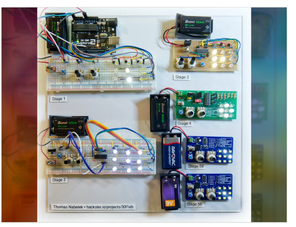
Thomas Nabelek (/nabelekt)
Created January 22, 2019 © GPL3+ (http://opensource.org/licenses/GPL-3.0)

Arduino Prototype to Manufacturable PCB: An LED Multiplexer

Learn the electronic and software design steps to take an idea from Arduino-based breadboard prototype to manufacturable consumer device.

Intermediate(/projects?difficulty=intermediate) Work in progress 66



Things used in this project

Hardware components

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

Software apps and online services

Arduino IDE (/arduino/products/arduino-ide?ref=project-30f1eb) (https://www.arduino.cc/en/main/software)

DigiKey Electronics (https://www.digikey.com)

PCBWay (http://pcbway.com)

Hand tools and fabrication machines

See tool lists in each stage below.

Story

Introduction

My goal of this project was to teach myself – with the assistance of many helpful folks on the interwebs – how to take an Arduino-based breadboard prototype device all the way to a mass-producible PCB (printed circuit board) device. I created this guide so that you can follow along and hopefully replicate my work if you wish to learn the same thing!

Together we will create an LED multiplexer circuit that shows different patterns with the LEDs, that can adjusted for different speed and brightness, and that can be reprogrammed at a later date to modify the patterns, etc.

The idea of this project was not to create an adjustable and programmable LED multiplexer device, but rather to learn the process and bare-bones electronics and software skills needed to develop a manufacturable (i.e., in a factory) consumer electronic device. I chose the LED multiplexer as a circuit that was simple enough to not bog down progress of achieving my main goal, but complex enough to be a little more interesting and challenging than a few blinking LEDs.

I thought that I would brake the prototyping process down into five stages:

Stage 1) A breadboard circuit dependent on an Arduino to function

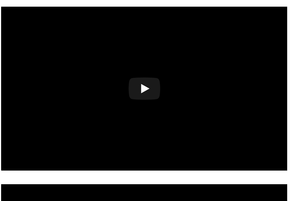
Stage 2) Stage 1 with the Arduino replaced with only necessary components

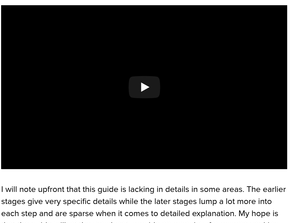
Stage 3) Stage 2 soldered together using stripboard

Stage 4) A PCB using through hole components that I would assemble myself

Stage 5) A smaller PCB using surface mount components that would be assembled by myself and in a factory

Here are a couple demo videos of the completed project:





I will note upfront that this guide is lacking in details in some areas. The earlier stages give very specific details while the later stages lump a lot more into each step and are sparse when it comes to detailed explanation. My hope is that the guide will, at the very least, provide 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.

Some previous experience with Arduinos (https://www.arduino.cc/en/Guide/Introduction) (we'll use the Arduino Uno) and the Arduino IDE (https://www.arduino.cc/en/main/software) will be beneficial to you, but if you're not sure about something, there are lots of helpful resources out there. Previous experience building circuits with breadboards is assumed – it's pretty simple, just know the node layout of your board. Some knowledge of the C programming language is assumed and is important if you want to understand what the code is doing.

I highly encourage you to read through this entire tutorial before beginning. If you are lazy like me, you won't want to do that, but doing so will allow you to know what you are getting yourself into, decide which stages you think will be beneficial for you to complete, apply some of the lessons I learned from the very beginning, know which components/parts and tools you will want to order up front, etc.

Throughout this project, I would encourage you to use the same color jumper wires that I use. Doing so will make the pictures more helpful to you.

Stage 1

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

Stages 1A and 1B

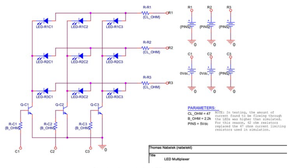
Designing the Circuit

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).

