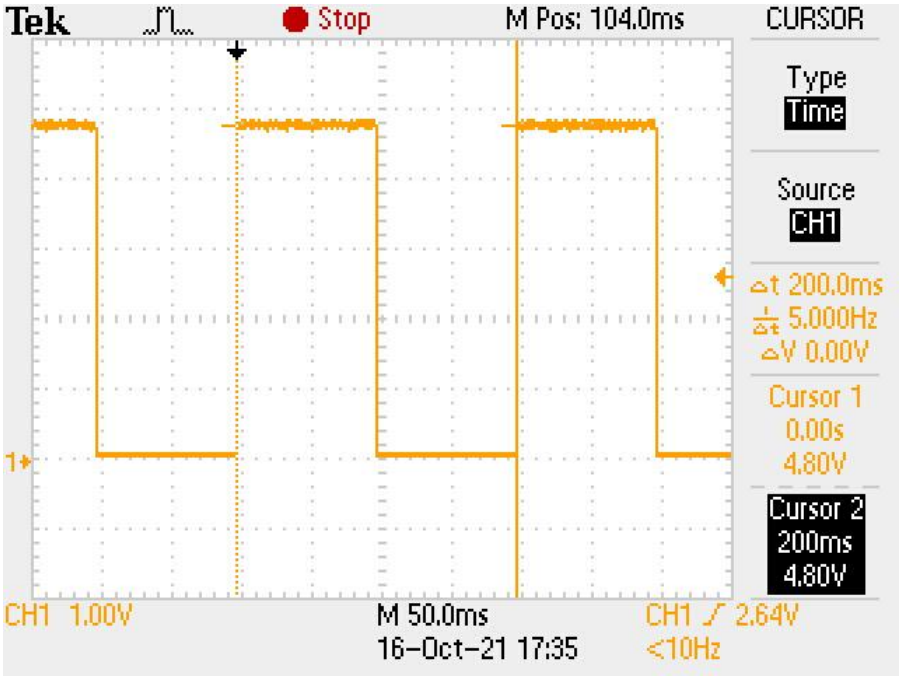

Figure 4: Breadboard ATmega328p Toggling Every 100 ms

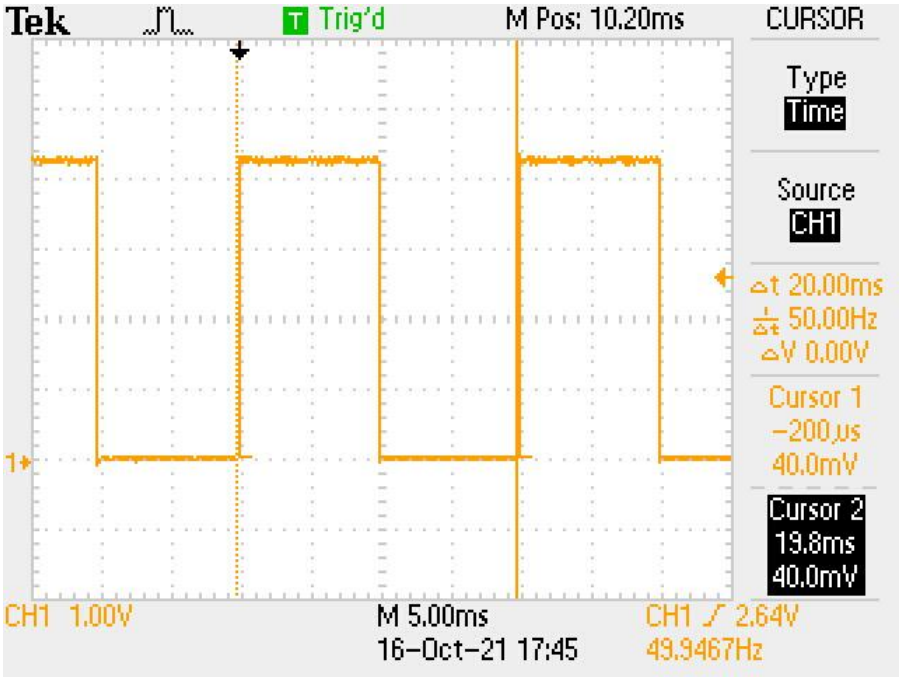

Figure 5: Breadboard ATmega328p Toggling Every 10 ms

After programming our breadboard ATmega328p, An Arduino Uno was programmed via the Arduino IDE, with the same blink sketch used in the earlier section. The running code can be seen in figure 6.

