Ninja Timer: Giant 7-Segment Display

Created by John Park

https://learn.adafruit.com/ninja-timer-giant-7-segment-display

Last updated on 2022-01-07 12:29:08 PM EST
# Table of Contents

## Overview
- Big Time 4
- Key Techniques 4
- Parts 5
- Parts 6
- Materials 7

## Build LED Digits
- Make LED Segments 7
- CAD Drawings 9

## Build the Circuit

## Test a Digit

## Assemble Panels

## Wire the Buttons

## Make Some Noise

## Enclose the System
- Mount the Circuit 104

## Timer Code

## Use It
Overview

Ninja Timer ready for action at the [Train Yard 317](https://adafruit.it/v0f) gym’s [Ultimate Nina Athlete Association](https://adafruit.it/v0A) competition.
Big Time

Ever wanted to build an enormous timer using 7-segment displays? Here's a way to do just that using NeoPixel strips for the segments and acrylic diffusers to blend the individual LEDs into seamless light sources.

This guide shows how to build 12" tall digits, each made from a meter of NeoPixels cut into segments and then joined at angles. Each digit is controlled by a single Arduino data pin, so you can make and use as many digits as you like, dependant on power supply and available Arduino pins.

Plus, you'll build rugged buttons for starting and stopping the timer, and control an ear-splitting car horn from your Arduino!

While this project was made as a timer for an obstacle course race, you can generalize the techniques in this guide to build any kind of segmented display you like!

Key Techniques

This guide will cover some techniques that are helpful when building a larger project. Fundamentally, this project is the same as simply using a button to tell an Arduino to light up a NeoPixel and beep a buzzer. But sometimes the details of scaling up your project can be intimidating.

Key techniques include:

• Diffusing groups of individual LEDs to make unified light sources
• Controlling long runs of NeoPixels without introducing signal noise
• Addressing a physical strip of NeoPixels as multiple logical segments
• Providing separate, integrated power to multiple sub-systems
• Creating polarized, modular interconnects between parts
• Running robust input wiring over dozens or hundreds of feet
In addition to the parts listed on the sidebar of this guide, you'll need the following:

- Project box such as this (https://adafru.it/uQd) for multiple button controls
- Sprinkler timer cabinet (https://adafru.it/uQe) to house the electronics
- 12V car horn (https://adafru.it/uQf)
- Cat 5e or Cat 6 (Ethernet) cable long enough for your application, x3 (for the Train Yard 312 gym we used two 50' and one 150' lengths)
- For the course stop button stand (you know, the one you dive for at the end of your run!) you'll need:
  - 4' of 3" ABS pipe
  - 3" ABS pipe coupler
  - 3" ABS closet flange
Parts
In addition to the parts listed on the sidebar of this guide, you'll need the following:

- Project box such as this (https://adafruit.it/uQd) for multiple button controls
- Sprinkler timer cabinet (https://adafruit.it/uQe) to house the electronics
- 12V car horn (https://adafruit.it/uQf)
- Cat 5e or Cat 6 (Ethernet) cable long enough for your application, x3 (for the Train Yard 312 gym we used two 50' and one 150' lengths)
- For the course stop button stand (you know, the one you dive for at the end of your run!) you'll need:
  - 4' of 3" ABS pipe
  - 3" ABS pipe coupler
  - 3" ABS closet flange (can be mounted with bolts into the flooring for the base)
Materials

Each digit panel is made up of multiple layers of cut acrylic. You'll use opaque acrylic for the body and face layers of the panel, and translucent white acrylic for the segment diffusers.

The included CAD drawings are designed to be cut from 6mm and 3mm (roughly 1/4" and 1/8") cast acrylic, but you can adapt the files as you like for other dimensions and materials.

Each panel is cut from six 16" x 12" pieces of acrylic:

- 1ea. 6mm translucent white
- 2ea. 6mm black
- 3ea. 3mm black

Acrylic cement (https://adafruit.it/uQA) and a needle applicator (https://adafruit.it/uQB) will be used to secure the segments.

You'll also need six sets of #6-32 x 1-1/2" screws (https://adafruit.it/uZA), nuts (https://adafruit.it/uZA), 1/4" nylon spacers (https://adafruit.it/uZA), and washers (https://adafruit.it/shd) to fasten each panel plus two 1-1/2" wood screws of those same diameters to mount the panels to a wall or piece of lumber.

Alternatively, you can print the plan drawings and use them as a template for cutting materials such as wood on a band saw or even with a hobby knife if you use cardboard. You diffusion material could be made from filter paper.

Build LED Digits

Make LED Segments

The first step is to trim your NeoPixel strip into small segments. Measure out and cut seven sections with 8 LEDs per segment. You can use scissors to cut at the line running through a three copper pad section.
You should end up with four extra pixels per meter strip -- save those, you'll end up using three of them as the dots for the colon and decimal point.
In order to prevent mis-wiring, use a black marker on the back of each strip to indicate the ground line and direction from DIN to DOUT of the NeoPixels. Then, you'll make a jig to arrange them into the segmented digits.

Mark your NeoPixel segments so that you don't accidentally wire any in backwards. It's very easy to do!

CAD Drawings

Click the button to download the drawings needed to cut the acrylic panels on a laser cutter. (Note, you can use these SVG files as a template for cutting by hand or with other CNC methods.)

NinjaTimerDrawings_2.svg.zip
https://adafruit.it/CTp
To keep the segments aligned during connection and soldering, as well as to support the flexible digits in the final panel, you'll cut out a jig panel from 3mm acrylic.

Flip the panel over and arrange the strips with LEDs facing down in the jig.
You can now solder small strips of wire to make the connections from ground to ground, 5V to 5V, and DOUT to DIN between each strip.

Use the above photo as a guide -- the F segment will be the one connected via wire to the Arduino and power supply and you'll connect them in the order F-A-B-C-D-E-G. (Note: if you play this sequence on a pan flute you may summon a colored LED fairie.)
Soldering short wires works just fine, but to step things up a notch, I decided to mill small PCBs called NeoJoints (https://adafruit.it/uQC) that were designed just for this purpose by Tod Kurt. I used 82 degree and 98 degree joints to achieve the properly slanted segments. There is also one straight section where B and C join.
It's helpful to tin all of the pads with solder, then tape down all of the parts, and then heat up the solder with the tip of your iron to join them.
To get data from the Arduino and level shifter to the NeoPixels, you'll solder on a 3-conductor JST SM connector to the open end of segment F. You'll connect the center wire will to the DIN pin, with one outside wire going to GND and the other to 5V.

Choose one cable gender and one conductor to be GND consistently for all digits and cables -- I decided to place the female connectors on the circuitboard and the male connectors on the NeoPixels.