This article (http://lednique.com/display-technology/multiplexed-display/) gives a bit of explanation about LED multiplexing. In short, multiplexing allows us to control LEDs with fewer signal pins than we have LEDs. We use transistors as gates to select a single row and column to be enabled so that the LED in that row and column will be ON when we want it to be. We switch between LEDs fast enough that the human eye or a camera perceives multiple LEDS to be on at the same time. The number of LEDs we can do this for 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 signals = 2*sqrt(9 LEDs), 10 signals = 2*sqrt(25 LEDs)

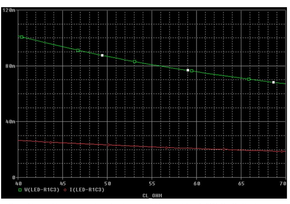
After some trial, error, and iteration, I settled on this simple design:



Circuit schematic for LED multiplexer

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

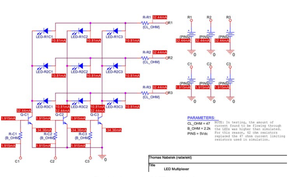
I wanted the LEDs to be as bright as possible without burning out. I was using Cree C512A-WNS/WNN LEDs, so I took a look at their datasheet (https://www.cree.com/led-components/media/documents/C512A-WNS-WNN-942.pdf)and found the maximum forward current that these LEDs can withstand to be 25 mA and the 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 example, I did a sweep of what I have labeled as the CL_OHM resistance value in the schematic to see the current and power dissipation through/at LED-R1C3:



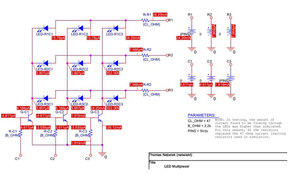
Current (red, mA) and power dissipation (green, mW) through/at LED-R1C3; CL_OHM axis values given in ohms (Ω)

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.

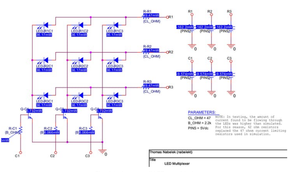
I created the schematic and did the simulation using OrCAD PSpice Designer Lite. (https://www.orcad.com/resources/download-orcad-lite) The software lets you simulate the circuit and get the voltage at any node, current on any branch, and power consumption for any component. Here are a bunch of simulated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



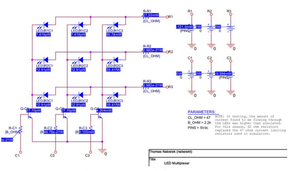
Simulated current, all LEDs enabled



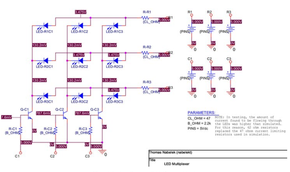
Simulated current, only LED-R1C3 enabled



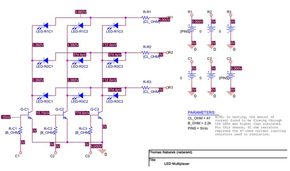
Simulated power, all LEDs enabled



Simulated power, only LED-R1C3 enabled



Simulated voltage, all LEDs enabled



Simulated voltage, only LED-R1C3 enabled

Notice, however, 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 Ω resistors replaced the 47 Ω 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 (https://en.wikipedia.org/wiki/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

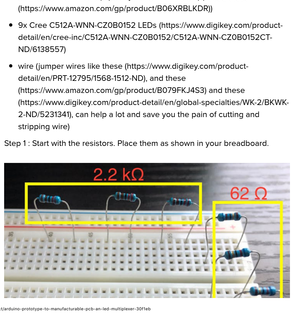
Tools Needed:

- Computer with Arduino IDE
- USB cable to connect to Arduino to computer

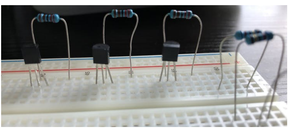
Component list:

- 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 (bipolar junction 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) and 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)

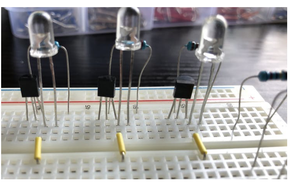
Step 1 : Start with the resistors. Place them as shown in your breadboard.



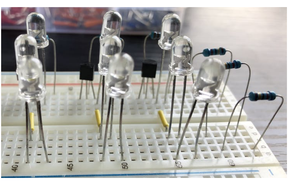
Step 2 : Add the transistors.



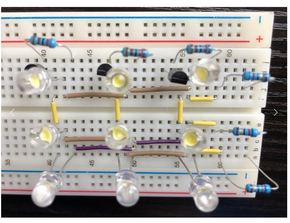
Step 3 : Add bridging wires and Row 1 LEDs. Note the polarity of the LEDs. The LED's cathode (shorter leg) should share a node with the NPN BJT's collector (right most leg when facing the flat part).
Pro-tip : Add all of the wires through Step 5 and then go back and add the LEDs. It's way easier.



Step 4 : Add remaining LEDs.

