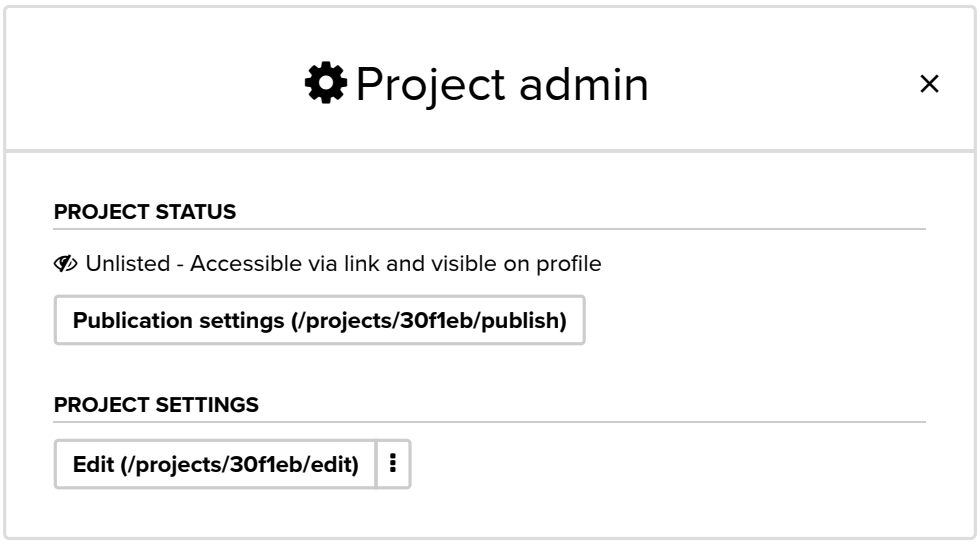


Thomas Nabelek (/thomas-nabelek)
Created January 22, 2019 © GPL3+ (http://opensource.org/licenses/GPL-3.0)

Arduino Prototype to Producible PCB: An LED Multiplexer

Learn the minumum electronic and software aspects of taking an Arduino-based breadboard prototype to producible consumer device.

Intermediate(/projects?difficulty=intermediate) Work in progress 12

(https://store.groupgets.com/products/mini-flir-lepton-smart-i-o-module)

RELATED CHANNELS AND TAGS

breadboard (/projects/tags/breadboard)
pcb (/projects/tags/pcb)
prototype (/projects/tags/prototype)
prototyping (/projects/tags/prototyping)
rapid prototyping (/projects/tags/rapid+prototyping)

Things used in this project

Hardware components

See component list in each stage below. Consider ordering the components from all components lists in this project to save on minimize cost and maximize time. × 1

Story

Introduction

My goal of this project was to teach myself (with the assistance of my helpful folks on the interwebs) how to take an Arduino-based breadboard prototype all the way to mass-producible PCB (printed circuit board). I thought that I would brake it down into five stages:

1) Breadboard circuit linked to an Arduino.

2) Phase 1, but with the Arduino replaced with only necessary components.

3) Phase 2, but soldered together using stripboard. (https://en.wikipedia.org/wiki/Stripboard)

4) A PCB fabricated by a third-party that I would assemble myself.

5) A PCB fabricated and assembled by a third party.

The idea of this project was not to create the adjustable LED multiplexer, but rather to learn the absolute bare-bones process of developing a producible consumer electronic device (minus just about everything but the actual technical aspects). I choose the LED multiplexer as a circuit that was simple enough to not slow progress of achieving my main goal but complex enough to be a little more interesting and challenging than a single blinking LED.

I will note upfront that this guide is lacking in a lot of details and is intended more as an overview for anyone seeking to learn the basics of this process. If you have specific questions, feel free to ask them in the comments and I will try to answer them. This guide assumes previous experience with Arduinos and the Arduino IDE, and of the C programming language if you want to understand what the code is doing.

Stage 1

I broke Stage 1 into there sub-stages: stage 1A, 1B, and 1C. Stages 1A and 1B were dedicated to getting the LED multiplexer down pat. Stage 1C was the point at which I integrated a pushbutton, two potentiometers, a toggle switch, and a battery to make the circuit a bit more interesting and independent of power provided by my computer.

Stages 1A and 1B

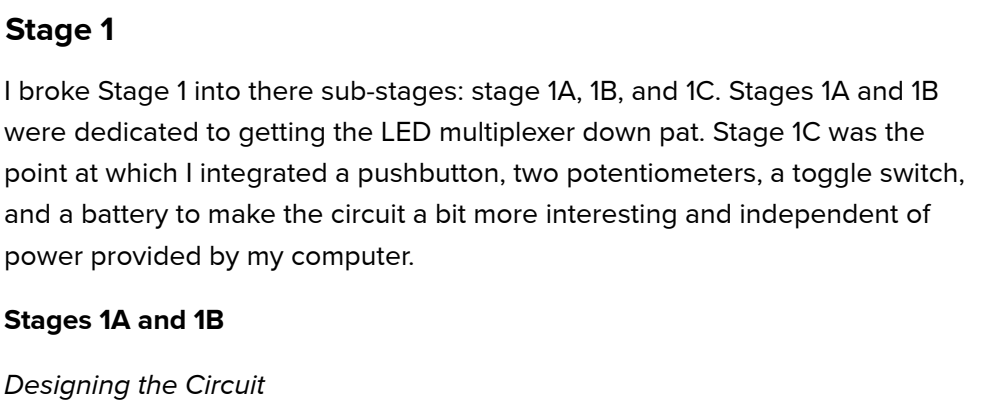
Designing the Circuit

If you don't care much about the circuit theory, feel free to skip down to the *Assembling and Running the Circuit* subsection.

I started with this Instructable by perez1028 (https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-I-made/)and went from there. I made some heavy adjustments and initially had some issues, as evidenced by my post here (https://electronics.stackexchange.com/questions/411244/questions-about-bjts-in-my-circuit?noredirect=1#comment1014421_411244)(thanks to the folks there).

INSERT discussion about multiplexing.
and this scales so that, assuming a square matrix, the required number of control signals is equal to twice the square root of the total number or LEDs (e.g., 6 = 2*sqrt(9), 10 = 2*sqrt(25))

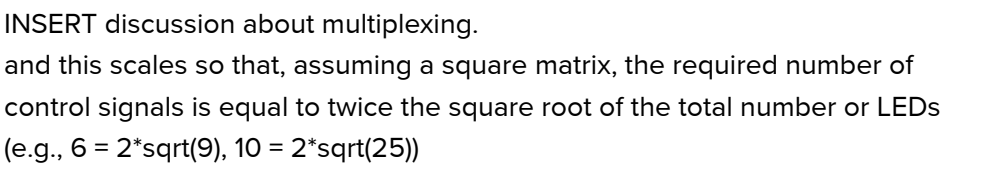
After some trial, error, and iteration, I settled on this:

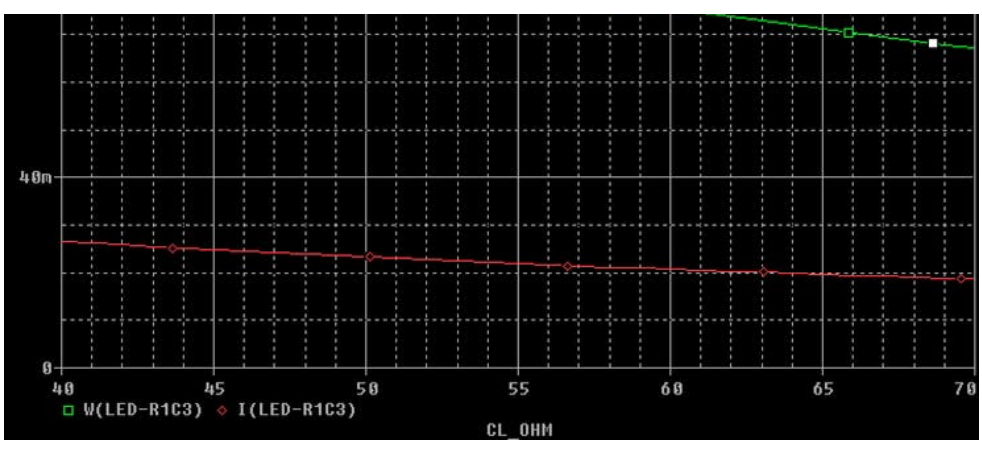


Circuit schematic for LED multiplexer

I used DC voltage sources to simulate the voltage that would be supplied by the Arduino I/O pins.

I wanted the LEDs to be as bright as possible without burning out. I was using Cree C512A-WNS/WNN LEDs, so I took a look at their datasheet (https://www.cree.com/led-components/media/documents/C512A-WNS-WNN-942.pdf)and found the maximum forward current that these LEDs can withstand to be 25 mA and the maximum power dissipation to be 100 mW. Also, on an Arduino, the maximum current output per I/O pin is 40 mA (and 200 mA from VCC). It is important to make these considerations for all components in the circuit. With these things in mind, I ran various simulations to try to find the optimal resistor values for my design. For instance, I did a sweep of what I have labeled as the CL_OHM resistance value to see the current and power dissipation through/at LED-R1C3:

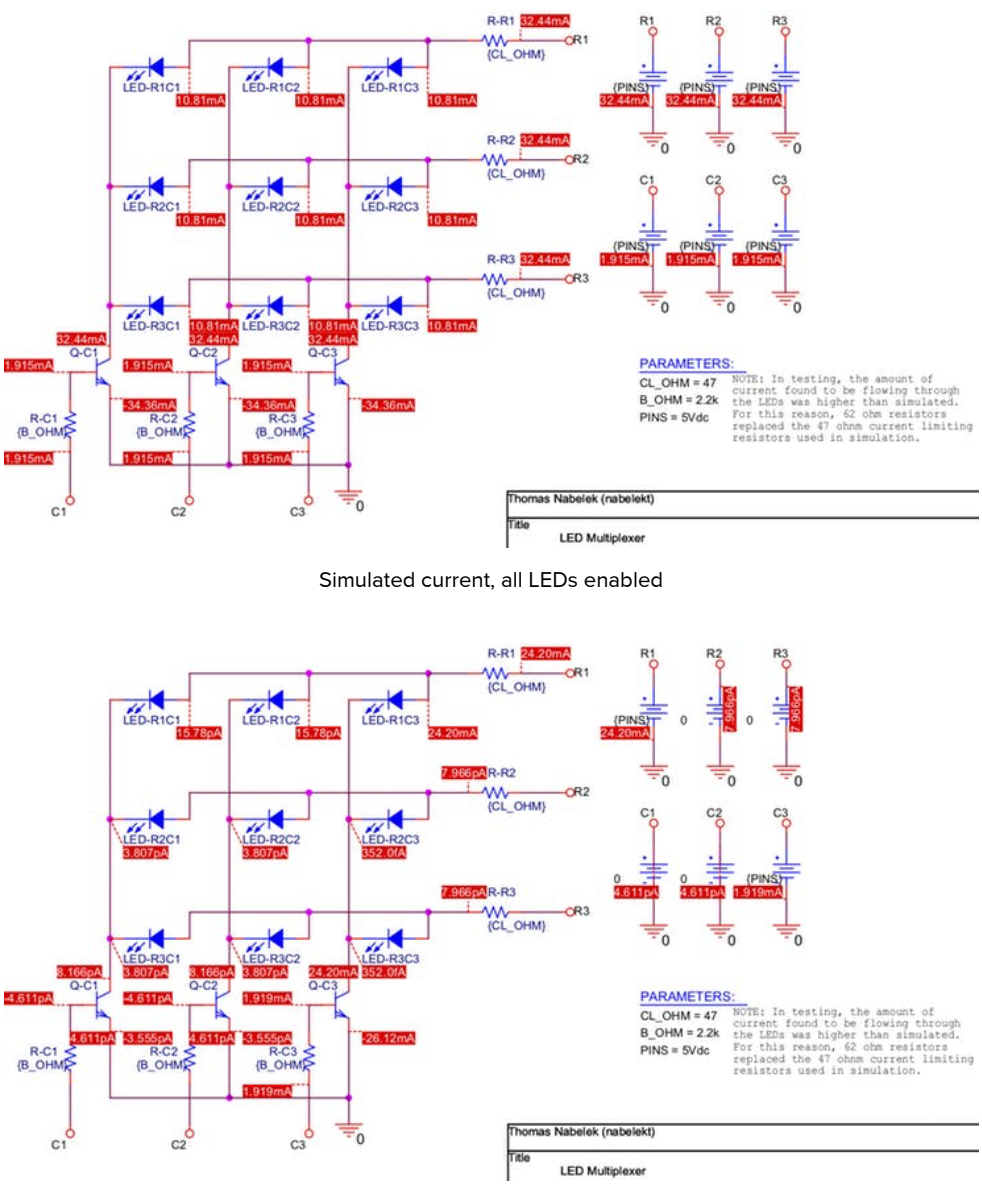




Current (red, mA) and power dissipation (green, mW) through/at LED-R1C3; CL_OHM axis values given in ohms

