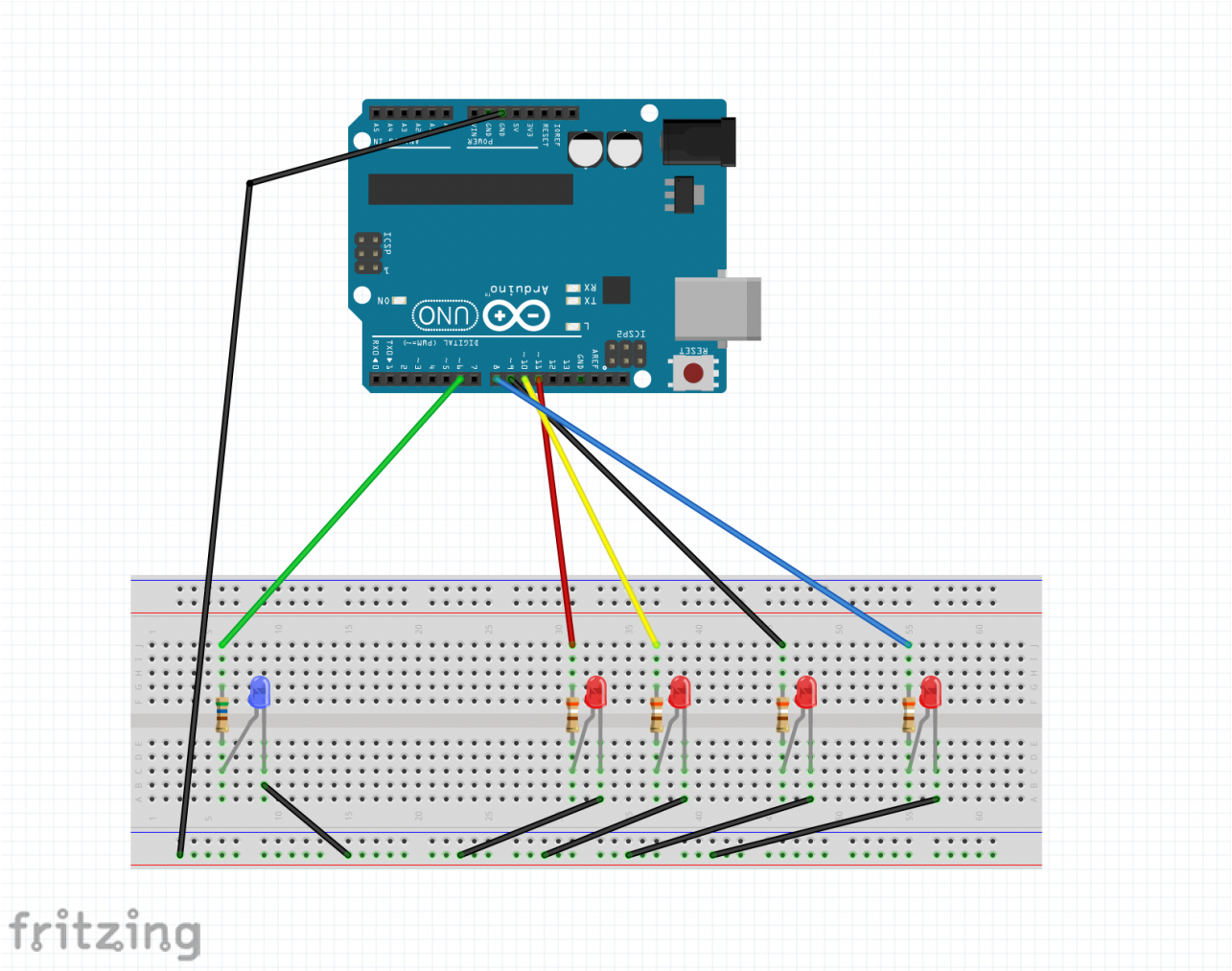
Binary PWM Counter Report

Description:

My project is an extension of our lab using the PWM to pulse and blink an LED. I expanded on it by counting every time the pulsing blue LED blinks using four other LEDs. They count the amount of blinks by lighting up in such a way that they represent the binary number equivalent to the amount of blinks counted. Since we have limited LED's and I want it to repeat, I made four counter LEDs, which count up to 15 blinks before resetting back to zero.

Design:



The design requires:

- 11 wires
- 4 red LED
- 1 blue LED

- 5 resistors 220 ohm or a little higher

Code/Explanation:

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

static int duty_cycle = 200; //Duty cycle starting point of 200
static int delta = 20; //Delta is the amount the duty cycle will change every intterupt
int binary_num = 0;//Binary num is a counter for the current number to be converted to binary
```

Here we have the basic setup of our major variables. Duty cycle represents the current duty cycle for the PWM on the blue LED. It starts at 200. Delta represents the amount we change the duty cycle by for each timer interrupt. Binary_num represents the current number of blinks the counter has tracked.

```
//Binary check takes the current binary number and sets the ports accordingly
//(I WANTED TO MAKE THIS BASED ON AN ARRAY WITH MATH LIKE PROJECT 3 BUT I COULDN'T FIGURE IT OUT FOR BARE METAL)
void binary_check(int bin)
{
  int number = bin;

  if(number == 0){
    PORTB = 0b00000000;
  }
  else if(number == 1){
    PORTB = 0b00000001;
  }
  else if(number == 2){
    PORTB = 0b00000010;
  }
  else if(number == 3){
    PORTB = 0b00000011;
  }
  else if(number == 4){
    PORTB = 0b00000100;
  }
  else if(number == 5){
    PORTB = 0b00000101;
  }
```

Here we have the first part of binary_check(). This function takes the binary_num value and sets the active output of the red LEDs accordingly. I wanted to make this more efficient and mathematical using a method similar to lab 3 using an array, but I couldn't figure out a way to make it work in bare metal c++. It's just a lot of if and else if statements but there probably is a way to make it work in a simpler manner (I was just short on time).