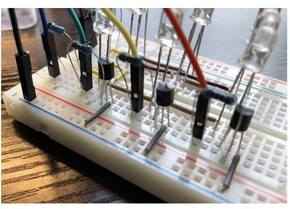


Step 5 : Add wires connecting the LED anodes to each other and to the current limiting resistors (the 62 Ω ones). The brown wires here connect the LED anodes in rows 1 and 2 while the purple wires connect the LED anodes in row 3. The yellow wires running horizontally



1 / 2

Step 6 : Wire the transistor emitters (left most pin when looking towards the flat side of the transistors) to the ground rail. These are the grey wires in the first pi
Wire to the Arduino. I will use the resistor designations found in the circuit diagrams above, but it is probably easiest to look at the pictures below.
Connect a jumper from R-R3 to pin 7 on the Arduino (gray jumper).
Connect a jumper from R-R2 to pin 6 on the Arduino (purple jumper).
Connect a jumper from R-R1 to pin 5 on the Arduino (blue jumper).
Connect a jumper from R-C3 to pin 4 on the Arduino (green jumper).
Connect a jumper from R-C2 to pin 3 on the Arduino (yellow jumper).
Connect a jumper from R-C1 to pin 2 on the Arduino (orange jumper).
Connect a jumper from the ground rail to GND on the Arduino (the second pic is missing the ground connection to Arduino – the black jumper in the corner of the breadboard shown in the first image).



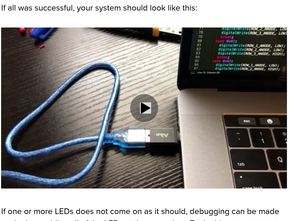
1 / 3

Step 7 : 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. I recommend getting the code from the GitHub repository (repo) (https://github.com/nabelekkt/LED_Multiplexer). When I copy and paste it here, all of the blank lines and indentations are removed. This is the code from the code/LED_multiplexer_only/LED_multiplexer_only.ino in the repo.

```
// Code partially based on "Transistor Multiplexing 3x3 LEDs" by Marty Peltz
(https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-1-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 better 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);  // Enable all LEDs together
```

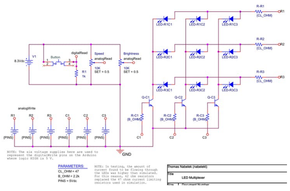
If you want to understand (at least in part) how I am mapping the change in the potentiometer resistance to the speed or brightness in `map_brightness_value()`, take a look at Paul McWhorter's video (https://www.youtube.com/watch?v=QxYcYknKbB0) (most of his videos seem pretty great – very informative and well explained). `map_speed_value()` uses a power equation that I derived 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 think this issue would be resolved.

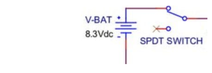
Stage 1C

In this sub-stage, we continue with the circuit completed in the last Stage 1B 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:



ERRATUM NOTE: This schematic does not show the toggle power switch. That was an oversight. The switch should look something like this, as seen in later stages:



Side note: For all stages in this project, for the toggle power switch, I used a SPDT switch but probably should have just used a SPST switch. Checkout this article 1 (https://learn.sparkfun.com/tutorials/switch-basics/all#poles-and-throws-open-and-closed)o learn what "SPDT" and "SPST" mean.

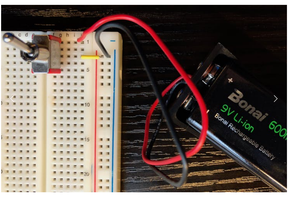
You may notice that the voltage on the positive breadboard rail is unregulated – it comes straight from the battery. Because the only components connected to that node are the switch, button, and potentiometers, this is not concerning. If other components such as the LEDs were also connected to this 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 "9 V" batteries can range in voltage, typically 7.2-9.6 V depending on its type, when fully charged. 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/BH-9VW-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 resistor
- 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 containing from Stage 1A/B.

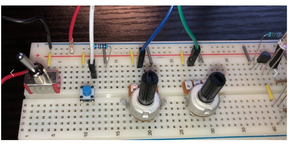


1 / 2

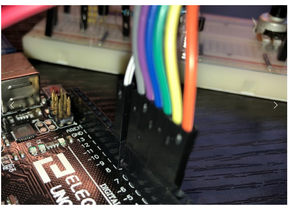
Step 2 : Add the pushbutton. Wire the signal to pin 8 on the Arduino (white jumper in pictures). Put the 10 kΩ resistor between the signal leg of the button and GND.

Step 3 : Add the two potentiometers. The pitch (spacing between pins) of 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 middle/variable/output/wiper pin of the left potentiometer (pattern speed adjustment) to pin A5 on the Arduino (blue jumper in pictures). Wire the middle pin of 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 to OFF as shown in the pictures (open circuit). Wire the positive voltage rail to Vin on the Arduino.