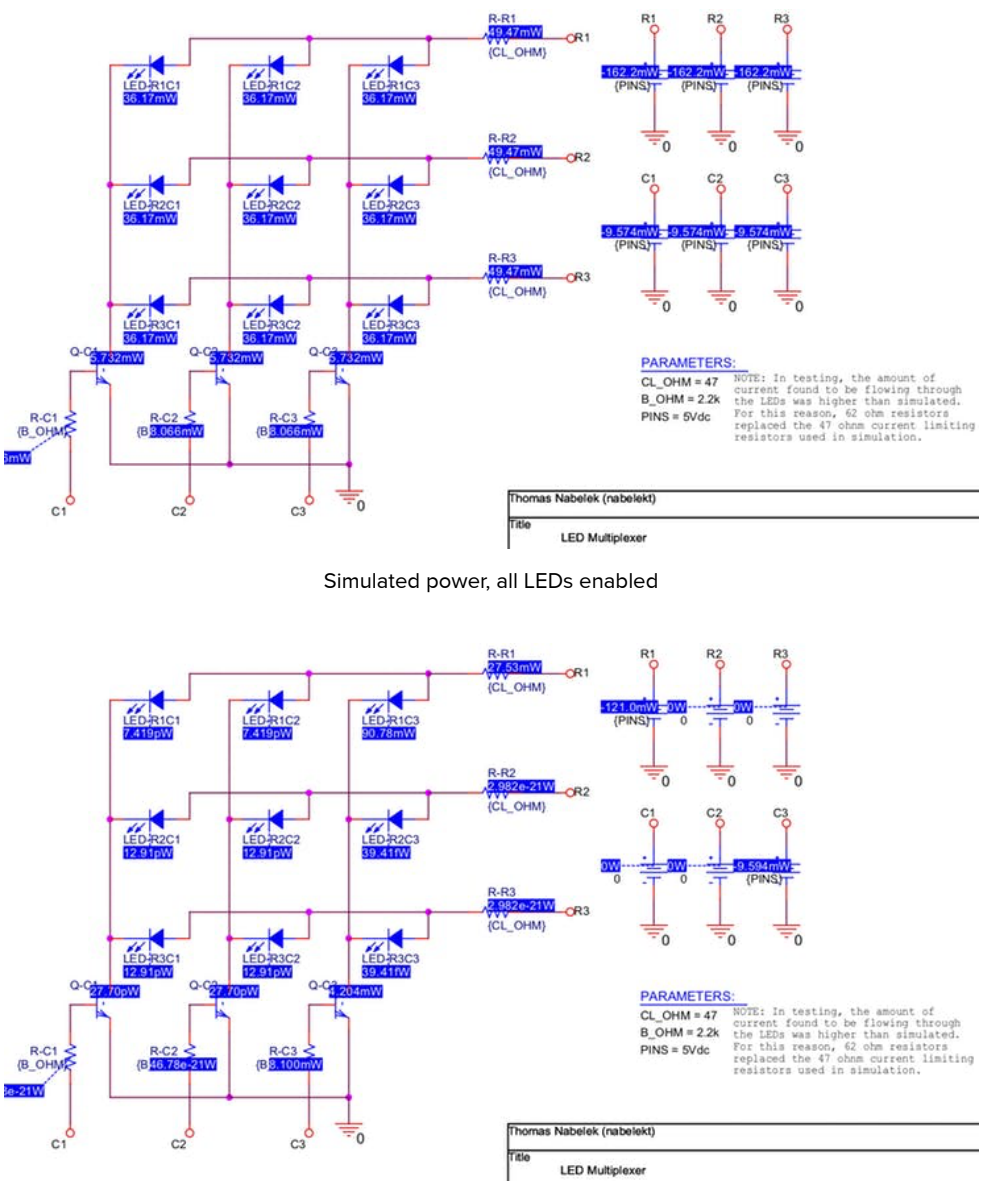
This was done with only pins R1 and C3 enabled at 5 V and all of the other pins set to 0Vdc – a "worst case" scenario. Given that 47 Ω is a common resistor value, it seemed like a great choice. At 47 Ω, this simulation showed about 24.5 mA of current and 91 mW of power dissipation.

I created the schematic and did the simulation using OrCAD PSpice Designer Lite. (https://www.orcad.com/resources/download-orcad-lite) The software lets you simulate the circuit and get the voltage at any node, current on any branch, and power consumption for any component. Here are a bunch of simulated values:



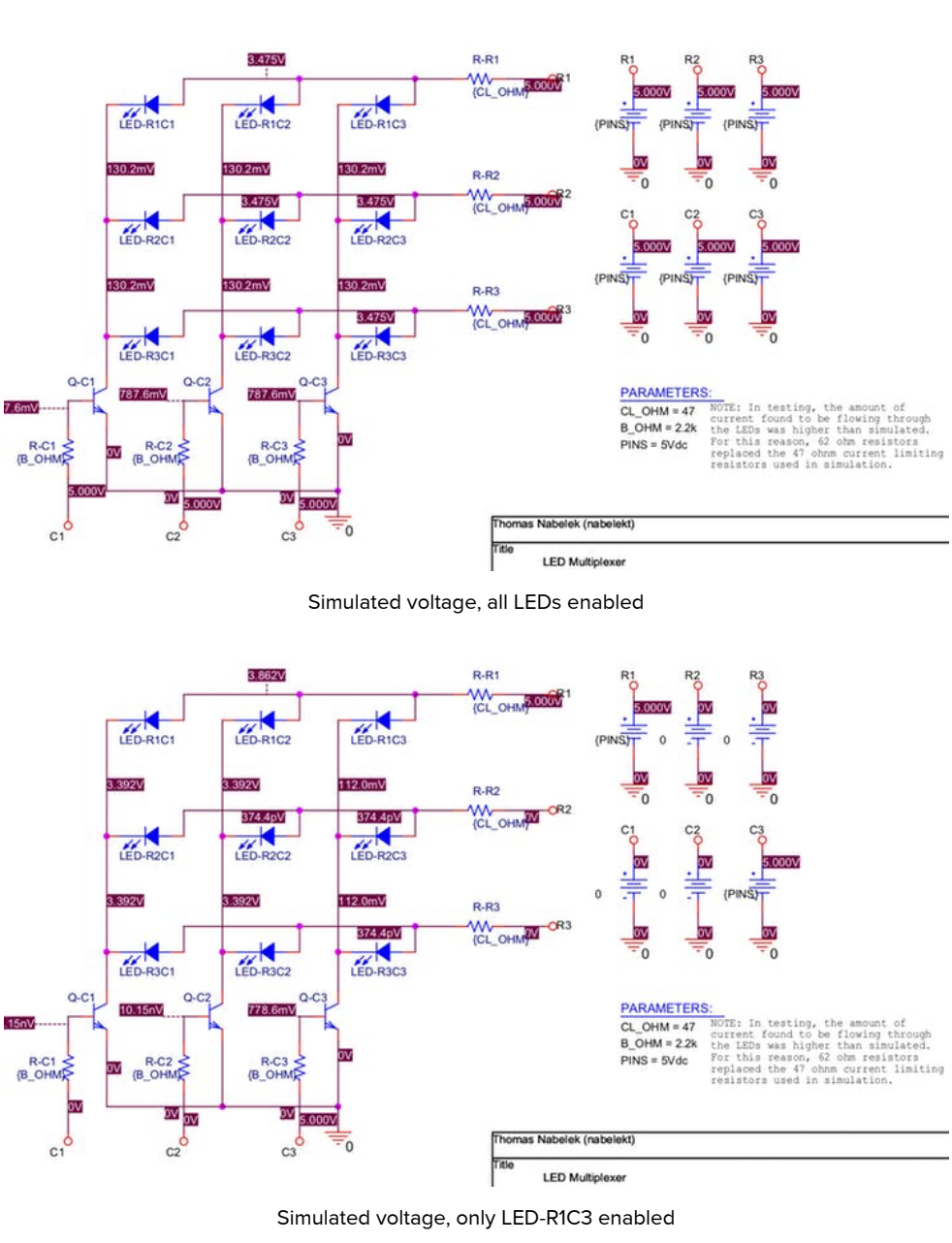
Simulated current, all LEDs enabled

Simulated current, only LED-R1C3 enabled



Simulated power, all LEDs enabled

Simulated power, only LED-R1C3 enabled



Simulated voltage, all LEDs enabled

Simulated voltage, only LED-R1C3 enabled

However, notice the note on the right side of the schematic: "In testing, the amount of current found to be flowing throughthe LEDs was higher than simulated. For this reason, 62 ohm resistors replaced the 47 ohnm current limiting resistors used in simulation."

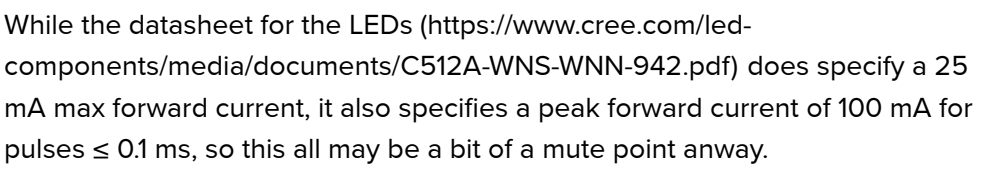
All of this was done enabling both the anode and cathode of only one LED – a sort of "worst case," as mentioned. In reality, with the way that the multiplexer works and the code below is written, a pulse-width modulation effect occurs. While the datasheet for the LEDs (https://www.cree.com/led-components/media/documents/C512A-WNS-WNN-942.pdf) does specify a 25 mA max forward current, it also specifies a peak forward current of 100 mA for pulses ≤ 0.1 ms, so this all may be a bit of a mute point anway.

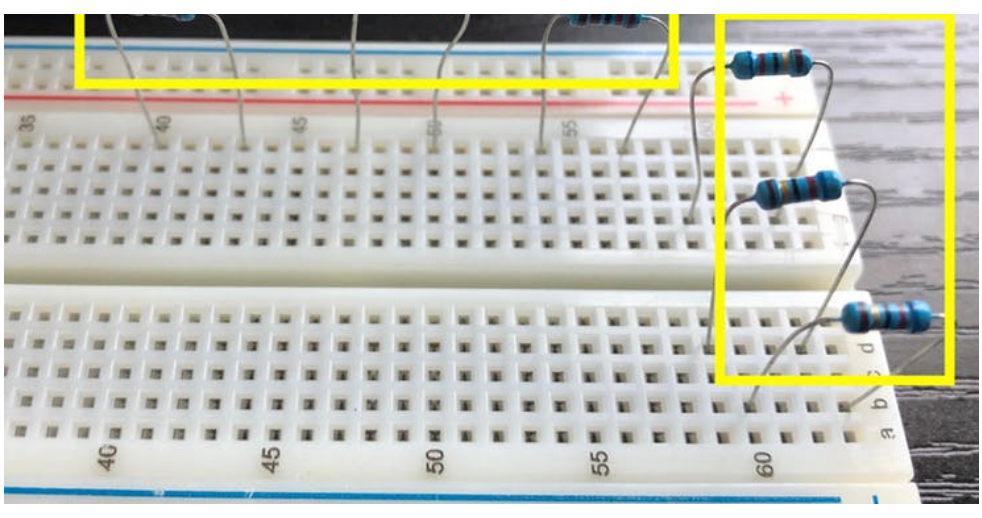
Assembling and Running the Circuit

Component list:

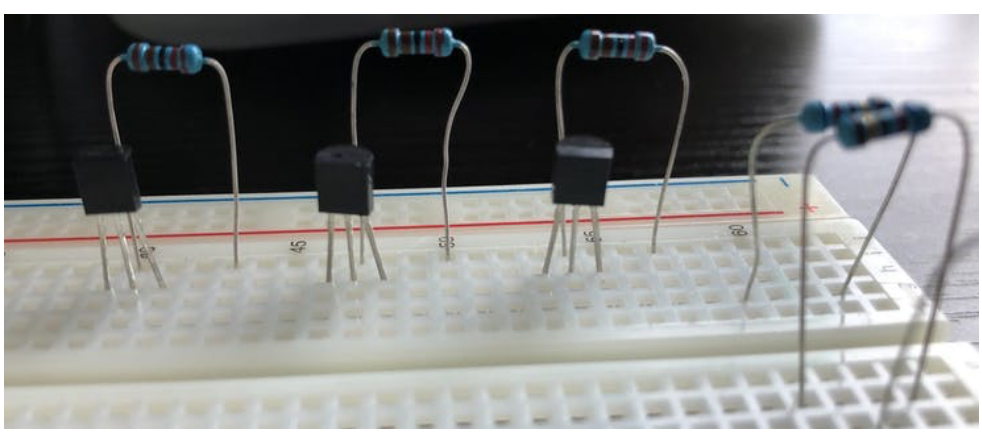
- Computer with Arduino IDE and USB cable to connect to Arduino
- Arduino Uno R3 or cheaper knock-off (these (https://www.amazon.com/gp/product/B01EWOE0UU) have worked very well for me)
- Solderless breadboard (https://www.digikey.com/product-detail/en/adafruit-industries-llc/239/1528-2143-ND/7244929)
- 3x 62 Ω resistors (I used these (http://amazon.com/gp/product/B07J48VNR7) but don't necessarily recommend them – pretty sure the tolerance is much greater than 1%)
- 3x 2.2 kΩ resistors
- 3x 2N3904 NPN BJTs (transistors) (I used these (https://www.amazon.com/gp/product/B06XRBLKDR))
- 9x Cree C512A-WNN-CZ0B0152 LEDs (https://www.digikey.com/product-detail/en/cree-inc/C512A-WNN-CZ0B0152/C512A-WNN-CZ0B0152CT-ND/6138557)
- wire (jumper wires like these (https://www.digikey.com/product-detail/en/PRT-12795/1568-1512-ND), and these (https://www.amazon.com/gp/product/B079FKJ4S3) or these (https://www.digikey.com/product-detail/en/global-specialties/WK-2/BKWK-2-ND/5231341), can help a lot and save you the pain of cutting and stripping wire)

Step 1 : Start with the resistors.

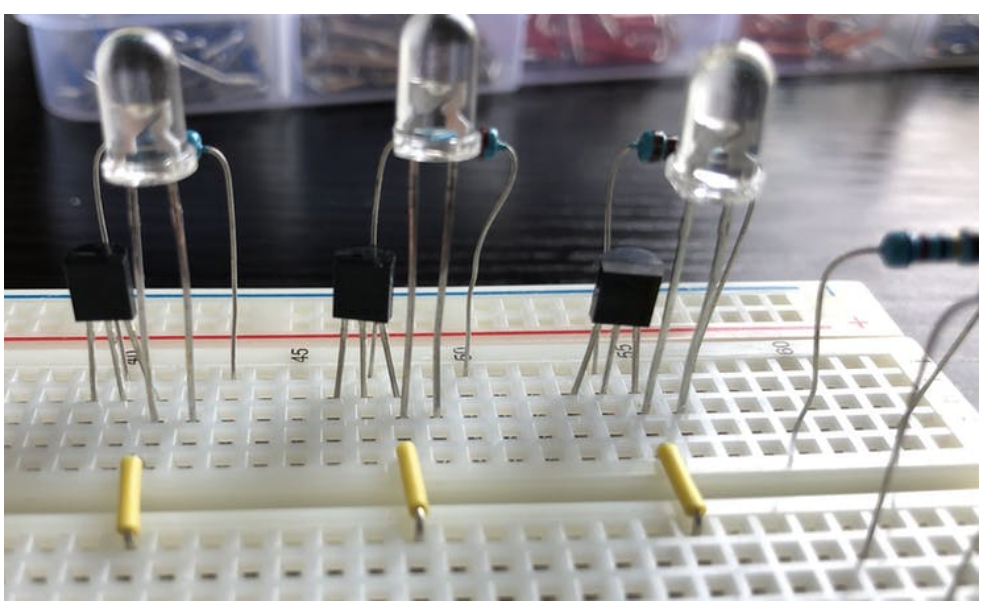




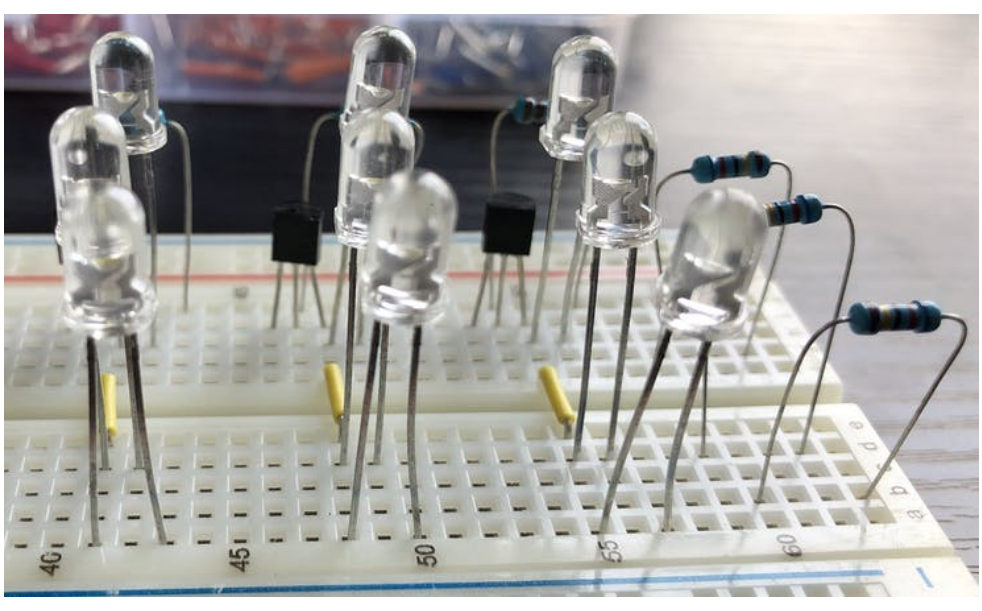
Step 2 : Add the transistors.



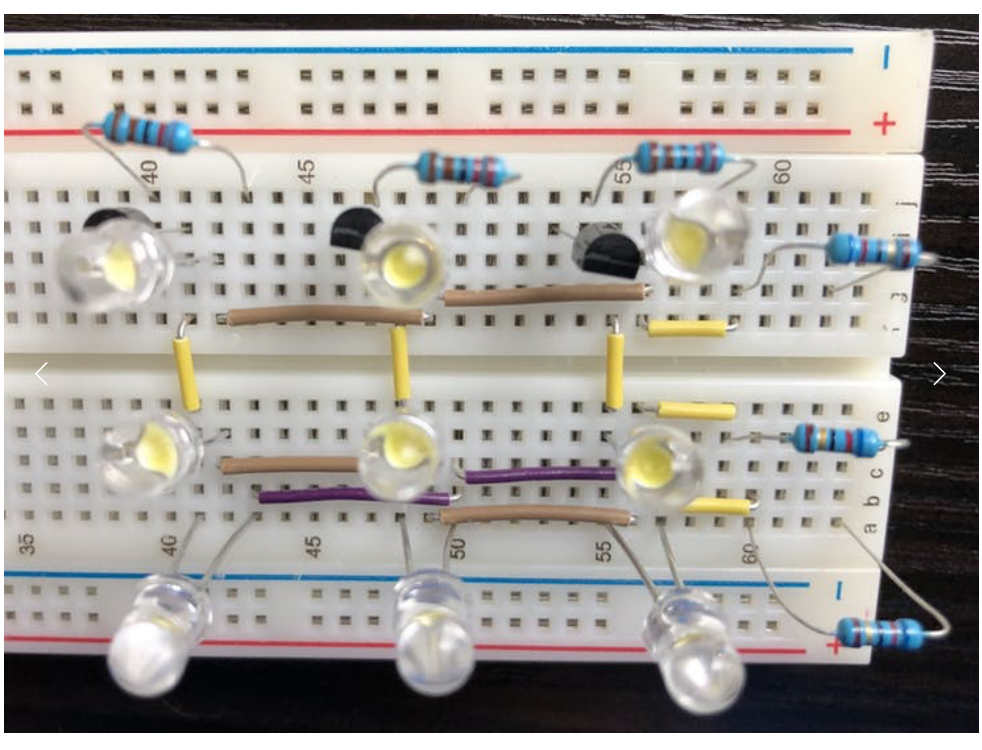
Step 3 : Add bridging wires and Row 1 LEDs. Note the politarity of the LEDs. The cathode (shorter leg) should share a node with the collector (right most leg) of the NPN BJT.
Pro-tip : Add all of the wires through Step 5 and then go back and add the LEDs. It's way easier.



Step 4 : Add remaining LEDs.

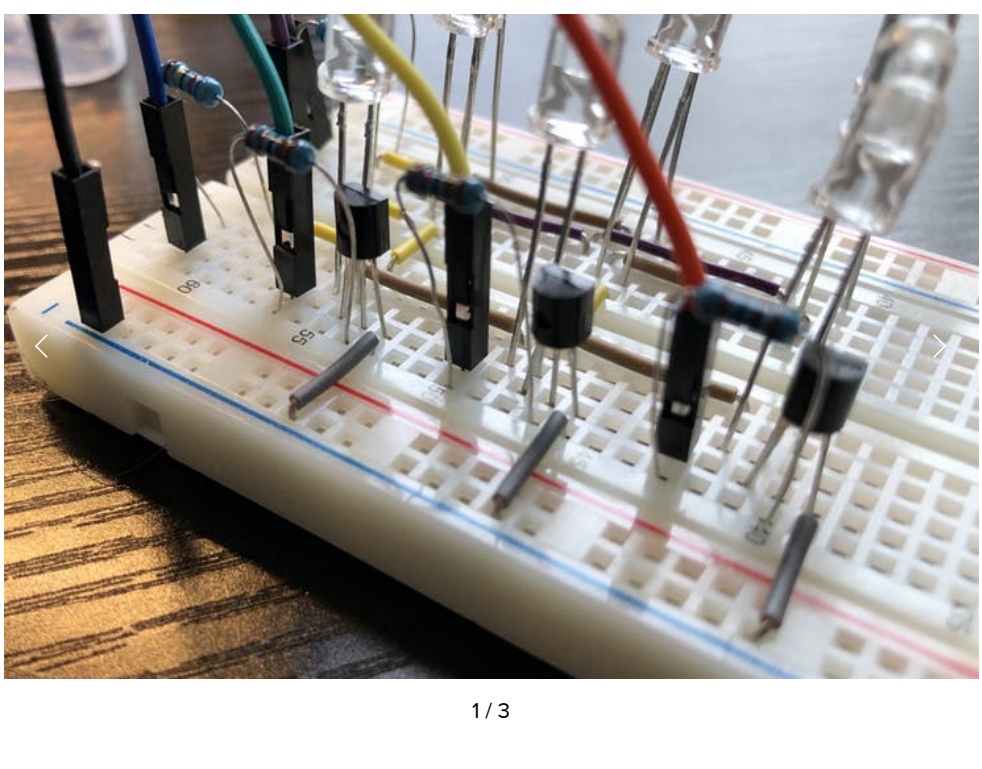


Step 5 : Add wires connecting LED anodes to each other and to current limiting resistors. The brown wires here connect the LED anodes in rows 1 and 2 while the purple wires connect the LED anodes in row 3.



1 / 2

Step 6 : Connect transistor emitters to ground (GND) and wire to Arduino (the second pic is missing GND connection to Arduino - the black jumper in the corner of the breadboard shown in the first image).



1 / 3

Great. Now it's time to program the Arduino. Use the following code and upload it from the Arduino IDE on your computer to the Arduino board.

```
// Code partially based on "Transistor Multiplexing 3x3 LEDs" by Marty Peltz
(https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-I-ma
de/)
#define COL_1_CATHODE 2
#define COL_2_CATHODE 3
#define COL_3_CATHODE 4
#define ROW_1_ANODE 5
#define ROW_2_ANODE 6
#define ROW_3_ANODE 7
#define LIGHT_LEVEL_INIT 200
#define MAX_BRIGHTNESS 999
#define PATTERN_TIME_INIT 20
int brightness = LIGHT_LEVEL_INIT;
int pattern_time = PATTERN_TIME_INIT;
void setup() {
 Serial.begin(9600);
 // Setup LED/transistor pins digital write pins
 pinMode(COL_1_CATHODE, OUTPUT);
 pinMode(COL_2_CATHODE, OUTPUT);
 pinMode(COL_3_CATHODE, OUTPUT);
 pinMode(ROW_1_ANODE, OUTPUT);
 pinMode(ROW_2_ANODE, OUTPUT);
```

Though the code is severely lacking in comments, I highly suggest that you step through it to understand how an LED multiplexer works. A web search for descriptions in other projects may do some good as well.

