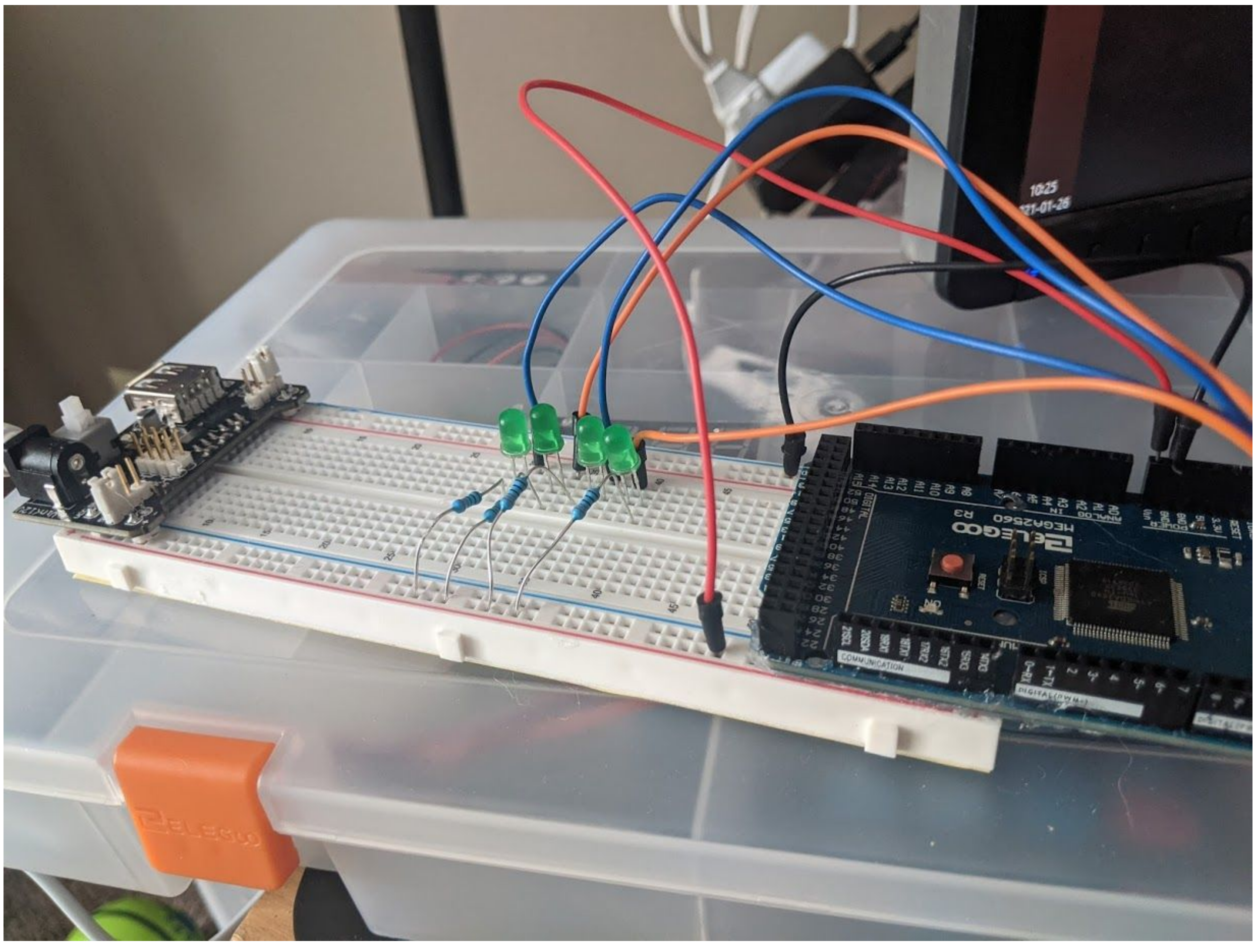
Lab 1 Report

Setup & 4-bit Binary Counter

CENG 347 Embedded Intelligent Systems



Samuel Donovan & Samantha Pfeiffer

2021-01-26

OVERVIEW

The purpose of this lab was to install the Arduino IDE, and set up a simple 4-bit binary counter with LEDs.