Test continuity with a multimeter after each joint is soldered to catch any possible shorts.

It's important to be consistent with polarity here, so I like to mark one path on all of the connectors with a silver paint pen as the GND line.
Before soldering, slide a short length of silicone sleeve you removed earlier over the connector wires. You'll use this to insulate the connection.
Use your soldering iron to tin each pad and wire, then solder the connections.

Slide the silicone insulator into place.
Optionally, you can also add a 2-conductor JST SM cable to the out end of the NeoPixel segment to provide power from both ends. This is helpful if you experience dimness on the last pixels of your strip, although I found this wasn’t the case for my digits in the end.

To keep things running smoothly and to avoid noise, solder a 1000uF capacitor to the strip. The first NeoJoint is an ideal location -- solder the capacitor negative to GND and positive to 5V line.
This digit is ready for a circuit that can control it!

For the three panels that have a dot for the colon or decimal point, cut a single NeoPixel from a strip and wire it from the end of the G segment over the 5V, DOUT, and GND pads to the 5V, DIN, and GND pads.
Build the Circuit

In order to run the 5V NeoPixels from the 3V Arduino M0 Pro, we'll use level shifters. You can find out more about this topic in this guide (https://adafruit.it/uQD).

For this application, we'll use two of the excellent 74AHCT125 quad channel level shifters (http://adafruit.it/1787).
Each level shifter has a reference voltage pin soldered to the 5V power rail, and a common ground pin. Then, there are four sets of three pins, each constituting a channel for taking in the 3V data signal from Arduino, and outputing it at 5V to the NeoPixel digit, as well as an output enable (OE) pin tied to ground.

You'll bring all of your Arduino pins onto the circuit with a 5-conductor JST SM cable, and go out to the digits, via 1K resistors to five individual 3-conductor JST SM cables.
A large capacitor (anywhere from 1,000 to 4,700 uF) will sit across the ground and 5V rails to smooth out any voltage spikes.

Run a 9" length of black wire from the ground rail on the proto board -- this will go to ground on the Arduino later.
For any unused channels on the quad level shifter you should tie the output enable pins to voltage (HIGH) with a roughly 1K resistor. Be sure to run voltage from top to bottom rails on the PermaProto board for this to work.

Solder each of the five 3-conductor JST SM cables to the output side of the level shifters paying attention to use the same ground wire, data wire, and power wire configuration as you have on your NeoPixel digits.
Solder a short length of red 26AWG stranded wire to the power rail and a black one to the ground rail, and then screw these into the female DC power jack. This is where you'll plug in the 5V 10A power supply to give juice to the NeoPixels.
Assemble the Proto ScrewShield as per these instructions (https://adafruit.it/uQE). Then, wire the end of the 5-conductor JST SM cable to the corresponding pins D4-D8 on the ScrewShield.
Double check that this is aligned with the proper input pins on your level shifter circuit at the other end of the JST connector cable.

Use a single wire to run the PermaProto board's ground rail to a GND on the ScrewShield.

Connect the 5-conductor JST SM connectors to join the Arduino/ScrewShield combo to the level shifter board.
Next, we'll upload a sketch to the Arduino to test out the display!

---

**Test a Digit**

Plug the digit into the JST SM 3-conductor plug connected to pin 4 on your ScrewShield.

Make sure your 5V 10A power supply is plugged into AC power, and then plug it into the barrel jack on your PermaProto board.

Now, plug the Arduino M0 Pro's Programming port into your computer via a USB cable.

> Each digit is will draw roughly 1.5A of current at 5V when using the colors in this sketch. Remember, a pure R, G, or B color requires 1/3 the current compared to running all three colors at the same time at full power.
You'll need to have the NeoPixel library installed for your Arduino IDE -- you can do this from the Arduino Library Manager by going to Arduino Sketch > Include Library > Manage Libraries... and searching for NeoPixel. In case you don't see it, be sure to add the board manager URL for Adafruit libraries as shown here (https://adafru.it/nBF).

Check out the excellent NeoPixel Überguide for more info on setup in case you have not used NeoPixels before.

Open the "simple" sketch, which is located at File > examples > Adafruit NeoPixel > simple

Change the PIN to pin 4 and the number of NeoPixels (NUMPIXELS) to 56 as seen here:

```cpp
// Which pin on the Arduino is connected to the NeoPixels?
// On a Trinket or Gemma we suggest changing this to 1
#define PIN 4

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS 56
```

Now, upload the code to the board.

The display should light up!
Now, you're ready to use the digit as a proper seven segment display. Normally, a seven segment display is made from seven individual LED elements that can be individually addressed in code. We'll use code to treat our single, physical LED strip like seven logical segments. In the code, I created the segLight() function to call "segment" -- which are really just pre-defined set of eight LEDs, 0-7, 16-23, 24-31 and so on. These segLight() calls are, in turn, called by the digitWrite() function, which is a set of predefined characters.

For example, to display a "1" on the first digit panel, in the color green, we'll use the command:

```c
digitWrite(0, 1, 2);
```

Where the first argument is the digit panel 0, the second argument is the character value "1", and the third value is the color green.
The `digitWrite()` command is really just a set of pre-defined `segLight()` commands, in this case:

```c
segLight(digit,1,0);
segLight(digit,2,0);
segLight(digit,3,col);
segLight(digit,4,col);
segLight(digit,5,0);
segLight(digit,6,0);
segLight(digit,7,0);
segLight(digit,8,col);
```

Each of those `segLight()` calls has an argument for digit panel number (in this case 0), logical segment set (0-7, 8-15, and so on), and color value where 0 is off and "col" is the 4 for green as specified by the `digitWrite()` call.