If all was successful, your system should look like this:

If one or more LEDs does not come on as it should, debugging can be made easier by enabling all of the LEDs at the same time. To do this, comment out all of the function calls to `multiplex_LEDs()` that are in `loop()` and uncomment the one at the bottom:

```
multiplex_LEDs(0b111, 0b111, 0b111);
```

If you want to understand (at least in part) how I am mapping the change in the potentiometer resistance to the speed or brightness in `map_brightness_value()`, take a look at Paul McWhorter's video (https://www.youtube.com/watch?v=QsYcYknKbB0) (most of his videos seem pretty great - very informative and well explained). `map_speed_value()`

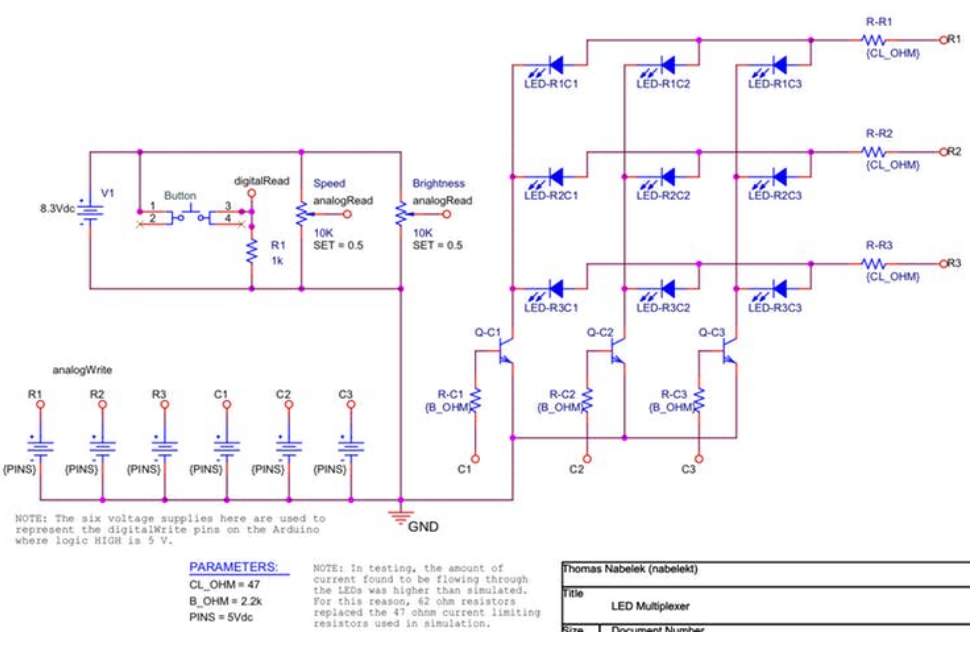
pretty great – very informative and well explained). `map_speed_value()` uses a power equation that I devised rather than a linear equation, but the basic idea is the same.

You may notice that adjusting the brightness can effect the speed of the pattern as well. I believe this is because of the way that the `multiplex_LEDs()` function is written. If the function were instead written to use clock time rather than iteration count for `pattern_time` in determining how long a particular LED configuration is held, I believe this issue would be resolved.

Stage 1C

In this stage we continue with the circuit completed in the last stage and add a pushbutton, two potentiometers, a toggle switch, and a battery to make the circuit a bit more interesting and independent of power provided by the computer via USB.

Here's the full circuit schematic:



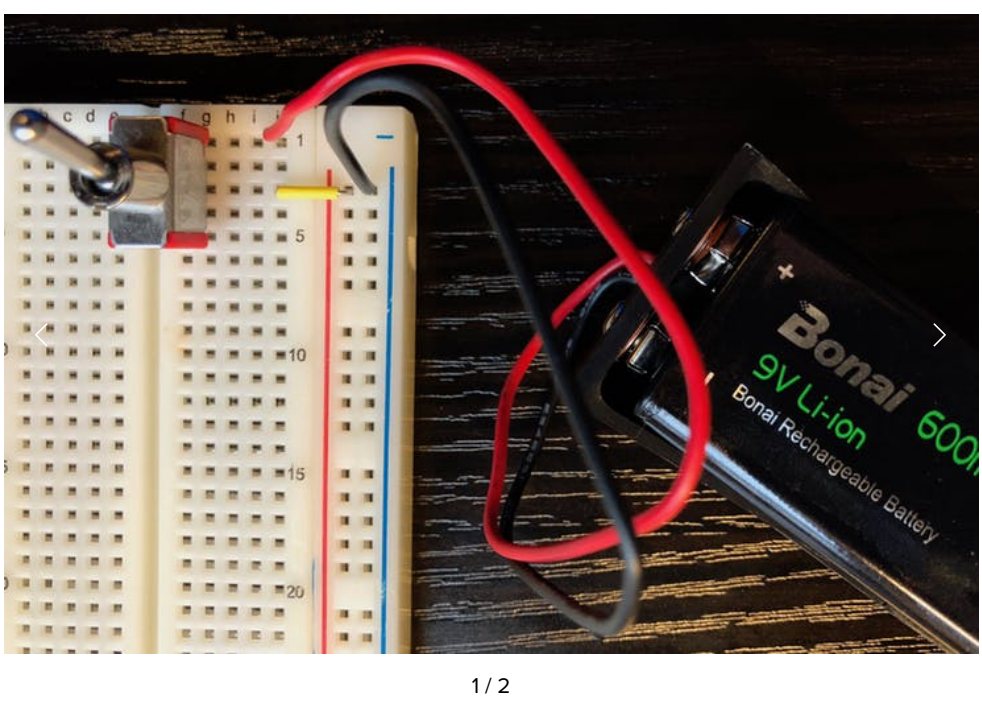
You may notice that the voltage on the positive breadboard rail is unregulated – it comes straight from the battery. Because the only components connected to that node are the switch, button, and potentiometers, this is not concerning. If other components such as the LEDs were also connected to this rail, it would probably be advisable to regulate the voltage through the Arduino using the Arduino's Vin and 5V pins (the Arduino accepts 7-12 V and regulates it down to 5 V). One thing I learned in doing this project is that "9V" batteries can range in voltage, typically 7.2-9.6 V depending on its type, when fully charged (see here). I measured one of my Li-ion batteries at 8.33 V when fully charged.

Now let's continue finish the circuit started in Stage 1A/B.

Component List:

- 9 V battery (I'm using one of these (http://www.amazon.com/gp/product/B0746FV8BF))
- Battery connector like these (http://www.amazon.com/gp/product/B00BXX42LQ) or this (https://www.digikey.com/product-detail/en/mpd-memory-protection-devices/BH9VW/BH9VW-ND/221547)
- Toggle switch (https://www.digikey.com/product-detail/en/e-switch/100SP1T1B4M2QE/EG2355-ND/378824)
- Pushbutton (https://www.digikey.com/product-detail/en/te-connectivity-alcoswitch-switches/1-1825910-0/450-1652-ND/1632538)
- 10 kΩ
- 2x 10 kΩ potentiometer (https://www.digikey.com/product-detail/en/tt-electronics-bi/P120PK-Y25BR10K/987-1710-ND/5957454)
- wire (jumper wires like these, (https://www.digikey.com/product-detail/en/PRT-12795/1568-1512-ND) and these (https://www.amazon.com/gp/product/B079FKJ4S3)or these, (https://www.digikey.com/product-detail/en/global-specialties/WK-2/BKWK-2-ND/5231341) can help a lot and save you the pain of cutting and stripping wire)

Step 1 : Add the toggle switch and battery on the left side of the breadboard from Stage 1A/B.

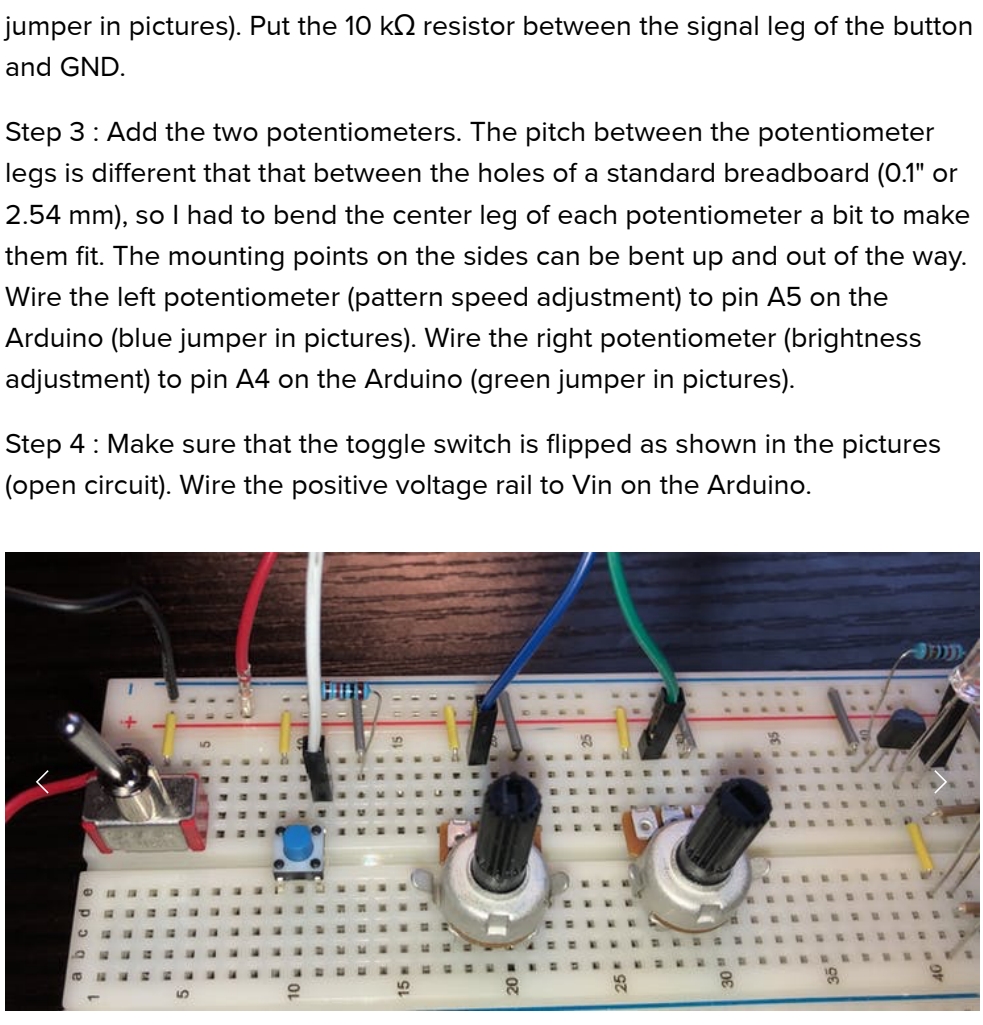


1 / 2

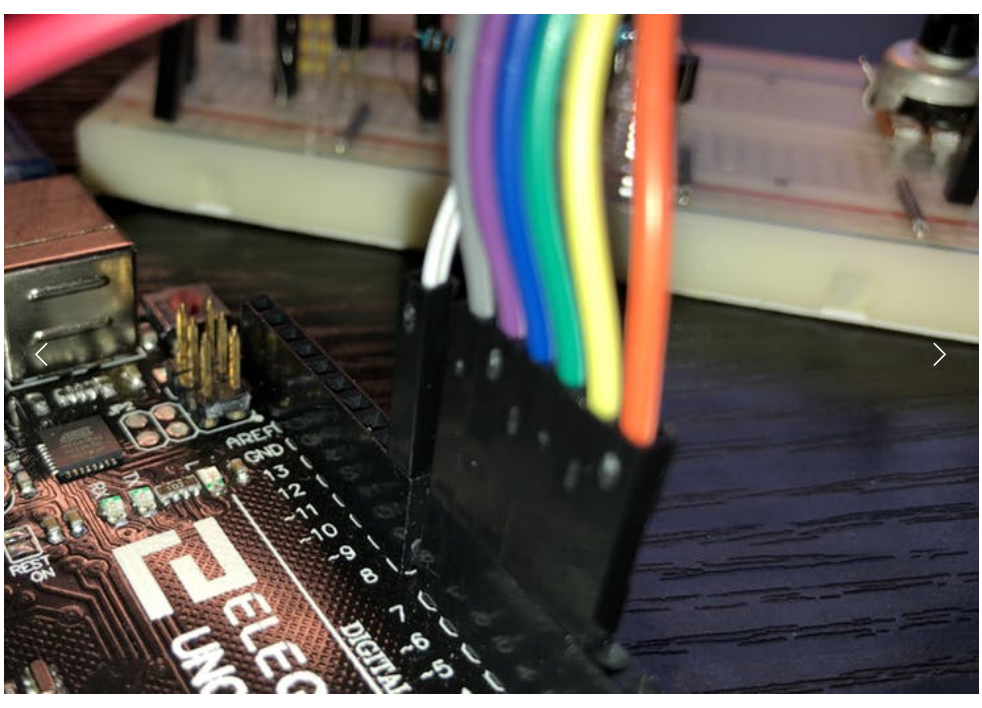
Step 2 : Add the pushbutton. Wire the signal to pin 8 on the Arduino (white jumper in pictures). Put the 10 kΩ resistor between the signal leg of the button and GND.