```
//ISR for the compare match A interrupt of Timer1.
ISR(TIMER1_COMPA_vect) // Interrupt Service Routine for overflow mode
{
  duty_cycle = duty_cycle + delta; //Every time we hit the interrupt add delta to the duty cycle

  //If duty cycle is higher than max of OCR0A (LED BLINK)
  if (duty_cycle >= 0xFF ){

    //Increase the binary counter if within range
    if(binary_num<15){
      binary_num = binary_num+1;
    }
    //Reset when it gets too high
    else{
      binary_num = 0;
    }
    delta = -delta; // reverse direction: increasing <--> decreasing duty cycle
  }
  //When duty cycle is zero
  if (duty_cycle == 0){
    delta = -delta; // reverse direction: increasing <--> decreasing duty cycle
  }

  //Once duty cycle ,delta , and binary_num are updated
  binary_check(binary_num);//Call binary_check to set the binary counter LEDs
  OCR0A = duty_cycle; //Set brightness of pulsing LED using PWM duty cycle
  _delay_ms(100);
}
```

This is the main code that controls everything related to the ISR behavior. Basically every time there is a timer interrupt from COMPA, we add the value of delta to the duty cycle value. If it ends up being higher than our PWM can take (a blink), we add one to the binary counter. Unless it is 15, then we reset the counter to zero.

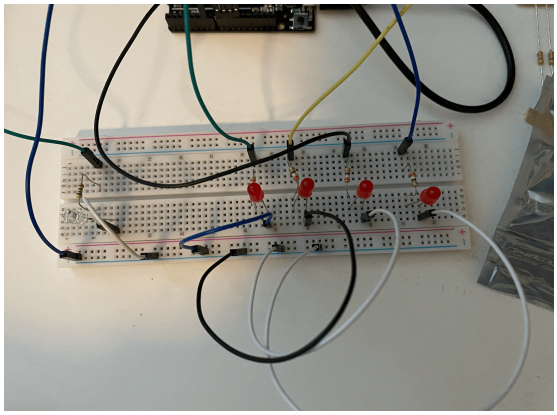
After changing the binary value, we invert the value of delta to be negative compared, which will slowly lower the duty cycle. If the duty cycle reaches zero we set delta to be positive and start increasing the duty cycle again. This allows us to pulse the blue LED back and forth forever.

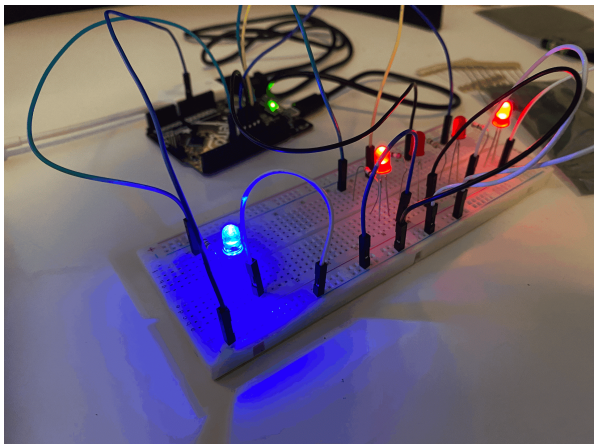
After all the values have been corrected, we pass the binary_num to binary_check() to set the red LEDs and OCR0A to the current duty cycle.

```
//MAIN FUNCTION
int main(){
  cli(); //Clear interrupt data
  DDRD = (1<<6); //Set port D6 as output
  DDRB = 0b00001111; // Set pins 8-11 as output pins
  PORTB = 0b00000000; //set the pin logic to 0 // pull up register inactive

  OCR0A = 10; //Controls PWM
  TCCR0A = 0b10000011;
  TIMSK0 = 0b00000000; //No need for interrupt
  TCCR0B = 0b00000100;

  //Controls separate timer for interrupt
  OCR1A = 47999; //What point the comparison happens
  TCCR1A = 0x00; //No pin behavior
  TCCR1B = 0b00001001; //WGM bit compare to OCR1A to timer and set prescale
  TIMSK1 = 0b00000010; // only set up compa

  sei();
  while(true){
  }
}
```

Here we have our main function that sets up everything. It clears the interrupt data, sets up the output pins, and sets up our PWM and timer behavior.

Discussion:





So it’s hard to get a picture that really shows off the output in motion. When it starts up, the blue LED starts partially glowing with the red LEDs not lit up. The blue LED gets brighter, blinks, and gets dimmer before starting over. Every time it blinks, the red LED’s light up in the pattern of a binary representation of the number of blinks so far. It does this until all 4 red LEDs light up and then starts over from 0.

I would say this project ended up being a little simpler than I wanted to do for a final project, but I didn’t have much time to work on it. I did an experiment with the red LEDs going in reverse back to zero instead of resetting when it hit 15, but it didn’t look good visually to me. Binary going in reverse looks weird to me, so I reverted it back to the hard reset after 15 blinks, as it looks cleaner both visually and in terms of code.