Copy the test code below, create a new sketch in Arduino, paste the code, and save. This program will cycle the numbers 0-9, each in a different color.

```c
#include &lt;Adafruit_NeoPixel.h&gt;

Adafruit_NeoPixel strip[] = { //here is the variable for the multiple strips
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN0, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN1, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN2, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN3, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN4, NEO_GRB + NEO_KHZ800)
};
const int bright = 200; //adjust brightness for all pixels 0-255 range,
// 32 being pretty dim, 255 being full brightness

void setup() {
  //flash an eight
  delay(500); //pause a moment to let capacitors on board settle
  for(int s=0;s&lt;5;s++){
    strip[s].begin(); // Initialize pins for output
    strip[s].setBrightness(bright);
    strip[s].show();
    delay(200);
  }
  //Neopixel array setup
  //NeoPixel array setup
}
```
for(int t=0;t<5;t++){
  digitWrite(t,8,0); //clear it
  strip[t].show();
  digitWrite(t,8,2); //display it
  strip[t].show();
}
delay(1500);
//END void setup()

/////////////////////////////////////////////////////////////////////////

void loop() {
  //this uses digitWrite() command to run through all numbers with one second
  //timing on each of the five digit panels in the display simultaneously
  for(int i=0;i<10;i++){
    for(int j=0;j<5;j++){
      digitWrite(j, i, (i+1));
      strip[j].show();
    }
    delay(1000); //one second intervals
    for(int j=0;j<5;j++){
      digitWrite(j, i, 0); //clears the digit between refreshes
      strip[j].show();
    }
  }
}
//END void loop()

/////////////////////////////////////////////////////////////////////////

void digitWrite(int digit, int val, int col){
  //use this to light up a digit
  //digit is which digit panel one (right to left, 0 indexed)
  //val is the character value to set on the digit
  //col is the predefined color to use, R,G,B or W
  //example:
  //    digitWrite(0, 4, 2);
  //would set the first digit
  //on the right to a "4" in green.
  /*
   * Letters are the standard segment naming, as seen from the front,
   * numbers are based upon the wiring sequence
   
   A 2
   
   F 1 |       B 3
   |      
   G 7 |
   
   E 6 |       C 4
   |      
   D 5 |
   
   dp 8
  */
  //these are the digit panel character value definitions,
  //if color argument is a 0, the segment is off
  if (val==0){
    //segments F,A,B,C,D,E,G, dp
    segLight(digit,1,col);
    segLight(digit,2,col);
  }
}
segLight(digit,3,col);
segLight(digit,4,col);
segLight(digit,5,col);
segLight(digit,6,col);
segLight(digit,7,0);
segLight(digit,8,col);
}
if (val==1){
    segLight(digit,1,0);
    segLight(digit,2,0);
    segLight(digit,3,col);
    segLight(digit,4,col);
    segLight(digit,5,0);
    segLight(digit,6,0);
    segLight(digit,7,0);
    segLight(digit,8,col);
}
if (val==2){
    segLight(digit,1,0);
    segLight(digit,2,col);
    segLight(digit,3,col);
    segLight(digit,4,0);
    segLight(digit,5,col);
    segLight(digit,6,col);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==3){
    segLight(digit,1,0);
    segLight(digit,2,col);
    segLight(digit,3,col);
    segLight(digit,4,col);
    segLight(digit,5,col);
    segLight(digit,6,0);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==4){
    segLight(digit,1,col);
    segLight(digit,2,0);
    segLight(digit,3,col);
    segLight(digit,4,col);
    segLight(digit,5,0);
    segLight(digit,6,0);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==5){
    segLight(digit,1,col);
    segLight(digit,2,col);
    segLight(digit,3,0);
    segLight(digit,4,col);
    segLight(digit,5,col);
    segLight(digit,6,0);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==6){
    segLight(digit,1,col);
    segLight(digit,2,col);
    segLight(digit,3,0);
    segLight(digit,4,col);
    segLight(digit,5,col);
    segLight(digit,6,col);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==7){
    segLight(digit,1,0);
}
void segLight(char digit, int seg, int col){
  //use this to light up a segment
  //digit picks which neopixel strip
  //seg calls a segment
  //col is color
  int color[3];

  //color sets
  if (col==0){ //off
    color[0]=0;
    color[1]=0;
    color[2]=0;
  }
  if (col==1){ //red
    color[0]=255;
    color[1]=0;
    color[2]=0;
  }
  if (col==2){ //green
    color[0]=0;
    color[1]=255;
    color[2]=0;
  }
  if (col==3){ //blue
    color[0]=0;
    color[1]=0;
    color[2]=255;
  }
  if (col==4){ //white -- careful with this one, 3x power consumption
    //if 255 is used
    color[0]=100;
    color[1]=100;
    color[2]=100;
  }
}

//END void digitWrite()

////////////////////////////////////////////////////////////////////////////////////
if (col==5) {  //yellow
  color[0]=220;
  color[1]=150;
  color[2]=0;
}
if (col==6) {  //
  color[0]=0;
  color[1]=150;
  color[2]=150;
}
if (col==7) {  //
  color[0]=150;
  color[1]=0;
  color[2]=150;
}
if (col==8) {  //
  color[0]=220;
  color[1]=50;
  color[2]=50;
}
if (col==9) {  //
  color[0]=200;
  color[1]=100;
  color[2]=0;
}
if (col==10) {  //
  color[0]=0;
  color[1]=50;
  color[2]=200;
}

//sets are 0-7, 8-15, 16-23, 24-31, 32-39, 40-47, 48-55, 56
//seg F
if (seg==1) {
  //light first 8
  for (int i=0; i<8; i++) {
    strip[digit].setPixelColor(i, color[0], color[1], color[2]);
  }
}
//seg A
if (seg==2) {
  //light second 8
  for (int i=8; i<16; i++) {
    strip[digit].setPixelColor(i, color[0], color[1], color[2]);
  }
}
//seg B
if (seg==3) {
  for (int i=16; i<24; i++) {
    strip[digit].setPixelColor(i, color[0], color[1], color[2]);
  }
}
//seg C
if (seg==4) {
  for (int i=24; i<33; i++) {
    strip[digit].setPixelColor(i, color[0], color[1], color[2]);
  }
}
//seg D
if (seg==5) {
  for (int i=32; i<41; i++) {
    strip[digit].setPixelColor(i, color[0], color[1], color[2]);
  }
}
//seg E
if (seg==6) {
  for (int i=40; i<49; i++) {
    strip[digit].setPixelColor(i, color[0], color[1], color[2]);
  }
}
```c
if(seg==7){
    for(int i=48; i<57; i++){
        strip[digit].setPixelColor(i,color[0],color[1],color[2]);
    }
}
if(seg==8){
    for(int i=56; i<57; i++){
        strip[digit].setPixelColor(i,color[0],color[1],color[2]);
    }
}
```
Refer to the images here for the stacking order.

The bottom layer, cutA, will serve as the back and mounting plane for each panel. It extends higher than the rest and has extra holes for fastening.

cutB is the translucent layer, from it you will cut both the digit diffusion segments and dots as well as a spacer layer to provide offset between the LEDs and the back plane.

cutC is the jig.

cutD is the first of two layers that provide the spacing between the LEDs and the diffusion segments. Its segment cutouts are slightly smaller than the diffusion segments to prevent them from being pushed down toward the LEDs. cutE is the same, just cut from thicker material.
cutF is the top plane layer into which the diffusion segments are inlaid. There are a few variants of these to allow for the different colon and decimal point placements of your timer or clock.