Step 3 : Add the two potentiometers. The pitch between the potentiometer legs is different that that between the holes of a standard breadboard (0.1" or 2.54 mm), so I had to bend the center leg of each potentiometer a bit to make them fit. The mounting points on the sides can be bent up and out of the way. Wire the left potentiometer (pattern speed adjustment) to pin A5 on the Arduino (blue jumper in pictures). Wire the right potentiometer (brightness adjustment) to pin A4 on the Arduino (green jumper in pictures).

Step 4 : Make sure that the toggle switch is flipped as shown in the pictures (open circuit). Wire the positive voltage rail to Vin on the Arduino.



1 / 3



1 / 2

Step 5 : Program the Arduino. The code is made up of three separate files. The directory/file structure on your computer should look something like this:

LED_Multiplexer
|––– LED_multiplexer.h
|––– LED_multiplexer_arduino.ino
|––– patterns.c

LED_multiplexer_arduino.ino is the one that you will need to hit the upload button on in the Arduino IDE.

LED_multiplexer.h:

```
#pragma once // Prevent cyclic inclusion
#include <stdbool.h>
bool button_pushed;
void multiplex_LEDs(int, int, int);
bool update_rows(int);
void check_button();
int map_speed_value(int);
int map_brightness_value(int);
void pattern_A();
void pattern_B();
void pattern_C();
void pattern_D();
void pattern_E();
void pattern_F();
void pattern_G();
void pattern_H();
```

LED_multiplexer_arduino.ino:

```
// Code partially based on "Transistor Multiplexing 3x3 LEDs" by Marty Peltz
(https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-I-ma
de/)
#include "LED_multiplexer.h"
#include "patterns.c"
#include <math.h>
#define COL_1_CATHODE 2
#define COL_2_CATHODE 3
#define COL_3_CATHODE 4
#define ROW_1_ANODE 5
#define ROW_2_ANODE 6
#define ROW_3_ANODE 7
#define BUTTON_PIN 8
#define BRIGHTNESS_POT_PIN A4
#define SPEED_POT_PIN A5
#define LIGHT_LEVEL_INIT 100
#define MAX_BRIGHTNESS 1000
#define PATTERN_TIME_INIT 20
#define NUM_PATTERNS 8
// Setup pattern function pointers
void (*patterns[NUM_PATTERNS])(void) =
 {&pattern_A, &pattern_B, &pattern_C, &pattern_D, &pattern_E,
```

patterns.c:

```
#include "LED_multiplexer.h"
void pattern_A() {
 // Single light on around in a circle CW with center light lit
 multiplex_LEDs(0b001, 0b010, 0b000);
 multiplex_LEDs(0b010, 0b010, 0b000);
 multiplex_LEDs(0b100, 0b010, 0b000);
 multiplex_LEDs(0b000, 0b110, 0b000);
 multiplex_LEDs(0b000, 0b010, 0b100);
 multiplex_LEDs(0b000, 0b010, 0b010);
 multiplex_LEDs(0b000, 0b010, 0b001);
 multiplex_LEDs(0b000, 0b011, 0b000);
 button_pushed = false;
}
void pattern_B() {
 // Single light on around in a circle CCW with center light lit
 multiplex_LEDs(0b001, 0b010, 0b000);
 multiplex_LEDs(0b000, 0b011, 0b000);
 multiplex_LEDs(0b000, 0b010, 0b001);
 multiplex_LEDs(0b000, 0b010, 0b010);
 multiplex_LEDs(0b000, 0b010, 0b100);
 multiplex_LEDs(0b000, 0b110, 0b000);
 multiplex_LEDs(0b100, 0b010, 0b000);
```

Once the code is uploaded, unplug the USB cable from your computer and/or the Arduino. Flip the toggle switch. If you have been successful, your system should operate like this:

Congrats! Stage 1 and the first prototype adjustable LED multiplexer are complete!

Stage 2

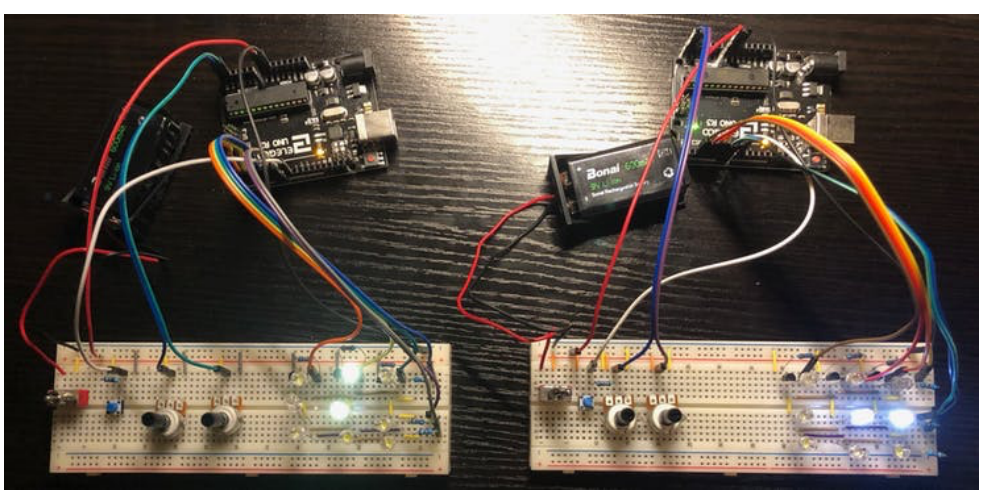
The end goal of this stage is to have the same system as that at the end of Stage 1 but with the Arduino replaced with only necessary components. However, it is good to test out our Stage 2 circuit with an Arduino before replacing the Arduino with a microcontroller and company. Repeat Stage 1 so that you have a duplicate prototype. Alternatively, you can make adjustments to the Stage 1 circuit and not retain the result of Stage 1. I wanted to keep the result of each stage, so I started Stage 2 by repeating almost exactly what was done in Stage 1, and I will assume here that you do the same:

Component List:

- Everything from the component lists of Stages 1A/B and 1C combined

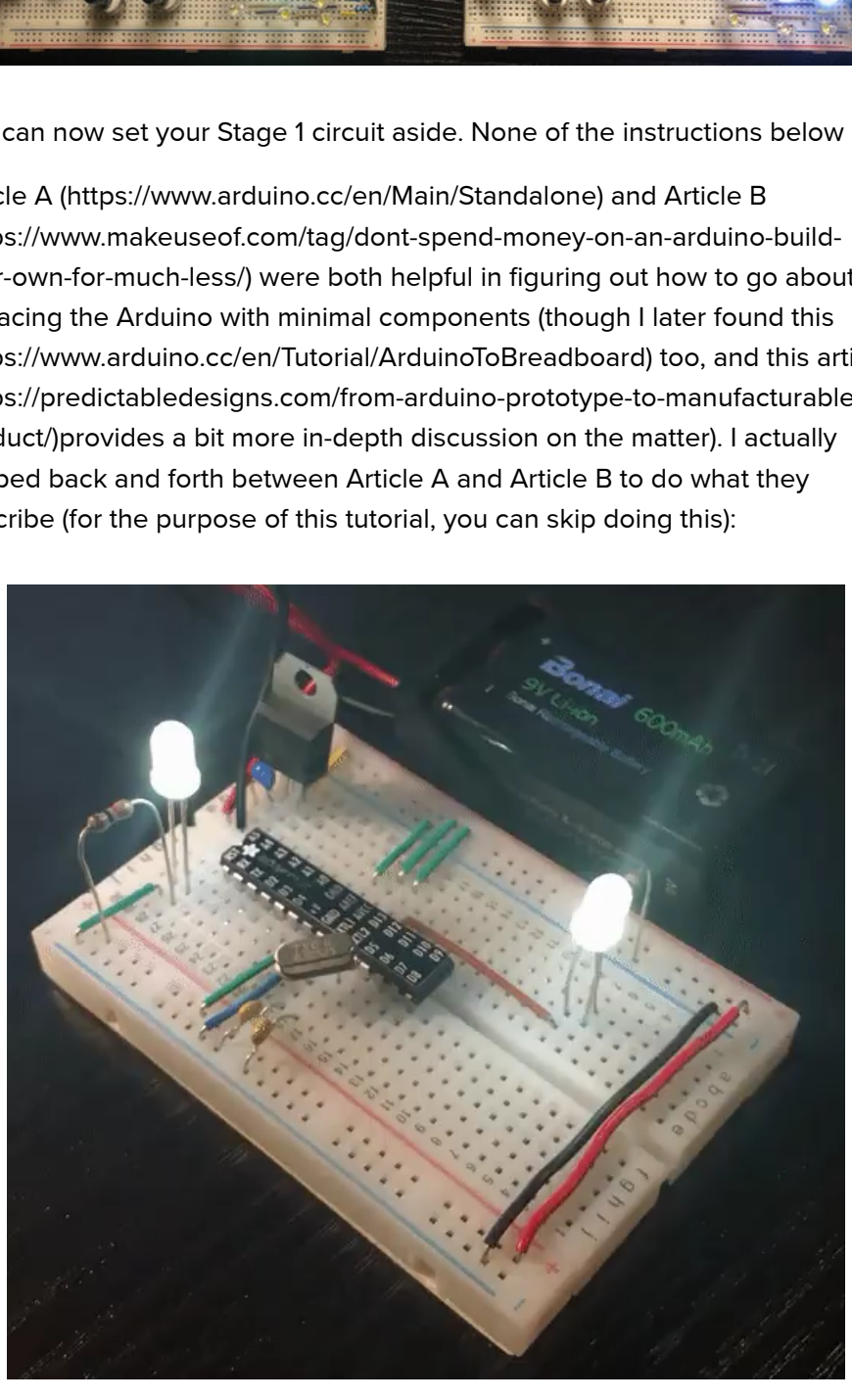
Steps : Follow the steps from Stages 1A/B and 1C, but note that in this second iteration, the toggle switch, pushbutton, and potentiometers should be a bit more crowded together on the left side in order to make room for the microcontroller unit (MCU) that we are going to replace the Arduino with in this stage.

You should now have duplicate systems:



You can now set your Stage 1 circuit aside. None of the instructions below a

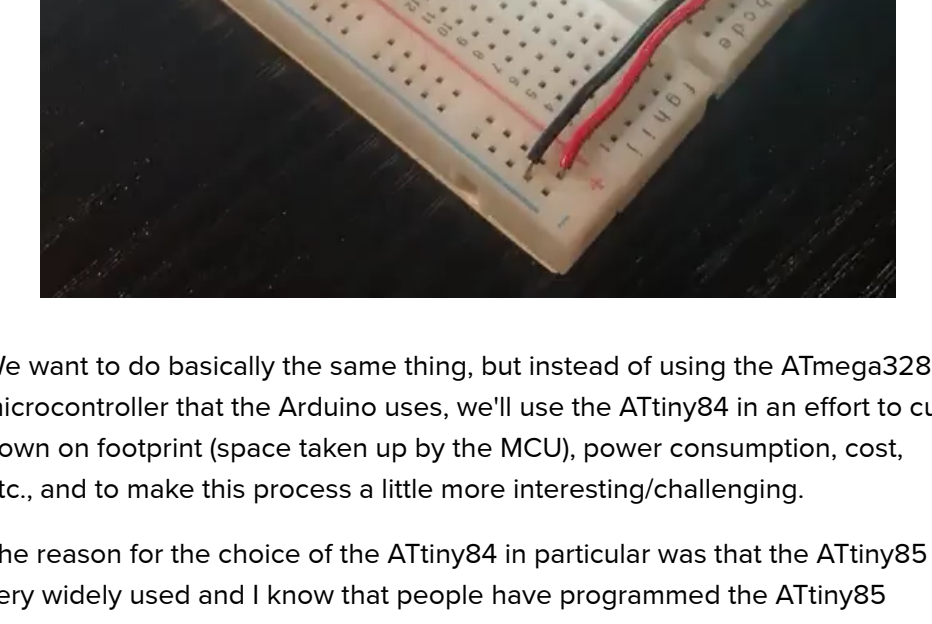
Article A (https://www.arduino.cc/en/Main/Standalone) and Article B (https://www.makeuseof.com/tag/dont-spend-money-on-an-arduino-build-your-own-for-much-less/) were both helpful in figuring out how to go about replacing the Arduino with minimal components (though I later found this (https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard) too, and this article (https://predictabledesigns.com/from-arduino-prototype-to-manufacturable-product/)provides a bit more in-depth discussion on the matter). I actually jumped back and forth between Article A and Article B to do what they describe (for the purpose of this tutorial, you can skip doing this):

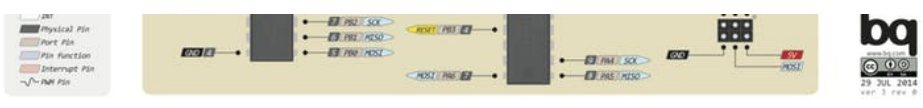


We want to do basically the same thing, but instead of using the ATmega328P microcontroller that the Arduino uses, we'll use the ATtiny84 in an effort to cut down on footprint (space taken up by the MCU), power consumption, cost, etc., and to make this process a little more interesting/challenging.