--- ARDUINO IDE ---

The Arduino IDE can be downloaded from https://www.arduino.cc/en/software. Versions are available for Windows, Linux and Mac OS X.

In addition to the IDE, the Java Runtime Environment is also needed. This can be downloaded at https://www.java.com/en/download/.

The cross-platform availability of both of these softwares enables easier troubleshooting between groups and with the professor should issues arise. Both group members installed these on machines running Windows 10.

🔶 One group member encountered an error when attempting to launch the IDE, stating that `_JAVA_OPTIONS xmx8192M caused an invalid heap size`. This was fixed by deleting the `_JAVA_OPTIONS` environment variable in Windows.

One group member also installed a dark theme for the IDE, as to not blind themselves every time they went to code. The source files and instructions for doing so can be found at https://create.arduino.cc/projecthub/konradhtc/one-dark-arduino-modern-dark-theme-for-arduino-ide-2fca81

Figure 1 Windows Arduino IDE, Default Theme

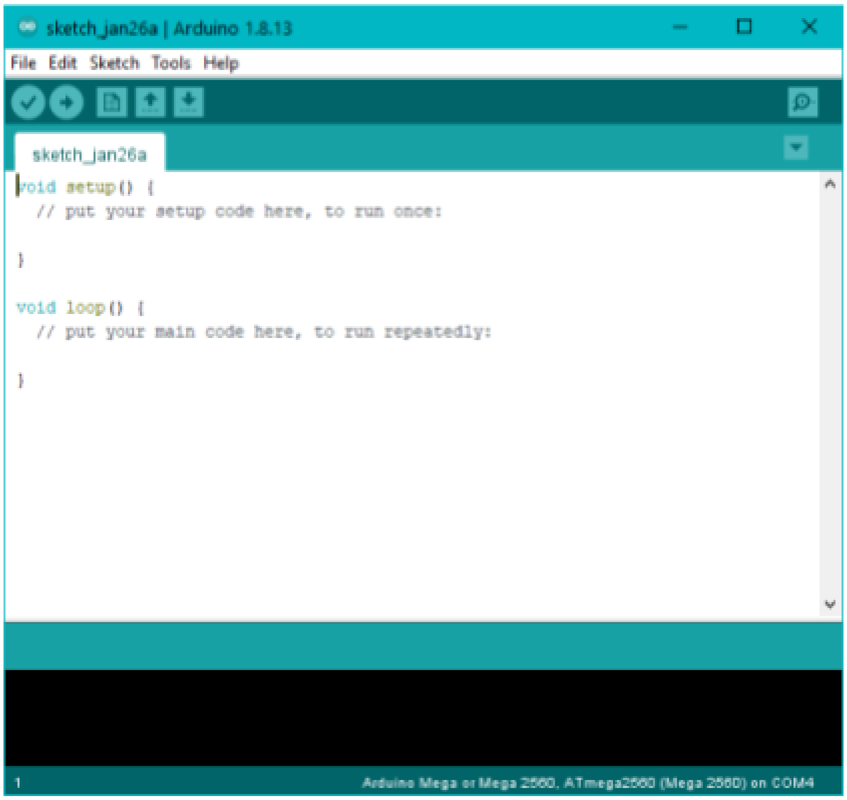
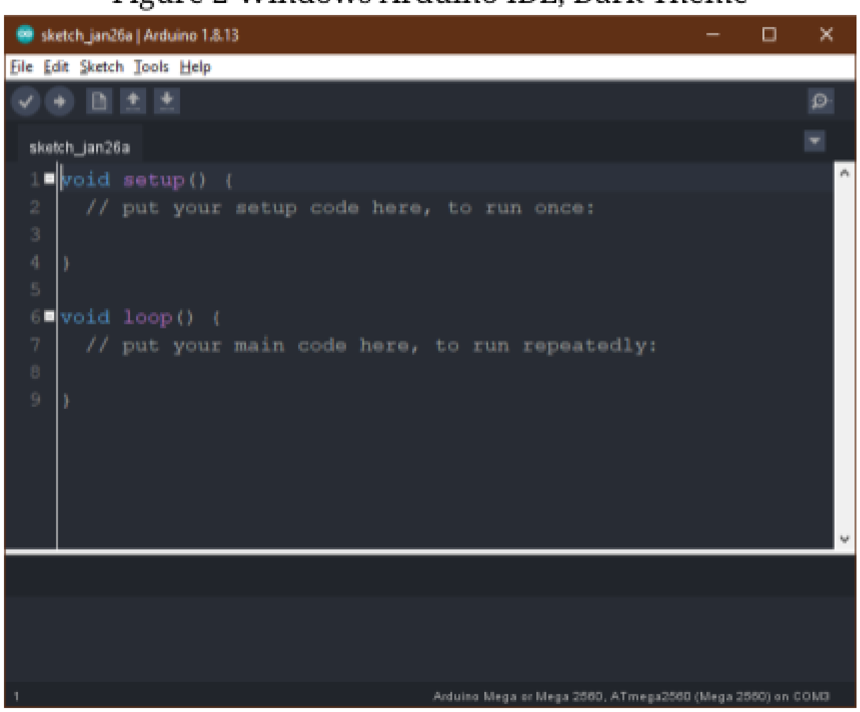