6mm Translucent White acrylic works great for the diffuser segments, such as 7329 White from Delvie Plastics online.
Dry fit your panels together to make sure everything fits, then pull off the top plane and segments so you can use acrylic solvent to permanently bond them.

Keep track of your segments by labeling them A-G once they've been cut.
Apply the acrylic cement with a thin syringe. It will be drawn via capillary forces into the grooves.
Here's how all the layers go together, starting with the top layer (cutF) face down with six screws in it. (Your segments will already be cemented in place, unlike in the photos below, where they are still loose.)
Wire the Buttons

For the obstacle course, there is a start button at the beginning and a stop button at the end of it. These are made from large, 100mm arcade buttons.

There is also a button box for the judges table with a start, pause, stop, and reset button.

Since all of these are pretty far from the actual controller box, we'll use Cat 5 (or Cat 6) cable to wire them. Cat 5 cable (sometimes called Ethernet cable) is great for this type of application because it contains eight conductors, is rugged, inexpensive, and is readily available in many different lengths anywhere from 1' to 300'.

You could create a wireless button box if necessary, but wired is simple and reliable. LoRa or RFM69 Packet Radio would be good wireless systems to consider if you MUST go wireless.

An elegant way to use Cat 5 cable in an Arduino project is with the Patch Shield. This allows you to easily jumper any Arduino pin to any wire within each of four RJ-45 jacks. It includes four daughter boards with RJ-45 jacks broken out to eight pins.
Assemble the Patch Shield in the minimal configuration (https://adafru.it/uQF) per these instructions (https://adafru.it/uRa).

You'll then use small jumper wires to patch Arduino pins to the corresponding conductor on the shield's RJ-45 jack as seen here.

Your configuration can vary -- this setup uses jack J1 for the course start button, J2 for the course stop button, and J3 for the judges’ desk control box that has four buttons.
Here are the connections:

J1 conductor 1 to GND

J1 conductor 2 to Arduino pin 11

J2 conductor 1 to GND

J2 conductor 2 to Arduino pin 12

J3 conductor 8 to GND

J3 conductor 1 to Arduino pin A0 -- RESET

J3 conductor 2 to Arduino pin A1 -- START

J3 conductor 3 to Arduino pin A2 -- PAUSE

J3 conductor 4 to Arduino pin A3 -- STOP
Note that I've used the connected double rows to piggyback the black ground wire to the three jacks.

Assemble the satellite jacks, then connect two wires to each button using spade terminals, or simply solder the wires onto the common and normally open tabs on the button switch.
Repeat this method for the other 100mm button.

To build the judges' desk controller, measure out and mark centers for each button on a project box. This can be cardboard, wood, plastic, or even aluminum if you have the right tools to create the holes.

Thread the button shafts through each hole, and then secure them with the collars. I used three 60mm arcade buttons and one standard arcade button.

Thankfully, the bezels overlap and hide the holes, so they don't need to be perfect! I made mine with a cutoff disk and sanding drum on a rotary tool.
You'll need to cut off the two pegs under the 60mm buttons to get a flush fit, or, if you're feeling fancy, measure and drill two receiving holes for them!
You may also want to cut a hole for the panel mount Ethernet extension cable. This makes it very neat to connect and disconnect your Cat 5 cable.
Now, you will need to wire up all of the buttons and Ethernet extension inside the box. You can run the ground wire from the RJ-45 satellite board to one of the 60mm button switch's common tab and then daisy chain that to the other two 60mm button switches. I used a quarter-size Perma Proto board to connect the RJ-45 satellite and wiring.

I also used the arcade button quick connect wire pair for the 30mm button and soldered a receiving jack to the proto board.

Then, connect the other wires to the N.O. tab on each switch using these connections:
J3 conductor 8 to GND

J3 conductor 1 to Arduino pin A0 -- RESET

J3 conductor 2 to Arduino pin A1 -- START

J3 conductor 3 to Arduino pin A2 -- PAUSE

J3 conductor 4 to Arduino pin A3 -- STOP

You can solder a board header for the JST XPH cable, such as this one (https://adafruit.it/uRb), or just strip the wires and solder directly to the board.
Plug the Ethernet extension cable into the RJ-45 jack and you're ready to close it up.
To make the course stop button stand, fit the ABS pipe into the pipe flange to act as a mount. Then, unscrew the collar nut from the red 100mm button, remove it's outer ring.

Trim the small keyed tab from the button, and then seat it into the pipe coupler. The fit should be tight, but you can glue it in place if you like.

Twist the switch into place on the button, and then run the cable through the pipe.

Finally, fit the pipe collar and button onto the pipe.
For the stand, cut a 3-1/2' long length of 3" ABS pipe.
About 4” up from the bottom, cut a square to match the Ethernet extension cable with a cutoff wheel on a rotary tool, and drill holes for the screws.
Feed the wire with RJ-45 jack into the pipe from the top and press fit the button/collar on to it.
Plug the Ethernet extension into the RJ-45 jack, then fit the pipe bottom onto the pipe flange.
Now, you can plug a long length of Cat 5e or Cat 6 cable into the stop button pipe stand.
You can mount the flange to a floor of your obstacle course's warped wall, or mount it to a heavy base to keep it portable, yet stable.
Later, we’ll add code to reset, start, pause, and stop the timer from the buttons alongside the code for lighting up the digits, but for now you can test out the button by uploading the Arduino example button.ino sketch, and making a couple of changes to the code.

First, change the buttonPin to 12 as seen below. Then, instead of using an external pull down resistor, you can use the internal pull up resistor on the Arduino by setting the pinMode for the buttonPin to INPUT_PULLUP, also seen in the code example below.
const int buttonPin = 12;     // the number of the pushbutton pin
const int ledPin = 13;       // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
    // initialize the LED pin as an output:
    pinMode(ledPin, OUTPUT);
    // initialize the pushbutton pin as an input:
    pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
    // read the state of the pushbutton value:
    buttonState = digitalRead(buttonPin);

    // check if the pushbutton is pressed.
    // if it is, the buttonState is LOW:
    if (buttonState == LOW) {
        // turn LED on:
        digitalWrite(ledPin, HIGH);
    } else {
        // turn LED off:
        digitalWrite(ledPin, LOW);
    }
}

Put the Patch Shield on top of your ScrewShield on the Arduino, and then upload the code.

When you press the gigantic, satisfying, shiny red button, the LED built onto the Arduino will light up!

---

**Make Some Noise**

What’s a course timer without a loud horn to signal start and stop?! Well, there aren’t too many horns louder than a car horn -- let’s wire one up to a relay so we can trigger it when the start and stop buttons are pressed.
You can pick up a car horn at any auto parts store or online. They take 12V DC and around 3A to beep. Loudly.