The reason for the choice of the ATtiny84 in particular was that the ATtiny85 is very widely used and I know that people have programmed the ATtiny85 using an Arduino like I wanted to, so I probably wouldn't run into to much trouble. The ATtiny84 is kind of the big brother of the ATtiny85. The 84 offers more I/O pins – which are needed for our adjustable LED multiplexer – but is otherwise very similar to the 85. Also, the maximum current ratings for the ATtiny84 – 40 mA per pin and 200 mA Vcc and GND as listed on page 174 of its datasheet – (http://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf) is the same as that of the Arduino (and its ATmega328P microcontroller). I initially planned to make use of the ATtiny24 but soon found that it did not have enough program memory (the program memory size, EEPROM size, and RAM size is the only difference that I know of between the ATtiny24, 44, and 84).

I first figured out how to program the Attiny84 by following this ATtiny85 video, (https://www.youtube.com/watch?v=i9WOwDrpRKs&t=150s) remapping the pins using the diagram below (view it larger here (https://raw.githubusercontent.com/nabelekt/LED_Multiplexer/master/stage_2/ATINY-PinOut-Diagram.png), we will use it later), and working with a different boards library in the Arduino IDE, as I will describe.





We will program the ATtiny84 using the SPI protocol (https://www.arduino.cc/en/reference/SPI) (see section 19.5 Serial Programming (page 163) of the ATtiny84 datasheet (http://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf)) and the Arduino used in the first part of Stage 2. Note that there are other ways to program an AVR MCU like the ATtiny84, such as using an AVRISP mkII (http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42093-AVR-ISP-mkII_UserGuide.pdf). I chose to use the method that I will outline in the procedure below because it uses the already familiar Arduino IDE, requires little or no code modification, and does not require any additional hardware.

Enough discussion for now, let's get to it:

Component List:

- A smaller solderless breadboard (https://www.digikey.com/product-detail/en/dfrobot/FIT0096/1738-1326-ND/7597069) (not absolutely necessary, but will make initial testing and debugging easier)
- ATtiny84 MCU (https://www.digikey.com/product-detail/en/microchip-technology/ATTINY84A-PU/ATTINY84A-PU-ND/2774082)
- 10 kΩ resistor
- 1 Cree C512A-WNN-CZ0B0152 LED (https://www.digikey.com/product-detail/en/cree-inc/C512A-WNN-CZ0B0152/C512A-WNN-CZ0B0152CT-ND/6138557) (you could borrow one from the 3x3 LED matrix)
- 2x 10 µF capacitor like this (https://www.digikey.com/product-detail/en/tdk-corporation/FG24X7R1A106KRT00/445-181259-ND/)
- L7805C 5V Linear Voltage Regulator (https://www.digikey.com/product-detail/en/stmicroelectronics/L7805CV/497-1443-5-ND/)
- 6 jumper wires like those used in previous stages

Step 1 : Unplug all jumper wires from the Arduino used in the first part of Stage 2 (your duplication of Stage 1's system).

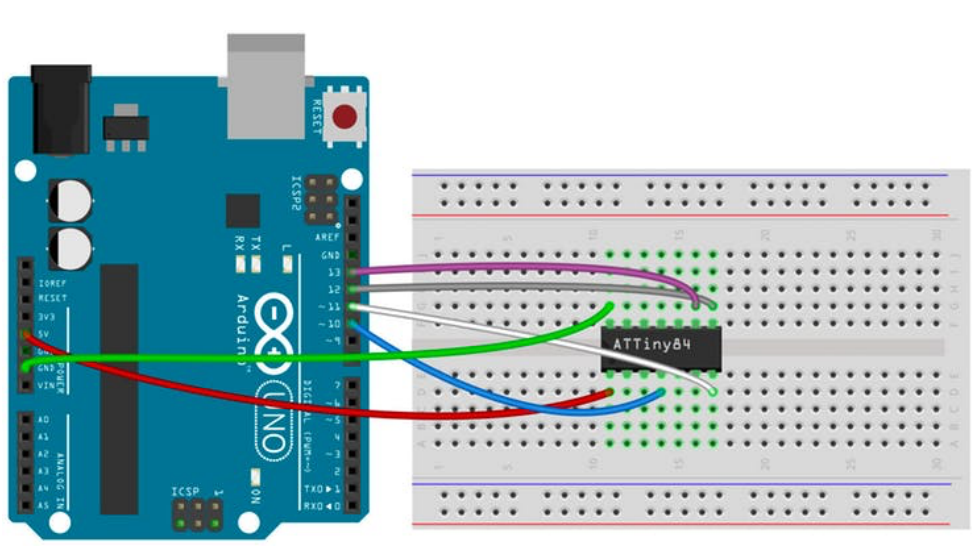
Step 2 : Plug the Arduino into your computer. In the Arduino IDE, select Tools > Board > Arduino/Genuino Uno. Find and select your Arduino under Tools > Port.

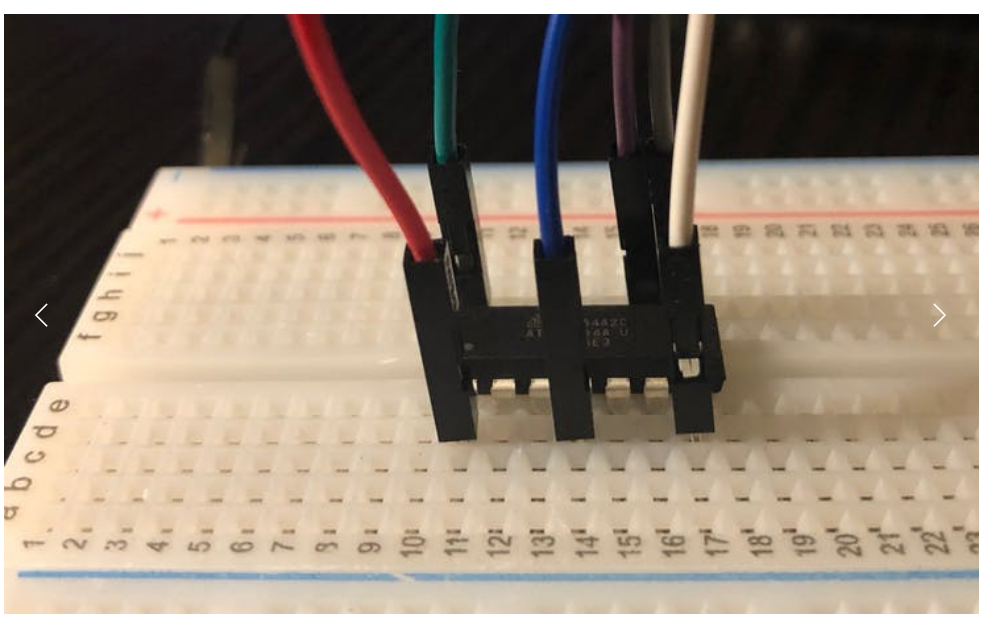
Step 3 : Choose File > Examples > 11.ArduinoISP > ArduinoISP. Upload this sketch to your board. This programs the Arduino to act as an ISP (In-System Programmer) which will allow us to use it to program our MCU.

Step 4 : Place the MCU on the small breadboard (not the same one with your circuit) bridging the gap.

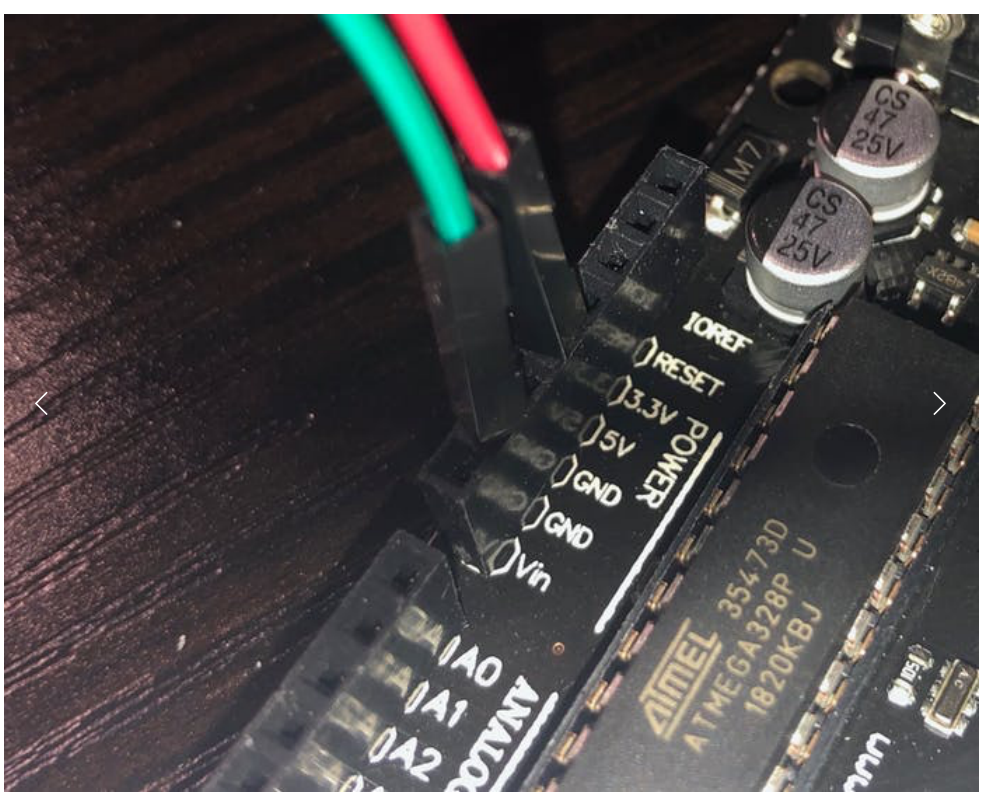
Step 5 : Disconnect the Arduino from your computer and make sure that it is not powered.

Step 6 : Refer to the diagram above and use the physical pin designations which are closest to the depiction of the MCU. E.g., pin 1 is in the top left corner of the MCU.
Connect pin 1 VCC to 5V on the Arduino.
Connect pin 4 RESET to pin 10 on the Arduino.
Connect pin 7 MOSI to pin 11 on the Arduino.
Connect pin 8 MISO to pin 12 on the Arduino.
Connect pin 9 SCK (labeled as "USCK" on the diagram) to pin 13 on the Arduino.
Connect pin 14 GND to pin GND on the Arduino.