1 / 3



1 / 2

Step 5 : Let's program the Arduino. The code is made up of the three separate files given below. Copy and paste them, or just grab them from the GitHub repository (https://github.com/nabelekt/LED_Multiplexer). The three files must be all together in a folder called LED_multiplexer_arduino. The directory/file structure on your computer should look like this:

LED_multiplexer_arduino
|—— LED_multiplexer.h
|—— LED_multiplexer_arduino.ino
|—— patterns.c

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:

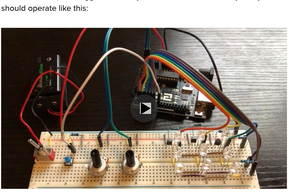
```
// Code partially based on "Transistor Multiplexing 3x3 LEDs" by Marty Peltz
(https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-I-ma
de/)
#include "LED_multiplexer.h"
#include "patterns.c"
#include <math.h>
#define COL_1_CATHODE 2
#define COL_2_CATHODE 3
#define COL_3_CATHODE 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);
```

Open LED_multiplexer_arduino.ino in the Arduino IDE and upload it to your Arduino.

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.

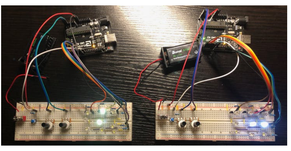
Repeat Stage 1 so that you have a duplicate prototype. You could simply make adjustments to the Stage 1 circuit and not retain the result of Stage 1, but I don't recommend this. I wanted to keep the result of each stage, so I started Stage 2 by repeating almost exactly what was done in Stage 1, and I will assume here that you do the same.

Component List:

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

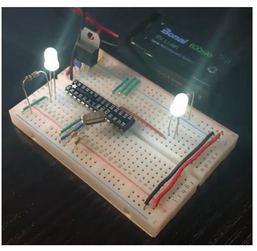
Steps : Follow the steps from Stages 1A/B and 1C, but note that in this second iteration, the toggle switch, pushbutton, and potentiometers should be a bit more crowded together on the left side in order to make room for the microcontroller unit (MCU) that we are going to replace the Arduino with.

You should now have duplicate systems:



Set your Stage 1 circuit aside. For the rest of this stage, we will be modifying the new Stage 2 circuit only.

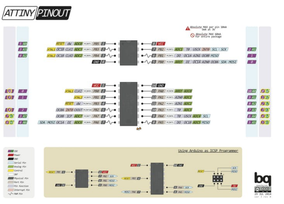
Article A (https://www.arduino.cc/en/Main/Standalone) and Article B (https://www.makeuseof.com/tag/dont-spend-money-on-an-arduino-build-your-own-for-much-less/) were both helpful in figuring out how to go about replacing the Arduino with minimal components (though I later found this (https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard) 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) are 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 a separate programmer like the 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)
- 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).

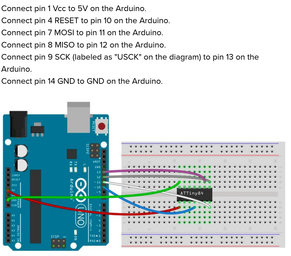
Step 2 : Plug the Arduino into your computer. In the Arduino IDE, select Tools > Board > Arduino/Genuino Uno. Find and select your Arduino under Tools > Port.

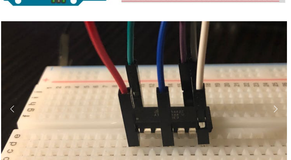
Step 3 : Choose File > Examples > 11.ArduinoISP > ArduinoISP. Upload this sketch to your board. This programs the Arduino to act as an ISP (In-System Programmer) which will allow you to use it to program your MCU.

Step 4 : Place the MCU so that it bridges the gap of the small breadboard (not the same one with your circuit).

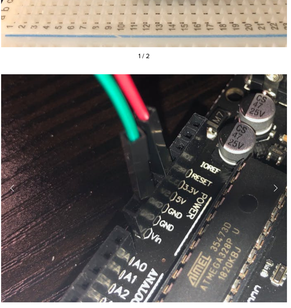
Step 5 : Disconnect the Arduino from your computer and make sure that it is not powered.

Step 6 : Refer to the diagram above and use the physical pin designations shown 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 GND on the Arduino.





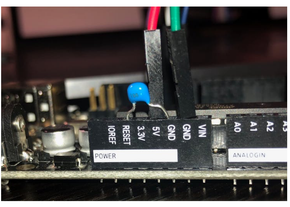
1 / 2



1 / 2

You have now setup the connections necessary to use the SPI protocol for programming.