To trigger the horn from our Arduino, we'll use a relay. You could wire one up on a proto board from parts (a relay, diode, and some resistors usually) but to make it simple we'll use the Relay FeatherWing (https://adafru.it/sEJ). This works well with the Arduino M0 Pro we're using, since it is a 3V board, just like the Feathers.
The way this works is any time you set pin 10 to HIGH on the Arduino, a small 3V control voltage signal will trigger the relay to close its circuit. This allows the 12V power source running to the terminal blocks to flow through to the horn. So, as simply as you'd normally blink an LED, you can trigger the much more powerful 12V circuit and blast the horn.

The connections to control the relay are:

Arduino GND to Power Relay GND

Arduino 3V to Power Relay 3V

Arduino pin 10 to Power Relay pin 10

To specify which pin is used to control the relay, flip the board over and solder the pad for pin 10.
Then, solder the Power Relay to a half-size Perma Proto board. As we did before with the level shifter board, we'll use a JST SM cable to connect the relay board to the Arduino. For this, use the 4-conductor JST SM cable. Even though we'll only use three of the conductors, this 4-pin cable cannot accidentally be connected to the NeoPixels and vice versa.

Solder into 3V & GND rails on the PermaProto board, and pin 10 on the relay.
To supply power to the horn through the relay, cut off the DC plug end from the 12V power supply, strip the wires, and solder the ground wire to the ground rail of the board.
Then, screw the positive wire into the far right terminal (when looking at the input holes) on the terminal block of the Power Relay shield.

Don't want cup up your power supply wires? No problem, just use a 2.1mm female DC jack as you did for the NeoPixel power. Just be sure not to mix them up when plugging in later!

The horn creates a lot of audible noise as well as digital signal noise! This is one reason to build the circuit on a separate board from the logic shifter and Arduino. We'll also use capacitors to smooth out some of that noise.
Connect one horn wire to ground on the proto board and the other to the center terminal of the relay terminal block. This circuit will be closed when the relay is triggered, thus beeping loudly.

To help contain that noise even more, connect a 1000uF capacitor from the 12V power line to ground near the relay.
This is the horn wiring without a capacitor across the terminals, which causes lots of electrical noise. Add a capacitor, otherwise you may see pixels behave strangely, or even cause the system to freeze up.

A capacitor is soldered to the horn-side wiring across the terminal connectors, with the negative leg to the ground wire, positive leg to the 12V wire, then wrapped in electrical tape and heat shrink tubing.
Also, some car horns have polarity marked on their terminals, others do not. If yours doesn't, you'll need to try it both ways and see which one works best. You can then mark the terminal polarity so you know in the future.

If the horn's electrical noise is not contained you may see NeoPixels change colors or light up when the horn beeps!

If you do use the same type of connector for two different power supplies, it is VERY easy to make a simple mistake and melt things. Label your plugs and jacks clearly to help avoid disaster!
Enclose the System

Any electronics project that will be put into real-world use needs an appropriate enclosure. For the Ninja Timer, a sprinkler timer cabinet with integrated GFCI outlet (https://adafruit.it/uQe) is a great choice. It's rugged, relatively inexpensive at around $30, has many predrilled mounting points, included two conuit holes to run wiring, has two interior layers to separate components where needed, and provides a convenient place to plug in power supplies.
You'll also need a power supply cord, such as this (https://adafru.it/uRc) to wire the outlet's line side to an external AC power source, following the instructions in the package.

AC power can injure and/or kill. Be very careful with it, never touch a bare wire while it is plugged in. If you have questions, hire a trained electrician to do wiring for you.
Cut and strip the wire from the AC plug on the 12V power supply. Place it in the bottom of the box, then run the cut AC line out of the box passthrough and then back into the GFCI enclosure.

Screw the wires into the "load" terminals on the GFCI. This allows you to power the supply without need for an extra plug extension.
Screw the GFCI plate back into the box.

There's no need to cut up the 5V 10A power supply, simply place it in the bottom of the box, and route it's wires up through the top layer cutout and plug it into the outlet.
This is what it'll look like with the top layer panel in place.

Plug the 5V micro USB power supply into the outlet as well.
Mount the Circuit

Now you can mount the three boards onto the top panel of the box. Use the 2.5mm nylon screws and standoff and the pre-drilled holes to connect them. Place the level shifter board close to the wire passthrough hole, as these will be pushed outside the box to plug into the digit panels.

For a perfect fit, you may drill out some holes to accommodate additional standoffs and screws.
Once everything is mounted to the panel, screw it into place in the box. Then, connect the three power supplies, as well as the 5-pin and 4-pin JST SM connectors. Be certain you plug 5V 10A into the level shifter board, 12V 5A into the relay board, and USB into the "Native USB" port on the Arduino. (The Arduino will run with power to either port, but we'll leave the "Programming" port open so we can plug the computer USB into in during programming.)

Run the 3-pin JST SM connector cables out of the box, as well as the car horn wire.
You can either run the three Cat 5e / Cat 6 cables to the buttons and button box directly into the enclosure, or use three Ethernet extension cables. Plug them into the J1, J2, and J3 jacks on the Patch Shield.

Make life easier on yourself later by labeling things now! You can trace each run of 3-pin JST SM cables and mark them 0-4, as well as the "start", "stop", and "desk" RJ-45 jacks for the buttons.
You can use a metallic paint pen to label your connectors, or use color-matched zip ties for easy identification.
Timer Code

Now that you system is assembled, it's time to upload the timer code.

First, you'll need to download the TaskScheduler library, which is used by the Arduino sketch to do multiple things at once (such as run the timer while listening for button presses). Go here and download the zip file. Uncompress and move the TaskScheduler folder in your libraries folder
https://github.com/arkhipenko/TaskScheduler (https://adafru.it/vdb)

Then, download the file below, unzip it, and place the directory in your Arduino sketches folder. Open it into Arduino IDE, plug in your Arduino to your computer, and then upload the code to the Arduino.
// Ninja Timer
// Course timer for Ultimate Ninja Athlete Association course
// by John Park
// for Adafruit Industries
// Timer logic by Tod Kurt
//
// MIT License
////----------------------------------------
/*
Hardware:
- Arduino M0 Pro
- NeoPixel 60 strips, x5 meters
- 2x quad Logic Level Shifters with 1k resistors
- Relay Feather Wing
- 12V Car Horn
- Arcade buttons x6
- Patch Shield and CAT5 connectors/cables for buttons