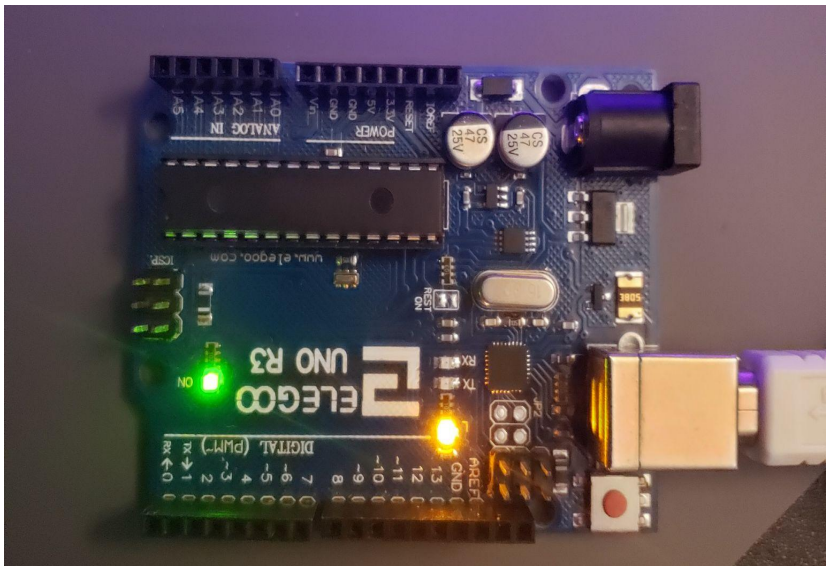

Figure 6: Arduino Uno running the blink sketch

Photos were captured with an oscilloscope showing output of pin nine toggling every second in figure 7, every 100 ms in figure 8, and every 10ms in figure 9.

