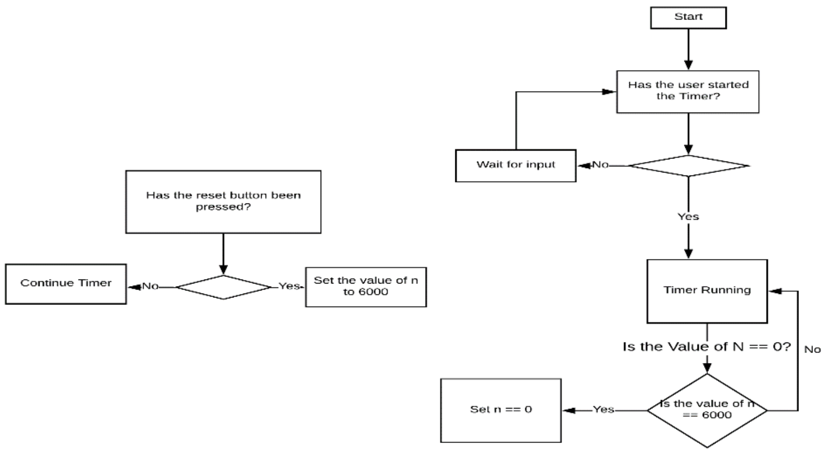
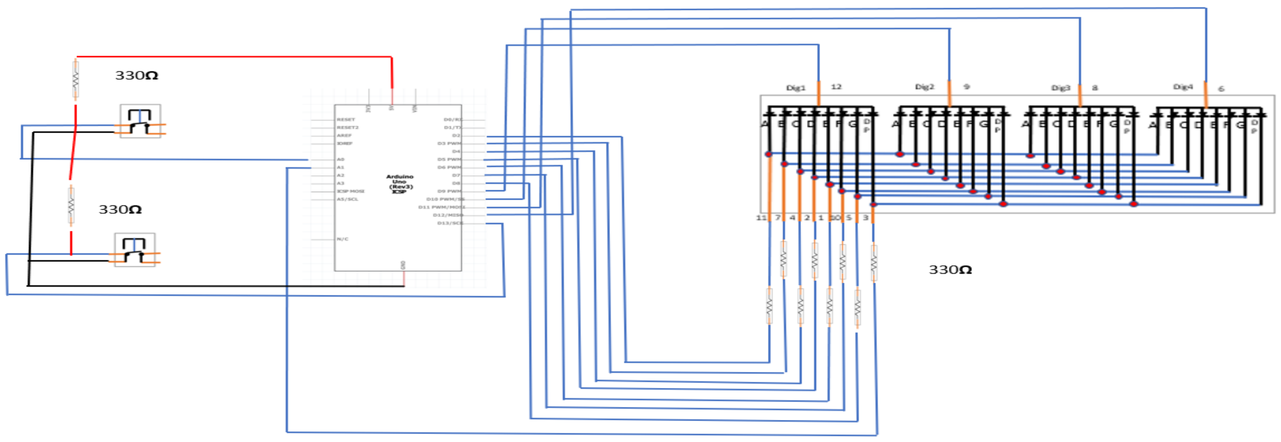
Part 1 - Stopwatch

Part 1 is a basic stopwatch that works in the following way. A user will start the stopwatch using the start/stop button (at any time this button can be pressed again to stop the stopwatch) once the stopwatch gets to 60 seconds the time will reset back down to 0 seconds. You can reset the time manually by pressing the reset button.

Internally the code of the timer is simple it uses millis to determine every second by calculating time intervals. if the current millis of the timer – the previous millis of the timer is greater than or equal to the interval it will set the previous to the current millis and add a second to the timer. The interval is 1000/100 to get the milliseconds on the stopwatch. The timer will keep doing this until n (the total value of seconds of the timer) equals 6000 (60 seconds) and it will reset n back to 0. It displays numbers by having all values of numbers 0 – 9 pre-programmed it will then decide based on a case statement what it will need to use and where it will need to use it.