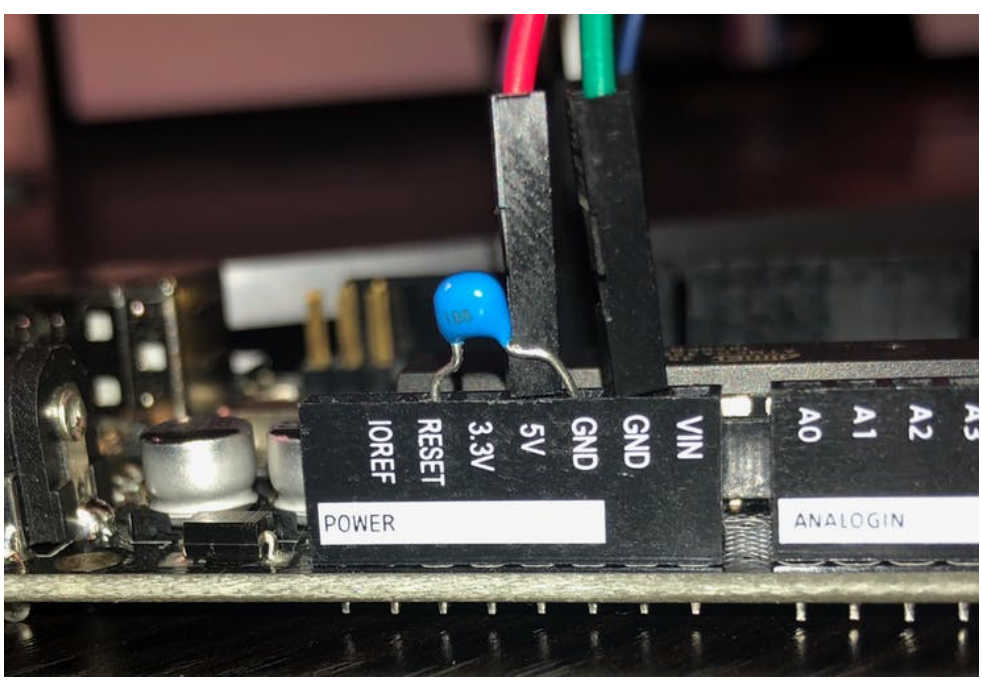


1 / 2



1 / 2

Step 7 : Wire a 10 µF capacitor between RESET and GND on the Arduino (may not be necessary, (https://forum.arduino.cc/index.php?topic=104435.0) but I spent over an hour learning the hard way that I needed it). My capacitors do not have polarity. If yours does, make sure that you connect the positive leg to RESET and the negative to GND.



Step 8 : In the Arduino IDE, open up the Preferences, pop open the Additional Boards Manager URLS window, and paste the following on its own line: `https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json` . Click OK and then close and reopen the Arduino IDE (I am not sure if this action is necessary or not).

Step 9 : Go to Tools > Board > Board Manager. Search for "attiny". You should see the “attiny by David A. Mellis” board library (https://github.com/damellis/attiny). (https://github.com/damellis/attiny) Select the latest version (I am using 1.0.2) and install. This adds support for ATtiny microcontrollers to the Arduino IDE. Once the install is complete, you may need to close and reopen the Arduino IDE again.

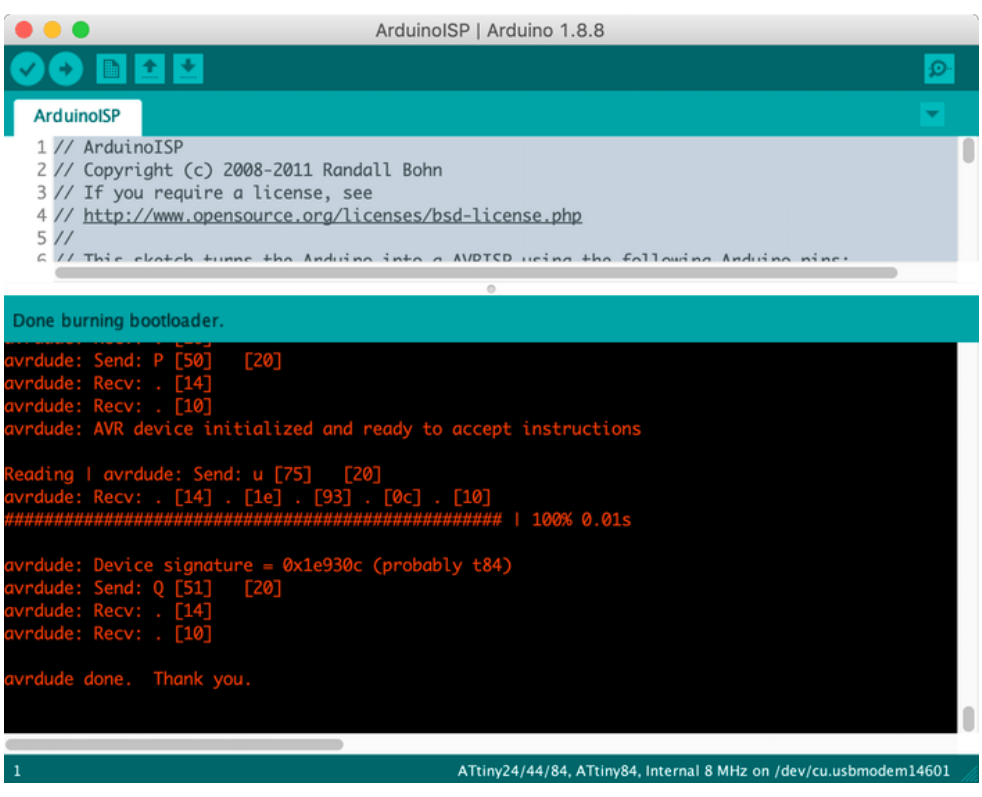
Step 10 : Go to Tools > Board and you should now see a section called "ATtiny Microcontrollers." Select "ATtiny24/44/84". Now, under Tools select Processor > ATtiny84 and Clock > Internal 8 MHz.

Unlike the breadboard Arduino recreation described in Article A and Article B thats I referenced above, we will use the oscillator internal to the ATtiny84 rather than an external clock/oscillator. I highly suggest that you search the interwebs and learn about the pros and cons of internal vs. external oscillators.

Step 11 : Plug the Arduino back into your computer. Find and select your Arduino under Tools > Port.

Step 12 : Select Tools > Programmer > Arduino as ISP.

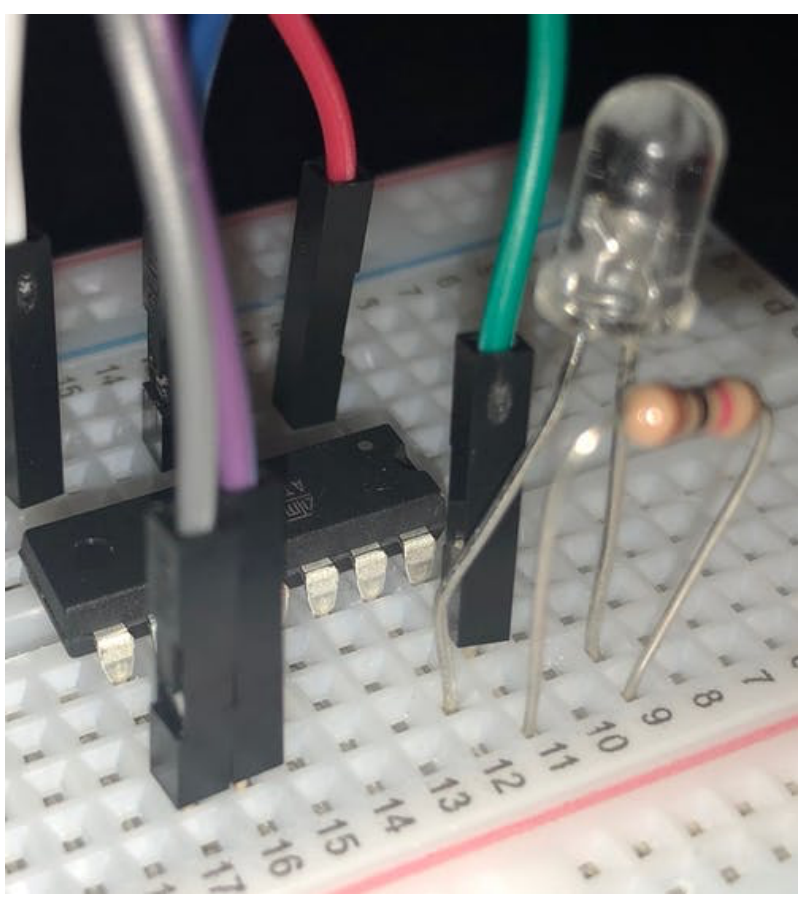
Step 13 : Select Tools > Burn Bootloader. If successful you should see something like the following (I have verbose output enabled in the Preferences – you probably should too):



As long as you should see "Done burning bootloader," you should be good to go. You have just programmed the MCU with the bootloader that will allow our code to run. You can read more about what the bootloader does here (https://www.arduino.cc/en/Hacking/Bootloader) and here (https://www.baldengineer.com/arduino-bootloader.html).

Step 14 : Disconnect the Arduino from your computer so that it is not powered.

Step 15 : Using the diagram above again, now use the pin designations that I have circled in yellow. These are the pin designations that we will use in our Arduino code. E.g., both pin 0 and pin A0 are what we previously considered to be pin 13. Plug the anode (longer leg) of the LED into the same node as pin 0 and the cathode into a node by itself. Put the 10 kΩ current limiting resistor between the cathode of the LED and GND.

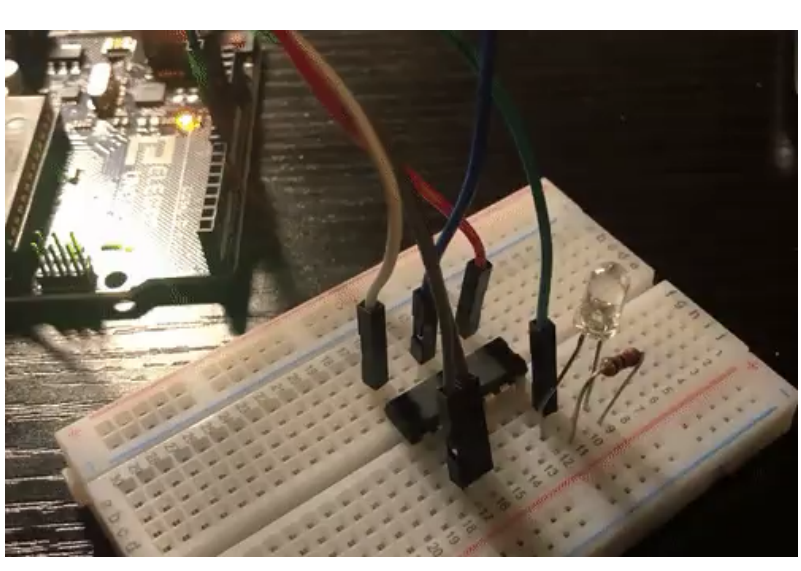


1 / 2

Step 16 : Upload the code below to the ATtiny84 just as you would to the Arduino itself – just make sure that the capacitor described in Step 7 is in place and that "ATtiny24/44/84", "ATtiny84", and "Internal 8 MHz" are still selected for Board, Processor, and Clock, respectively.

```
/*
 modified from http://www.arduino.cc/en/Tutorial/Blink
*/
#define LED_PIN 0
void setup() {
 pinMode(LED_PIN, OUTPUT);
}
void loop() {
 digitalWrite(LED_PIN, HIGH);   // turn the LED on (HIGH is the voltage leve
l)
 delay(500);                       // wait
 digitalWrite(LED_PIN, LOW);    // turn the LED off by making the voltage LO
W
 delay(500);                       // wait
}
```

If you have been successful, you should have the following:



We now know that we can program the ATtiny84. Excellent!

Using the internal oscillator means that the only primary components that we need to replace the Arduino are the voltage regulator – to drop the voltage supplied by the battery to the 5 V that the MCU can accept – and the MCU itself.

Let's return to the Stage 2 circuit that we built as a duplicate of the Stage 1 circuit.