Figure 2 Windows Arduino IDE, Dark Theme



--- 4-BIT BINARY COUNTER ---

A 4-bit binary counter is used to express a base-2 number between 0000_2 and 1111_2 (0_{10} to 15_{10}). Table 1 shows each bit value as it corresponds to its decimal counterpart. The table also shows the inverse logic values of those bits; the need for these values will be explained in the next section.

Table 1 4-Bit Binary Counter Truth Table

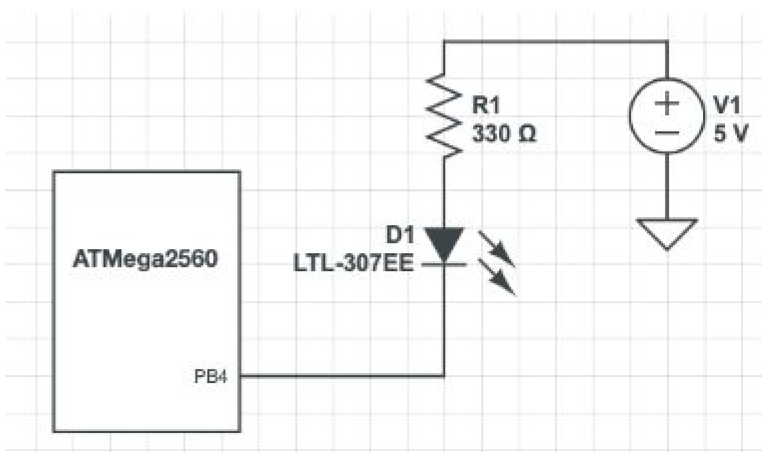
Decimal Value	B_3	B_2	B_1	B_0	$\sim B_3$	$\sim B_2$	$\sim B_1$	$\sim B_0$
0	0	0	0	0	1	1	1	1
1	0	0	0	1	1	1	1	0
2	0	0	1	0	1	1	0	1
3	0	0	1	1	1	1	0	0
4	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	0	1	1	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	0	1
11	1	0	1	1	0	1	0	0
12	1	1	0	0	0	0	1	1
13	1	1	0	1	0	0	1	0
14	1	1	1	0	0	0	0	1
15	1	1	1	1	0	0	0	0

METHODS

--- CIRCUITRY ---

The 4-bit binary counter was created using an ATMega2560 controller board, an 830 tie-point breadboard, a power supply module, four LEDs, and four 330 Ohm resistors. Each LED represented a binary bit and would light up to indicate a bit value of 1.

Since microcontrollers are not meant to directly drive LEDs, each anode was connected to a positive 5v source and each cathode to PORTB of the microcontroller. The counter was created using the high nybble of PORTB, so pins PB7, PB6, PB5, and PB4 were used. In order to current limit, a resistor was connected in series between each LED and the power supply. An example of this configuration was provided in the lab write up and can be seen below.



This configuration creates inverse logic. When a pin is set to 0, the LED is forward biased by the +5V source, and therefore lit. Setting that pin to 1 removes the voltage drop, no longer forward biasing the LED, therefore turning it off. Thus, the ~B values of Table 1, were used to create the correct blinking pattern.