The circuit uses 2 buttons for control of start/stop which have pull up resistors that can either stop or start the current allowing for the program to detect if the button is being pressed. Each pin of the 7 segment display is connected to the Arduino data pins allowing me to program in specific segments and digits in the program. These are also connected to 330 ohm resistors to lower the current to the LEDs inside the display.


Start
Has the user started the Timer?
Wait for input
No
Yes
Has the reset button been pressed?
Continue Timer
No
Yes
Set the value of n to 6000
Timer Running
Is the Value of N == 0?
No
Set n == 0
Yes
Is the value of n == 6000



Part 2 – Driving a large 7 Segment display

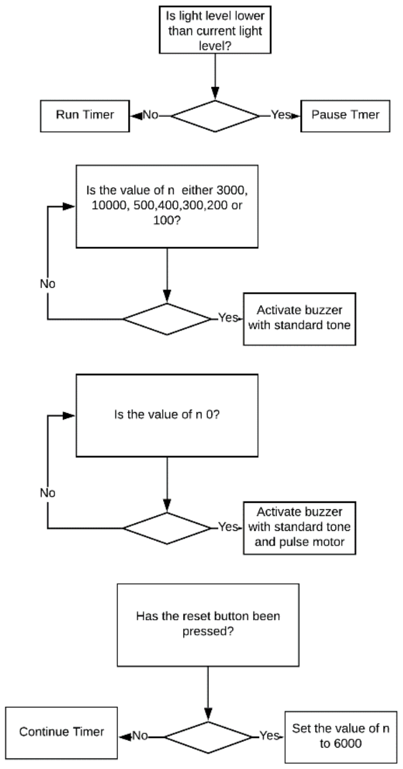
Part 2 is very similar to the first stopwatch although all the code is programmed in through a library. To use this stopwatch simply plug your Arduino into a neopixel 7 segment display with 88 neopixels (this can be configured from inside the h file in the library). The stopwatch will then count up to 60 using the same method as part one. The difference in this code was the use of a library every part of the code is in the cpp file but references the methods and variables from the h file. This library also uses the adafruit neopixel library to drive the neopixels so instead of specifying the pins that each segment of the display in a number like I did in part 1 I'm referencing each LED in the strip. However this is not very robust when changing different amounts of LEDs in a strip so when a number is declared it adds on the "segment" variable which is how many LEDs are in a single digit of the display from there it works in the same way to part 1. This library is then imported into the Arduino IDE with the statement "#include <test.h>" and the method "RunTimer" is run.