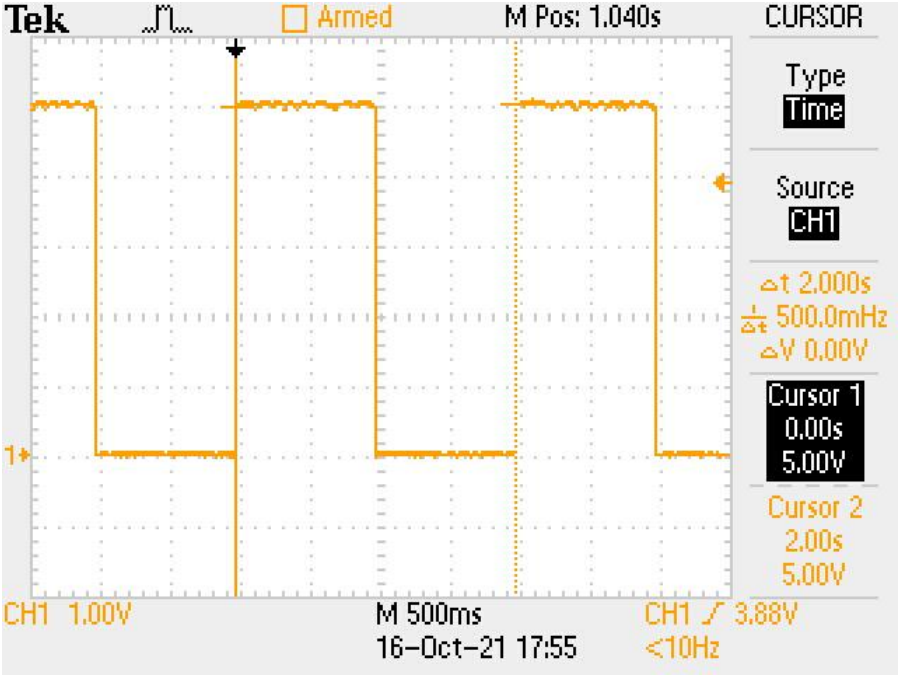

Figure 7: Arduino Uno Toggling Every Second

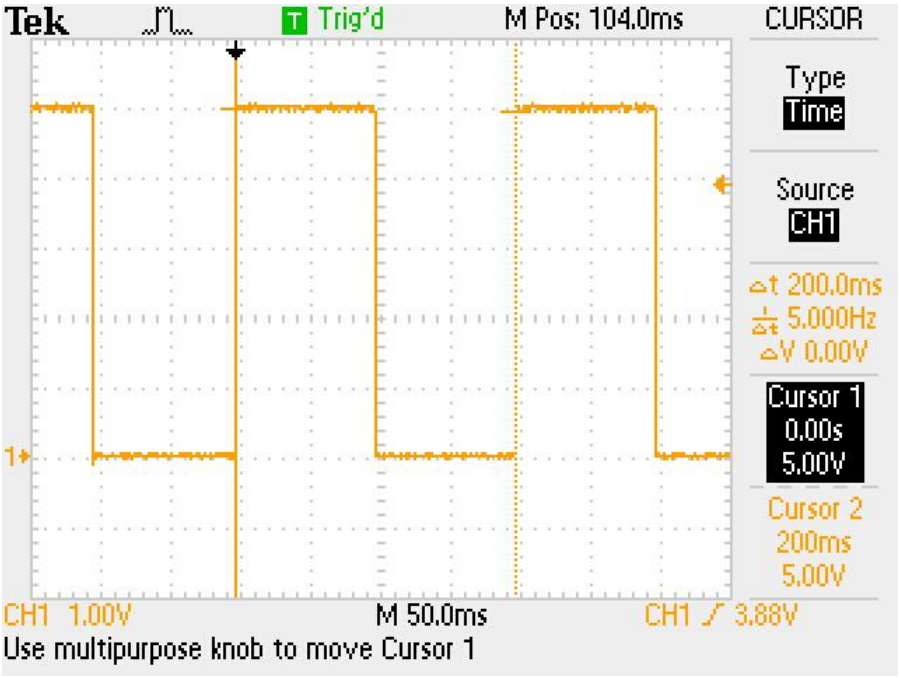

Figure 8: Arduino Uno Toggling Every 100 ms

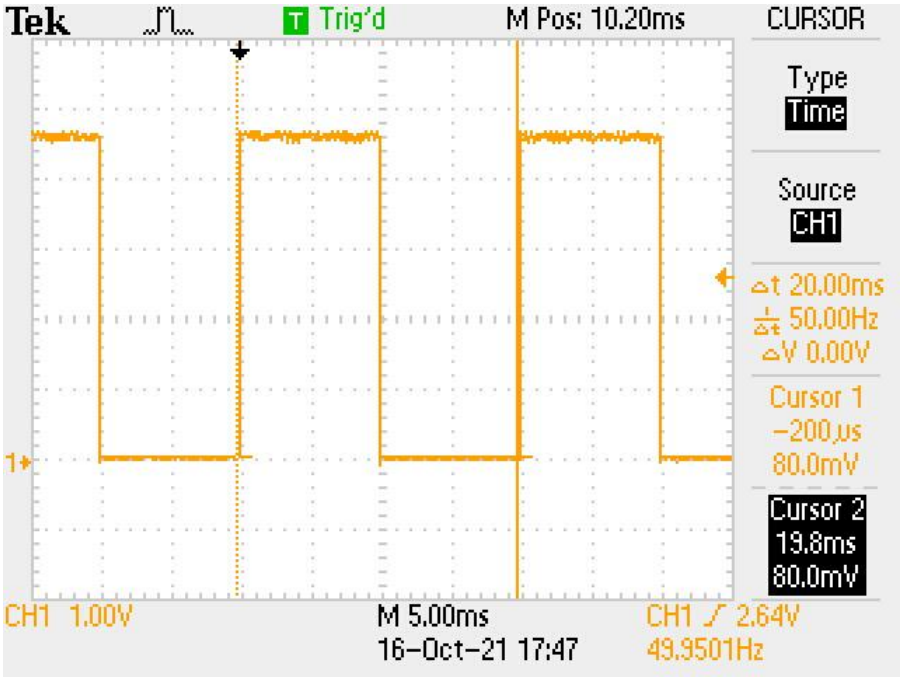

Figure 9: Arduino Uno Toggling Every 10 ms

The main functions used in our blink sketch were pinMode, digitalWrite, and delay. The three are described below. The pinMode function is a two parameter function used to define the operational characteristics of one of the pins on our microcontroller. The first parameter takes an Arduino pin number which declares which pin we wish to modify. The second parameter sets that pin as either an input, output, or an input with a builtin pullup resistor. The digitalWrite function is a two parameter function that operates differently depending on whether the pinMode function sets the desired pin to an output or an input. The first parameter is the Arduino pin that you wish to modify. The second parameter is a value, either HIGH or LOW. If the pin is set to an input, setting the value to HIGH enables the internal pullup resistor, and setting a value LOW disables the internal pullup resistor. If the pin is configured as an output, sending a LOW value sets the desired pin to a logical low (0v), and sending a HIGH value sets the pin to a logical high (5v or 3.3v).The delay function takes one parameter, a time in milliseconds. It pauses the program for the time passed as its argument.

```
#define DELAY_TIME 1000

// the setup function runs once when you press reset or power the board

void setup() {

    // initialize digital pin LED_BUILTIN as an output.

    pinMode(LED_BUILTIN, OUTPUT);

}

// the loop function runs over and over again forever

void loop() {

    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the
   voltage level)

    delay(DELAY_TIME);                 // wait for a second

    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the
   voltage LOW

 delay(DELAY_TIME);                    // wait for a second

}
```

Code Sample 1

Summary

Working on this lab in a Linux environment created a handful of challenges. Moving between programming via the Arduino Nano as an ISP and the USB interface of the UNO triggered a bug in the Arduino IDE. When switching between the AVR ISP and the Arduino as ISP, It showed the correct programmer, but uploading code still failed. Google thankfully pointed me towards someone else with the same problem. To get around this required simply toggling to a third programmer, and then reselecting the programmer I needed.

The most interesting thing I learned during this lab was the command line programming via avrdude and avr-gcc. I have prior experience with programming on the Arduino platform, and have worked some with the ATtiny series of microcontrollers on breadboards. This was the first time I've programmed a microcontroller without the IDE, and learning the basics of setting up the needed toolchain was interesting.