Functions:
- Power on, goes to 00:00.0 in minutes, seconds, tenths
- Press start button, the digits blink four times
- This includes three short and one long beep from the horn
- Timer starts counting up
- If course stop button is pressed, stop the time and hold, blast the horn
- If desk stop button is pressed, stop the time and hold, blast horn
- If desk pause is pressed, stop the time but allow resume on second press
- If desk reset is pressed, reset to 00:00.0
*/
>Status://----------------------------------------

#include <Adafruit_NeoPixel.h>
#include <TaskScheduler.h>

#define NUMPIXELS 57 // # of LEDs in a strip (some are actually 56, some 57 // due to dots), each strip is a digit
#define DATAPIN0 4 // digit 0 NeoPixel 60 strip far RIGHT
#define DATAPIN1 5 // digit 1 plus lower colon dot
#define DATAPIN2 6 // digit 2 plus upper colon dot
#define DATAPIN3 7 // digit 3 plus decimal dot
#define DATAPIN4 8 // digit 4 far LEFT

Adafruit_NeoPixel strip[] = { // here is the variable for the multiple strips
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN0, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN1, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN2, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN3, NEO_GRB + NEO_KHZ800),
  Adafruit_NeoPixel(NUMPIXELS, DATAPIN4, NEO_GRB + NEO_KHZ800)
};

const int bright = 200; // brightness for all pixels 0-255 range, 32 being dim

#define HORNPIN 10 // car horn: pin 10 on Arduino runs to Power Relay Feather // pin 10 (jumpered underneath relay board)

#define CSTARTPIN 11 // course start button pin using internal pullup resistor // connected via RJ-45 jack J1, conductor 2 (GND conductor 1)
#define CSTOPPIN 12 // course stop button pin // jack J2, conductor 2 (GND conductor 1)

#define DRESETPIN A0 // desk RESET button pin // jack J3, conductor 1 (GND conductor 8)
#define DSTARTPIN A1 // desk START button pin // jack J3, conductor 2 (GND conductor 8)
```c
#define DPAUSEPIN A2    // desk PAUSE button pin
#define DSTOPPIN A3     // desk STOP button pin

// Button states for reading the pushbuttons, since pullup: off = HIGH, on = LOW
int cStartState = 1;
int cStopState = 1;
int dStartState = 1;
int dStopState = 1;
int dPauseState = 1;
int dResetState = 1;

// Variables for debouncing
int cStartLastState = 1;
int cStopLastState = 1;
int dStartLastState = 1;
int dStopLastState = 1;
int dPauseLastState = 1;
int dResetLastState = 1;

int runningMode=0; // counter to tell overall state since reset
// 0 means it is a clean start, 1 means it's already running or ran
int pauseMode=0; // counter for using pause to stop and start timer
int stoppedMode=0;
int go = 0; // start counter once blinking/beeping is done
int dispColor = 3; // color to call out for display, feeds other voids

volatile bool isNinjaCounting = false; // true when start button pressed
volatile uint32_t ninjaMillisTime = 0; // the actual counter value

// How often the inputs are readInputs
const int inputsMillis = 10;
// How often the counter is updated
const int counterMillis = 10;
// How often the display is updated
const int displayMillis = 100;

// Task prototypes
void readInputs();
void updateCounter();
void updateDisplay();
void debugDisplay();
void startSequence();
void stopSequence();

Scheduler runner;

// Change these repeat times to appropriate ones for real application
Task t1(inputsMillis, TASK_FOREVER, &readInputs);
Task t2(counterMillis, TASK_FOREVER, &updateCounter);
Task t3(displayMillis, TASK_FOREVER, &updateDisplay);
Task t4(1000, TASK_FOREVER, &debugDisplay);
// Task t5();
// Task t6();

(itemView)

void setup() {
  delay(2000);
  Serial.begin(115200);
  Serial.println("NinjaTimer TEST");
  // Initialize serial for debugging
  Serial.println("'s' to start timer");
  
  // Additional setup code...
}
```

Serial.println("'x' to stop timer");
Serial.println("'p' to pause timer");
Serial.println("'r' to restart timer");

runner.init();
runner.addTask(t1);
runner.addTask(t2);
runner.addTask(t3);
runner.addTask(t4);
t1.enable();
// t2.enable();
t2.disable();
t3.enable();
t4.enable();

//NeoPixel array setup
for(int s=0;s<5;s++){
  strip[s].begin(); // Initialize pins for output
  strip[s].setBrightness(bright); //full brightness 255
  strip[s].show(); // Turn all LEDs off
  delay(200);
}
//flash dashes
for(int t=0;t<5;t++){
  digitalWrite(t,8,0);//blank
  strip[t].show();
  segLight(t, 7, dispColor);
  strip[t].show();
}
delay(1000);

//Relay pin setup
pinMode(HORNPIN, OUTPUT);

//Button pin setup
pinMode(CSTARTPIN, INPUT_PULLUP);
pinMode(CSTOPPIN, INPUT_PULLUP);
pinMode(DSTARTPIN, INPUT_PULLUP);
pinMode(DSTOPPIN, INPUT_PULLUP);
pinMode(DPAUSEPIN, INPUT_PULLUP);
pinMode(DRESETPIN, INPUT_PULLUP);

}  
//END void setup()

void loop()
{  
  runner.execute();
}

void readInputs()
{
  //check the state of the buttons
  cStartState=  digitalRead(CSTARTPIN);
  cStopState =  digitalRead(CSTOPPIN);
  dStartState = digitalRead(DSTARTPIN);
  dStopState =  digitalRead(DSTOPPIN);
  dPauseState = digitalRead(DPAUSEPIN);
  dResetState = digitalRead(DRESETPIN);

  //Course Start
/
if(cStartState!=cStartLastState){ //debounce - state has changed
    if (cStartState==LOW) { //the course start button is pushed
        if (runningMode==0){ //normal start
            dispColor=2; //color is green
            //sound the horn four times
            delay(100); //slight delay after pressing
            for(int h = 0; h < 3; h++){ //three horn blasts
                //flash the display and beep the horn
                for(int t=0; t<5; t++){
                    digitWrite(t,8,0); //blank
                    strip[t].show();
                }
                delay(100);
                for(int t=0; t<5; t++){
                    digitWrite(t,0,dispColor); //0s
                    strip[t].show();
                }
                digitalWrite(HORNPIN,HIGH);
                delay(250);
                for(int t=0; t<5; t++){
                    digitWrite(t,8,0); //blank
                    strip[t].show();
                }
                digitalWrite(HORNPIN,LOW);
                delay(750);
            }
            startCounter();
            digitalWrite(HORNPIN,HIGH);
            delay(500);
            digitalWrite(HORNPIN,LOW);
            runningMode = 1; //indicates it is running or has run
        }
        else if(runningMode==1){
            //start button disabled
            Serial.println("already started");
        }
    }
    cStartLastState = cStartState; //for debounce
}