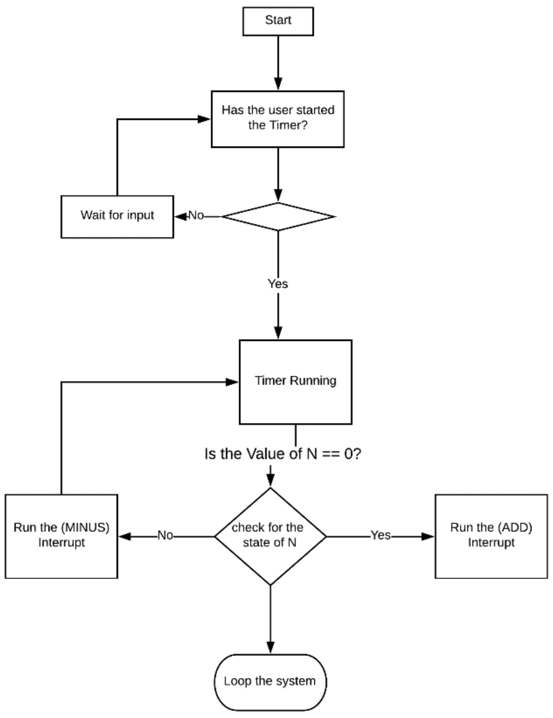
Part 3 – Creating a sports stopwatch

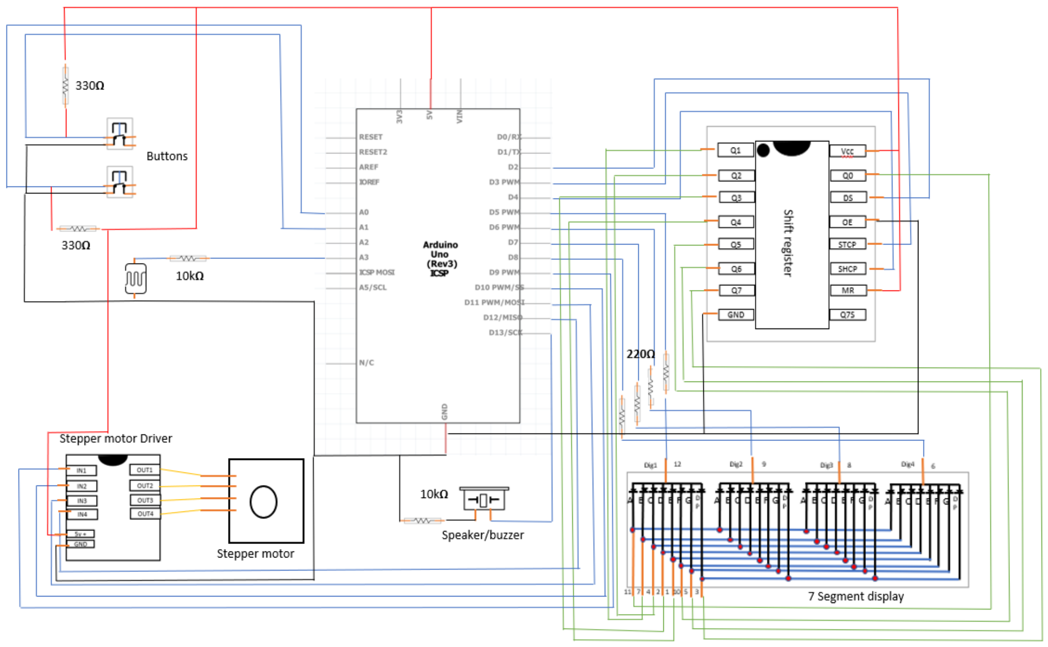
Part 3 is a sports stopwatch it counts down from 60 and beeps at 30, 10 5,4,3,2 and 1 seconds to signify the start when it hits 0 it will beep a longer tone and send a pulse to the motor to simulate the start of the race. The timer will then count up from 0 until either the line is crossed (simulated by the light sensor) where it will stop or it will reach 60 and start the countdown again. There is also a start/stop button and a reset button that works the same way as in part 1.

The circuit uses a shift register to provide additional pins for the rest of the circuit. The shift register deals with all the digital pins for the 4 digit 7 segment display providing an additional 8 pins for the price of 3 on the Arduino. The shift register works by taking in a clock and latch pin to map the locations of the other 8 pins the data pin puts the input into the Arduino. There are some other small connections such as a buzzer, and the light sensor that both use resistors to lower the current. Additionally, the main display now only has 4 resistors this is down to the multiplexing of the display as each LED is connected to the display pins if you resist the display pin it will lower the current over the whole set of LEDs.

The code for this is similar to part 1 but due to the shift register this changes how numbers can be processed. The code now uses bytes of data that have the number stored in them. This then shifts into the shift register when required. Additionally I am no longer using millis in this I am using TimerOne a library that allows me to attach an interrupt of different methods to run over the interval. TimerOne allows me to call my minus method to count down and my add method to count up. Additionally it will auto switch between these two by dethatching and attaching the interrupts.


Is light level lower than current light level?
Run Timer
No
Yes
Pause Tmer
Is the value of n either 3000, 10000, 500,400,300,200 or 100?
No
Yes
Activate buzzer with standard tone
Is the value of n 0?
No
Yes
Activate buzzer with standard tone and pulse motor
Has the reset button been pressed?
Continue Timer
No
Yes
Set the value of n to 6000


Start
Has the user started the Timer?
Wait for input
No
Yes
Timer Running
Is the Value of N == 0?
Run the (MINUS) Interrupt
No
check for the state of N
Yes
Run the (ADD) Interrupt
Loop the system


330Ω
Buttons
330Ω
10kΩ
Arduino Uno (Rev3) ICSP
Shift register
220Ω
Stepper motor Driver
Stepper motor
10kΩ
Speaker/buzzer
7 Segment display