Our code is uploaded into the 32kB of Flash memory in the ATmega328p, this flash memory is non-volatile and persists after power is cycled.

Extra Credit

Programming Atmega328p from the command line with avrdude was done with presupplied code from
https://balau82.wordpress.com/2011/03/29/programming-arduino-uno-in-pure-c/

This code written entirely in C shown in Code Sample 2 below functioned identically to the example blink sketch, toggling an led every second. The two header files needed for this code are the avr/io.h and util/delay.h. avr/io.h has all the predefined macros relevant to our ATmega328p. It converts terms like DDRB into the specific values relevant to our microcontroller during compilation. In the case of DDRB, the macro converts it into the specific memory address of our data direction register B. The util/delay.h header is used to give us access to the delay ms function, which allows us to delay our function for a set amount of time in milliseconds. The bitwise operators are used to toggle individual bits in our registers.

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

#include <util/delay.h>

#define BLINK_DELAY_MS 1000


int main (void) {

   /* set pin 5 of PORTB for output*/
```

```
    DDRB |= _BV(DDB5);


    while(1) {
    /* set pin 5 to turn led on */
    PORTB |= _BV(PORTB5);
    _delay_ms(BLINK_DELAY_MS);


    /* set pin 5 to turn led off */
     PORTB &= ~_BV(PORTB5);
    _delay_ms(BLINK_DELAY_MS);
    }
}
```

Code Sample 2

In the linux environment used for this lab, a bash shell script is used instead of a batch file. The shell script used to compile and upload code to the breadboard ATmega328p and Arduino Uno is shown below in Code Sample 3. Using an AVR pocket programmer to upload code to both microcontrollers allowed us to use only one shell script for both.

```
#!/bin/bash
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o led.o led.c
avr-gcc -mmcu=atmega328p led.o -o led
avr-objcopy -O ihex -R .eeprom led led.hex
avrdude -F -V -c usbtiny -p ATMEGA328P -b 115200 -U flash:w:led.hex
```

Code Sample 3

Avr gcc is our c compiler, the options used are -Os to optimize the code size, -DF_CPU specifies the clock speed at 16 megahertz, -mmcu specifies the chip

architecture, -c compiles our source files without linking them, and -o specifies the output object file to create.

Avr objcopy is used to modify the object file we created during compilation and turn it into a hex file we can upload into our microcontroller. For avr objcopy, the options used are -O to specify the output object file to write, -R removes any sections of our file that contain .eeprom before outputting our hex file.

Avrdude is a command line programmer used to upload our hex file to our microcontrollers. The options used are -F to ignore the device signature check, -V disables verification when uploading the code, -c specifies the programmer used, -b selects the baud rate used to upload the code, -U writes to flash led.hex.