--- CODE ---

Since this lab was completed separately, following are descriptions of each team member's code.

PFEIFFER

In the approach shown in **Appendix A**, the code consists of a series of 8 Exclusive OR statements and bitmasks in an infinite loop. Initially, 16 assignment statements were used to manually assign the bits values from 0 to 15. However, this code was rewritten to be more efficient by utilizing the pattern in which the 4 bits flip. B_0 flips between every number, B_1 every 2, B_2 every 4, and B_3 every 8. Thus, this pattern repeats every 8 numbers, and only 8 statements are needed to count from 0-15.

DONOVAN

In the approach shown in **Appendix B**, the code was first modularized into two functions, `setPinMode` and `setPinState`. While this approach is perhaps more verbose and not as efficient, it was taken due to its legibility and maintainability.

First in `main`, ***only*** the pins needed are set to be outputs, and their initial states are set to 1 (OFF). After this, an infinite loop is entered that repeatedly increments a variable `n` from 0-15 in 1-second intervals. Each time `n` is incremented, the necessary bits are extracted and the respective pins set to the corresponding state.

RESULTS

Both methods explained above resulted in a 4-bit binary counter. Each counter started at 0 and counted to 15 before repeating, as expected. Though each team member accomplished this task in different ways, the results were identical. For more results, see the video files submitted with this report.

APPENDIX A: 4-bit Binary Counter C Code (PFEIFFER)

```
//Filename: CENG347_Lab1.c
//Written By: Sam Pfeiffer
//Confid: AVR ATMega2560 @16MHz
//Description: A 4-bit binary counter using PB7, PB6, PB5, and
// PB4. Counting is done by a series of Exclusive OR bit masks.

// Includes //
#include <avr/io.h>
#include <util/delay.h>

int main()
{
  DDRB = 0xF0;
  PORTB = 0xFF;

  while(1)
  {
    PORTB ^= 0x10;
    _delay_ms(900);

    PORTB ^= 0x30;
    _delay_ms(900);

    PORTB ^= 0x10;
    _delay_ms(900);

    PORTB ^= 0x70;
    _delay_ms(900);

    PORTB ^= 0x10;
    _delay_ms(900);

    PORTB ^= 0x30;
    _delay_ms(900);

    PORTB ^= 0x10;
    _delay_ms(900);

    PORTB ^= 0xF0;
```

```
    _delay_ms(900);
  }
}
```

APPENDIX B: 4-bit Binary Counter C Code (DONOVAN)

```
/*   Filename : LED.c
 *   Author(s): Samuel Donovan
 *   Config: ATmega2560 @16MHz
 *   Description: 4-Bit binary counter, using Port B Pins 4-7.
Although unoptimized, the code was written to be legible.
 */


#include <avr/io.h>
#include <util/delay.h>

void setPinMode(volatile unsigned char &DDR, unsigned char PIN,
unsigned char MODE)
{
  if(MODE == 0)
    DDR &= ~(1 << PIN);
  else
    DDR |= 1 << PIN;
}

void setPinState(volatile unsigned char &PORT, unsigned char
PIN, unsigned char STATE)
{
  if(STATE == 0)
    PORT &= ~(1 << PIN);
  else
    PORT |= 1 << PIN;
}

int main()
{
  //Setup
  setPinMode(DDRB, 7, OUTPUT);
```

```
  setPinMode(DDRB, 6, OUTPUT);
  setPinMode(DDRB, 5, OUTPUT);
  setPinMode(DDRB, 4, OUTPUT);
  setPinState(PORTB, 7, 1);
  setPinState(PORTB, 6, 1);
  setPinState(PORTB, 5, 1);
  setPinState(PORTB, 4, 1);
  char n = B0000;


  while(true)
  {
    setPinState(PORTB, 7, ~((n >> 3) & 1));
    setPinState(PORTB, 6, ~((n >> 2) & 1));
    setPinState(PORTB, 5, ~((n >> 1) & 1));
    setPinState(PORTB, 4, ~(n&1));

    if(n == B1111)
      n = B0000;
    else
      n++;

    _delay_ms(1000);
  }
}
```