//Course Stop
//
if(cStopState!=cStopLastState){
    if (runningMode==1){ //only stop if started
        if (cStopState==LOW) { //the course stop button is pushed
            dispColor=1; //red
            stopCounter();
            digitalWrite(HORNPIN,HIGH);
            delay(1500);
            digitalWrite(HORNPIN,LOW);
            stoppedMode=1;
        }
    }
    cStopLastState=cStopState;
}

//Desk control box Start
//
if(dStartState != dStartLastState){
if (dStartState==LOW) { //the desk start button is pushed
    if (runningMode==0){ //normal start
        dispColor=2;  //color is green
        //sound the horn four times
        delay(100);
        for(int h = 0; h < 3; h++){
            //flash the display and beep the horn
            for(int t=0;t<5;t++){
                digitalWrite(t,8,0);//blank
                strip[t].show();
            }
            delay(100);
        }
        digitalWrite(HORNPIN, HIGH);
        delay(250);
        for(int t=0;t<5;t++){
            digitalWrite(t,0,dispColor);//0s
            strip[t].show();
        }
        digitalWrite(HORNPIN,LOW);
        delay(750);
    }
    startCounter();
    digitalWrite(HORNPIN, HIGH);
    delay(500);
    digitalWrite(HORNPIN,LOW);
    runningMode = 1;  //indicates it is running or has run
    } else if(runningMode==1){
        //start button disabled
        Serial.println("nope start");
    }
    dStartLastState=dStartState;
}

//Desk Stop
//
if(dStopState!=dStopLastState){
    if (runningMode==1) { //only can stop if started
        if (dStopState==LOW) { //the desk stop button is pushed
            dispColor=1; //red
            stopCounter();
            digitalWrite(HORNPIN, HIGH);
            delay(1500);
            digitalWrite(HORNPIN,LOW);
            stoppedMode=1;
        }
    }
    dStopLastState=dStopState;
}

//Pause
//
if(dPauseState!=dPauseLastState) { //for debouncing
    if(runningMode==1&&stoppedMode==0){ //only makes sense to pause if running
        if (dPauseState==LOW) { //the pause button is pushed
            } else {  //the 0s
                digitalWrite(HORNPIN, HIGH);
                delay(500);
                digitalWrite(HORNPIN,LOW);
            }
    }
    dPauseLastState=dPauseState;
}
if (pauseMode==0) { //it was previously running
dispColor=5; //yellow
pauseCounter();
pauseMode=1;
Serial.println("paused");
}
else if (pauseMode==1){ //it was previously pause, start up again
  dispColor=2; //green
  startCounter();
pauseMode=0;
  Serial.println("restarted");
}
}
dPauseLastState=dPauseState; //for debouncing

//Reset
//
if(dResetState!=dResetLastState){ //for debouncing
  if (dResetState==LOW) { //the reset button is pushed
    dispColor=3;//blue
    //flash dashes
    for(int t=0;t&lt;5;t++){
      digitWrite(t,8,0);//blank
      strip[t].show();
      segLight(t, 7, 4);//white dashes
      strip[t].show();
      delay(170);
    }
    runningMode=0;
pauseMode=0;
stoppedMode=0;
resetCounter();
}
dResetLastState=dResetState; //for debouncing
}
if( Serial.available() &gt; 0 ) {
  int ch = Serial.read();
  if( ch == 's' ) {
    startCounter();
  } else if( ch == 'x' ) {
    stopCounter();
  } else if( ch == 'p' ) {
    pauseCounter();
  } else if( ch == 'r' ) {
    resetCounter();
  }
}

//END void readInputs()

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// task called periodically by TaskScheduler void updateCounter() {
  if( isNinjaCounting ) {
    // FIXME: this doesn't take into account overhead
    ninjaMillisTime += counterMillis;
  }
} //END void updateCounter()


©Adafruit Industries
// Actually update the display.
// May also do other effects like blinking
// task called periodically by TaskScheduler
void updateDisplay() {
  int tenths = (ninjaMillisTime / 100) % 10;
  int secs = (ninjaMillisTime / 1000) % 60;
  int mins = (ninjaMillisTime / 1000 / 60) % 60;

  int secsTens = secs/10; //get the tens place of seconds
  int secsOnes = secs%10; //get the ones place of seconds
  int minsTens = mins/10; //get the tens place of minutes
  int minsOnes = mins%10; //get the ones place of minutes

  //write to NeoPixel seven segments, colors:
  //0 = off
  //1 = red
  //2 = green
  //3 = blue
  //4 = white
  //5 = yellow
  //
  digitWrite(0, tenths, dispColor);
  strip[0].show();
  digitWrite(1, secsOnes, dispColor);
  strip[1].show();
  digitWrite(2, secsTens, dispColor);
  strip[2].show();
  digitWrite(3, minsOnes, dispColor);
  strip[3].show();
  digitWrite(4, minsTens, dispColor);
  strip[4].show();

  char dispbuf[10];
  sprintf(dispbuf, "%02d:%02d.%01d", mins, secs, tenths);
  Serial.print("DISPLAY: "); Serial.print(ninjaMillisTime);
  Serial.print(" mm:ss.t: "); Serial.println(dispbuf);
}

//END void updateDisplay()

// A slower-updating display that logs to the serial port,
// useful for debugging
// task called periodically by TaskScheduler
void debugDisplay() {
  Serial.print("DEBUG: time:"); Serial.print(ninjaMillisTime);
  Serial.print(" isNinjaCounting:"); Serial.println(isNinjaCounting);
}

//END void debugDisplay()

// starts the counter running
// via START button
void startCounter() {
  t2.enable(); //this needs to be enabled here to deal with beep delay
  Serial.println("START");
  isNinjaCounting = true;
}

//END void startCounter()
// does NOT reset display
void stopCounter() {
    Serial.println("STOP");
    isNinjaCounting = false;
}
//END void stopCounter()

// pauses counter
// via PAUSE button
void pauseCounter() {
    Serial.println("PAUSE");
    isNinjaCounting = false;
}
//END void pauseCounter()

// reset counter to zero, stops counter
// via RESET button (or boot)
void resetCounter() {
    Serial.println("RESET");
    ninjaMillisTime = 0;
    stopCounter();
}
//END void resetCounter()

void digitWrite(int digit, int val, int col) {
    // use this to light up a digit
    // 'digit' is which one (right to left, 0 indexed)
    // 'val' is the value to set on the digit
    // 'col' is the color to use, R,G,B or W
    // example:
    //         digitWrite(0, 4, 2);
    // this would set the digit
    // on the far right to a "4" in green.