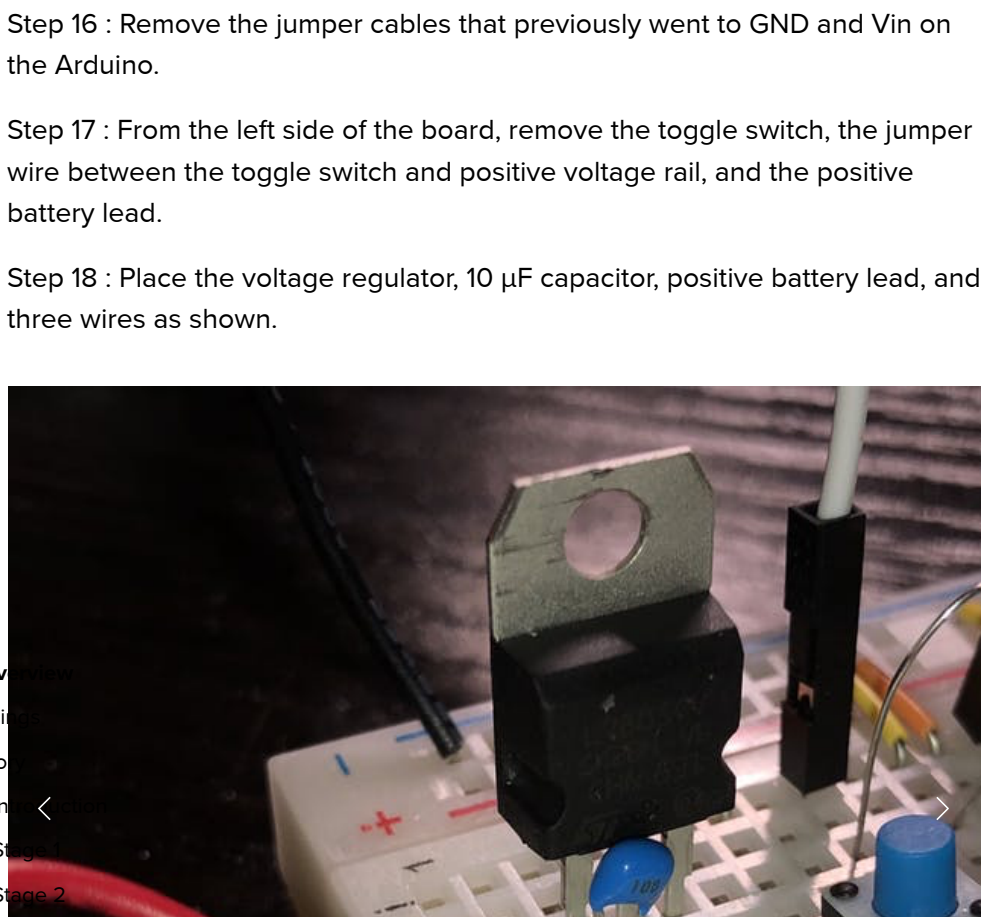
Step 16 : Remove the jumper cables that previously went to GND and Vin on the Arduino.

Step 17 : From the left side of the board, remove the toggle switch, the jumper wire between the toggle switch and positive voltage rail, and the positive battery lead.

Step 18 : Place the voltage regulator, 10 µF capacitor, positive battery lead, and three wires as shown.

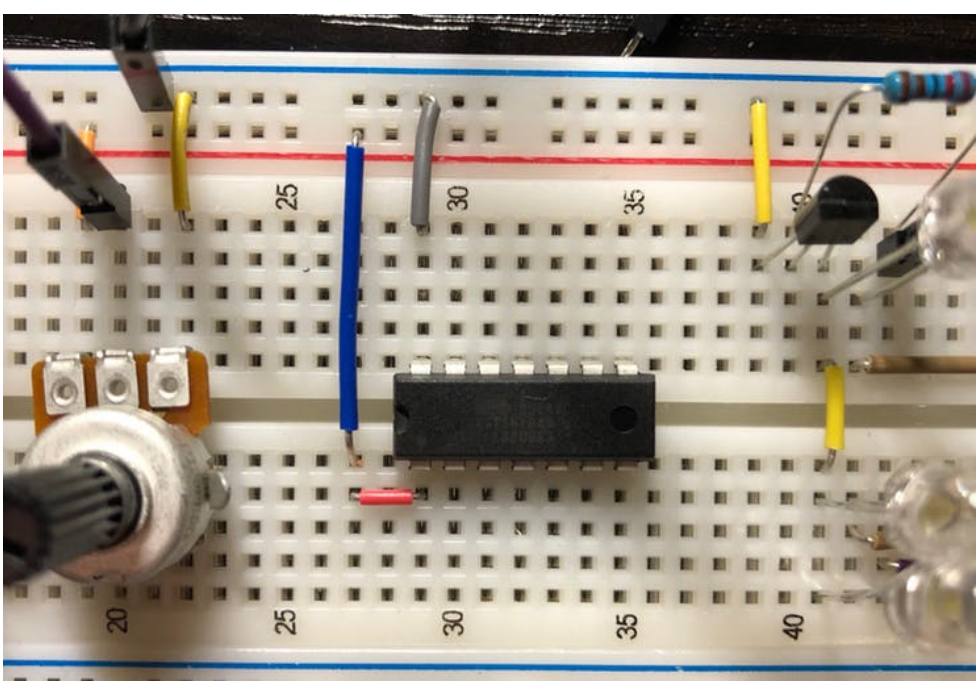
1 / 2

Step 19 : Replace the toggle switch in the new location. Ensure that it is flipped so that the power is off (open circuit).



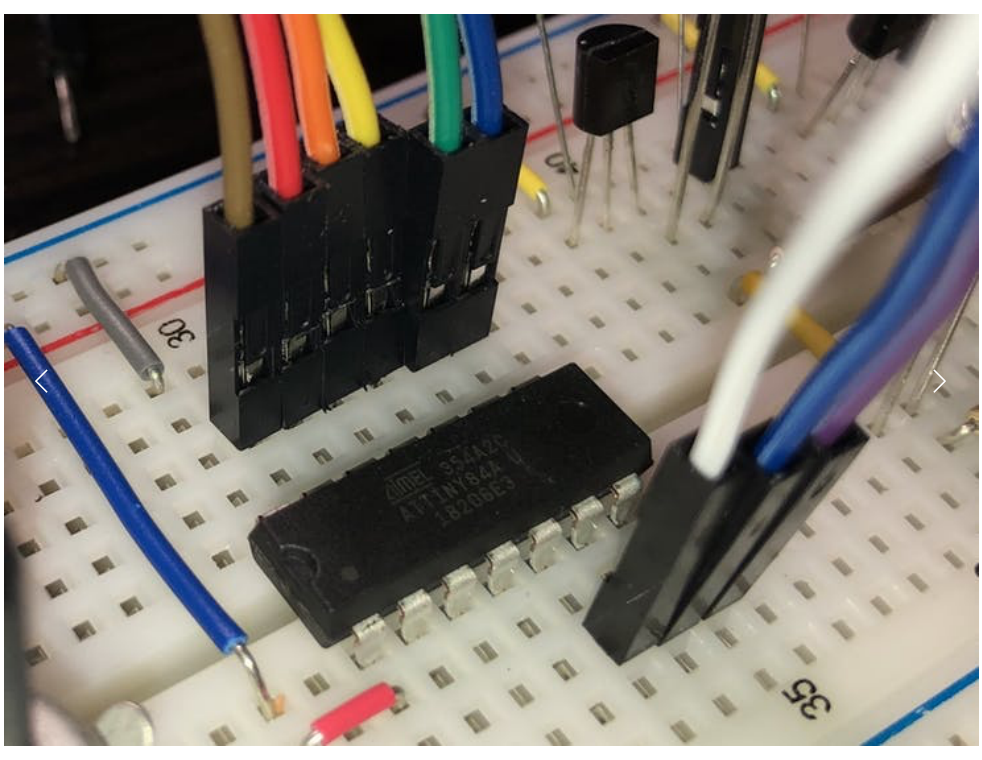
1 / 2

Step 20 : Place the MCU in the center of the board bridging the gap. Wire physical pin 0 VCC to the positive voltage rail and physical pin 1 GND to the ground rail.



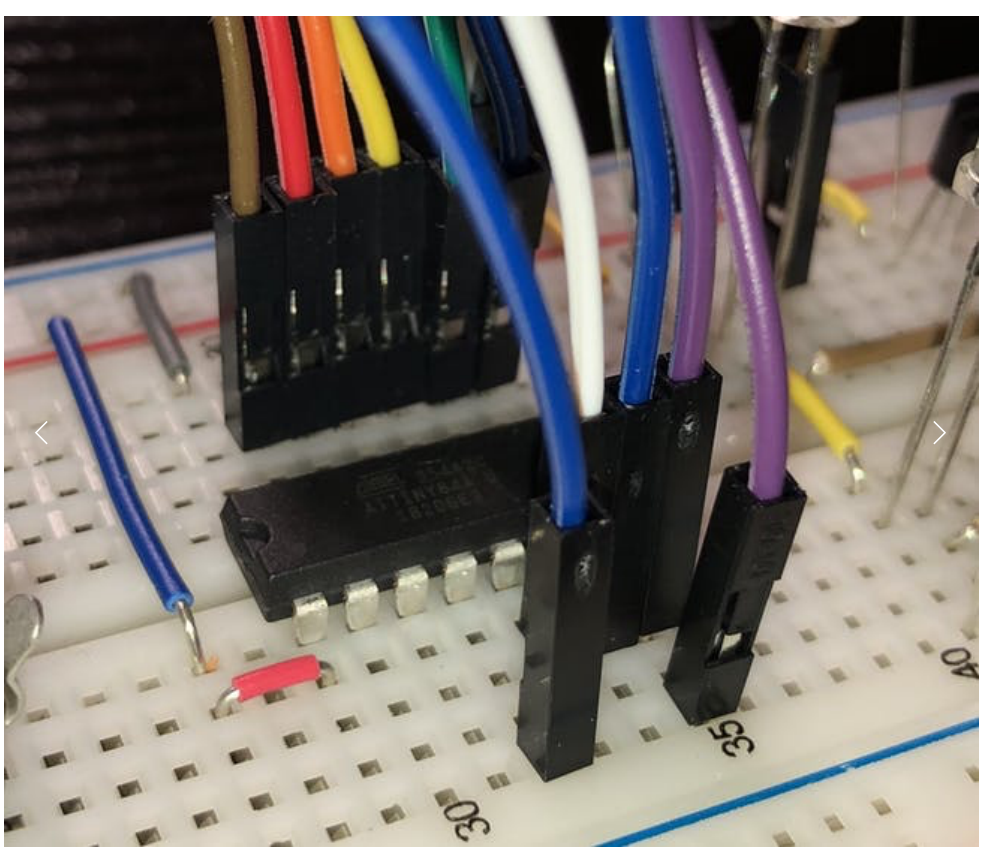
Step 21 : As in Step 6 above, refer to the pinout diagram and use the physical pin designationsof the MCU. E.g., pin 1 is in the top left corner of the MCU. Connect pin

*** Add pictures showing where these colored wires lead



1 / 2

Step 22 : Repeat Step 6 above, but note that pin 2 VCC and pin 14 GND are already wired as needed.
Connect pin 4 RESET to pin 10 on the Arduino.
Connect pin 7 MOSI to pin 11 on the Arduino.
Connect pin 8 MISO to pin 12 on the Arduino.
Connect pin 9 SCK (labeled as "USCK" on the diagram) to pin 13 on the Arduino.
Wire the ground rail to pin GND on the Arduino.



1 / 3

Initially, I made the mistake of regulating the voltage only for the MCU. This caused the input voltage on every button press to be ~8 V at the button pin. This too-high voltage would reset the MCU so that I got the following behavior:

This issue required some debugging to figure out and is the reason why we are regulating the voltage for the whole system. Fortunately, I don't think I fried anything.

We must change the assigned pins

Use the three code files from Step 5 of Stage 1C, but change the defined pins in LED_multiplexer_arduino.ino (I called this one LED_multiplexer_attiny84.ino) to be:

```
// Code partially based on "Transistor Multiplexing 3x3 LEDs" by Marty Peltz
(https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-I-ma
de/)

#include "LED_multiplexer.h"
#include "patterns.c"
#include <math.h>
#include <SoftwareSerial.h> // Used for printing to Serial Monitor for debug

#define COL_1_CATHODE 0
```

Some of your LEDs will probably blink rapidly while the MCU is being programmed. This is a result of the logic HIGH and LOW bits being written to the MCU's program memory also applying voltage to the anodes/cathode branches of the LED multiplexer.

Schematics

Project Repository (https://github.com/nabelekt/LED_Multiplexer)

0 (https://github.com/nabelekt/LED_Multiplexer/watchers) **nabelekt/LED_Multiplexer (https://github.com/nabelekt/LED_Multiplexer)**
0 (https://github.com/nabelekt/LED_Multiplexer/forks)

No description — Read More (https://github.com/nabelekt/LED_Multiplexer#readme)

Latest commit to the master branch on 2-6-2019 **Download as zip (https://github.com/nabelekt/LED_Multiplexer/archive/master.zip)**

Credits

(/thomas-nabelek) **Thomas Nabelek (/thomas-nabelek)**
1 project • 0 followers

Follow **Contact (/messages/new?recipient_id=877826)**

Comments

Write Preview

Share your thoughts! What do you like about this project? How could it be improved? Be respectful and constructive – most Hackster members create and share personal projects in their free time.

Post

Be the first to comment!

Similar projects you might like

There are no projects.