    /*
    // Letters are the standard naming, numbers are based upon the wiring sequence

    A 2
    
    F 1 |         | B 3
    |     |        |
    G 7
    
    E 6 |         | C 4
    |     |        |
    D 5 |         | dp 8

    */
    // these are the numeric character definitions,
    // if last argument is a 0, the segment is off
    if (val==0) {
        // "0"
        segLight(digit,1,col);
        segLight(digit,2,col);
        segLight(digit,3,col);
        segLight(digit,4,col);
    }
}
segLight(digit,5,col);
segLight(digit,6,col);
segLight(digit,7,0);
segLight(digit,8,col);
}
if (val==1){ // "1"
    // segments A, B, C, D, E, F
    segLight(digit,1,0);
    segLight(digit,2,0);
    segLight(digit,3,col);
    segLight(digit,4,col);
    segLight(digit,5,0);
    segLight(digit,6,0);
    segLight(digit,7,0);
    segLight(digit,8,col);
}
if (val==2){ // "2"
    // segments A, B, C, D, E, F
    segLight(digit,1,0);
    segLight(digit,2,col);
    segLight(digit,3,col);
    segLight(digit,4,0);
    segLight(digit,5,col);
    segLight(digit,6,col);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==3){ // "3"
    // segments A, B, C, D, E, F
    segLight(digit,1,0);
    segLight(digit,2,col);
    segLight(digit,3,col);
    segLight(digit,4,col);
    segLight(digit,5,col);
    segLight(digit,6,0);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==4){ // "4"
    // segments A, B, C, D, E, F
    segLight(digit,1,col);
    segLight(digit,2,0);
    segLight(digit,3,col);
    segLight(digit,4,col);
    segLight(digit,5,0);
    segLight(digit,6,0);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==5){ // "5"
    // segments A, B, C, D, E, F
    segLight(digit,1,col);
    segLight(digit,2,col);
    segLight(digit,3,0);
    segLight(digit,4,col);
    segLight(digit,5,col);
    segLight(digit,6,0);
    segLight(digit,7,col);
    segLight(digit,8,col);
}
if (val==6){ // "6"
    // segments A, B, C, D, E, F
    segLight(digit,1,col);
    segLight(digit,2,col);
    segLight(digit,3,0);
    segLight(digit,4,col);
    segLight(digit,5,col);
    segLight(digit,6,col);
    segLight(digit,7,col);
void segLight(char digit, int seg, int col){
    // 'digit' picks which neopixel strip
    // 'seg' calls a segment
    // 'col' is color
    /*
    //always blank the LEDs first
    for(int i = 0; i<56; i++){
        strip[0].setPixelColor(i, 0, 0, 0);
    }
    */
    int color[3];
    //color sets
    if (col==0) { //off
        color[0]=0;
        color[1]=0;
        color[2]=0;
    }
    if (col==1) { //red
        color[0]=255;
        color[1]=0;
        color[2]=0;
    }
    if (col==2) { //green
        color[0]=0;
        color[1]=255;
        color[2]=0;
    }
    //END void digitWrite(int digit, int val, int col)
}
if (col==3) { // blue
  color[0]={0};
  color[1]={0};
  color[2]={255};
}
if (col==4) { // white -- careful with this one, 3x power consumption
  color[0]={255};
  color[1]={255};
  color[2]={255};
}
if (col==5) { // yellow
  color[0]={200};
  color[1]={120};
  color[2]={0};
}

// sets are 0-7,

// seg F
if(seg==1){
  // light first 8
  for(int i=0; i<8; i++){
    strip[digit].setPixelColor(i,color[0],color[1],color[2]);
  }
  //strip(digit).show();
}

// seg A
if(seg==2){
  // light second 8
  for(int i=8; i<16; i++){
    strip[digit].setPixelColor(i,color[0],color[1],color[2]);
  }
  //strip(digit).show();
}

// seg B
if(seg==3){
  for(int i=16; i<24; i++){
    strip[digit].setPixelColor(i,color[0],color[1],color[2]);
  }
  //strip(digit).show();
}

// seg C
if(seg==4){
  for(int i=24; i<33; i++){
    strip[digit].setPixelColor(i,color[0],color[1],color[2]);
  }
  //strip.show();
}

// seg D
if(seg==5){
  for(int i=32; i<41; i++){
    strip[digit].setPixelColor(i,color[0],color[1],color[2]);
  }
  //strip(digit).show();
}

// seg E
if(seg==6){
  for(int i=40; i<49; i++){
    strip[digit].setPixelColor(i,color[0],color[1],color[2]);
  }
  //strip(digit).show();
}

// seg G
if(seg==7){
  for(int i=48; i<57; i++){
Now, you can put your timer to the test! Unplug it from your computer, and then connect the box cables to your digit panels, car horn, and three button Cat 5e/Cat 6 cables.

Next, plug the AC power cord into an outlet. It will power on and display all "0"s in blue after a couple of seconds.

Press course start button or desk control start button to begin your run. The horn will sound four times, and then start counting.

At the end of your run, dive for the course stop button. The horn will sound and your time will be displayed in red.

At any time, the desk controller box can be used to pause a run by pressing the yellow button, and restarting the run by pressing it a second time. This is useful if a dog, toddler, or knife wielding actual honest to goodness ninja shows up.

Finally, the blue button on the desk controller box can be pressed to reset the timer in anticipation of the next course run!
Use It

Time to install and use your course timer! Install the panels by screwing the top holes in each digit panel back plane to a length of wood to mount on the wall.
The controller box needs to be mounted below or behind the digits in order to be plugged in to the panels, as well as to AC power.

Before you power up the system, plug in each panel's connector cable to its corresponding connector on the controller box. Then, plug in the three Ethernet cables for the buttons, and the car horn wiring to the car horn.

Now, when you plug in the main AC power line (you can use a power strip as an on/off button if you like), the system will boot up after a couple of seconds and display all zeros in blue -- this indicates that you're ready to start timing a new run or event.
Press either start button (course or desk box) and the digits flash green and the horn beeps three times as a countdown. On the fourth blast of the horn, the timer starts.

Now, you can either pause and restart from the yellow desk box button, or stop the run from the course or desk box. The horn will blast and the final time will display in red. All buttons other than the blue reset button on the desk box are disabled.

Reset the box with the blue desk reset to start your next run!

Here's the Ninja Timer in action at the Train Yard 317 gym during the UNAA competition.
You can find all sorts of uses for a huge timer, or write some new code and add a real-time clock unit and use it as an alarm clock instead!