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 it was necessary for me). My capacitors do not have polarity. If yours does (https://learn.sparkfun.com/tutorials/polarity/all#electrolytic-capacitors), 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 last action is necessary or not).

Step 9 : Go to Tools > Board > Boards 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.

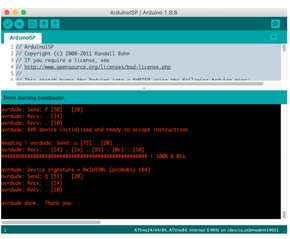
Step 10 : Go to Tools > Board and you should now see a section called "ATtiny Microcontrollers." Select "ATtiny24/44/84". Now, under Tools select "Processor > ATtiny84" and "Clock > Internal 8 MHz."

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 (NOT ArduinoISP).

Step 13 : Open up the IDE preferences and ensure that you have verbose mode enabled ("show verbose output during ✓compilation ✓upload"). Select Tools > Burn Bootloader. If successful you should see something like the following:

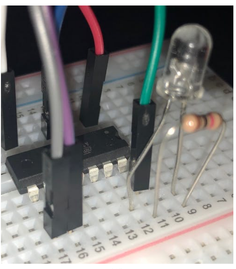


Bootloader successful burned to ATTiny84 MCU

As long as you 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), here (https://www.baldengineer.com/arduino-bootloader.html) and here (https://www.arduino.cc/en/Hacking/Bootloader). If we were using a different programmer, such as the AVRISP mkII (http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42093-AVR-ISP-mkII_UserGuide.pdf) that I mentioned above, we would not need to burn the bootloader (see second and third post here (https://forum.arduino.cc/index.php?topic=522951.0)).

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

Step 15 : Reference the diagram above again, now using the pin designations that I have circled in yellow on the far sides of the graphic. 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 on the breadboard 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.

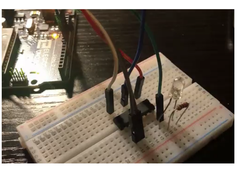


1 / 2

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:

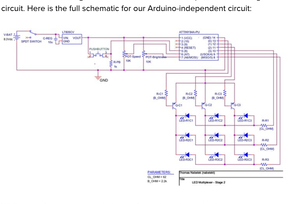


You now know that you can program the ATtiny84. Excellent!

If you have not achieved the desired result, try running through the Troubleshooting List in Step 27 below, but note that some of those items are specific to troubleshooting the programming of the MCU after it has been integrated into our LED multiplexer circuit.

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. Here is the full schematic for our Arduino-independent circuit:

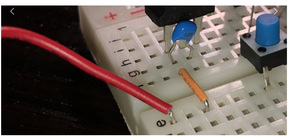


While completing the steps below, refer to the schematic above as it is helpful.
ERRATUM NOTE: This schematic does not show the GND pin on the voltage regulator (L7805CV) wired to common GND, but it should.

Step 17 : Remove the jumper cables that previously went to GND and Vin on the Arduino.

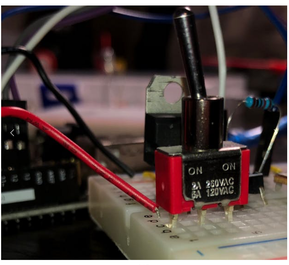
Step 18 : From the left side of the board, remove the toggle switch, the jumper wire between the toggle switch and positive voltage rail, and the positive battery lead.

Step 19 : Place the voltage regulator, 10 µF capacitor, positive battery lead, and three wires as shown.



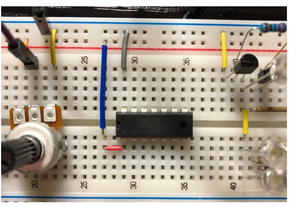
1 / 2

Step 20 : Replace the toggle switch in the new location shown below. Ensure that the switch it is flipped so that the power is OFF (open circuit).

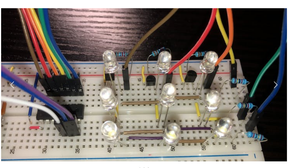


1 / 2

Step 21 : 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.



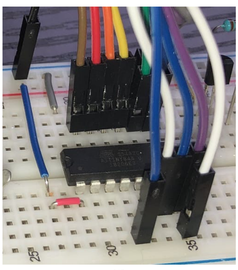
Step 22 : As in Step 6 above, refer to the pinout diagram and use the physical pin designations of the MCU. E.g., pin 1 is in the top left corner of the MCU. We are going to rewire the jumper cables that we unplugged from the Arduino in Step 1. I will use the resistor designations found in the diagram at the beginning of Stage 1A/1B, but it is probably easiest to look at the pictures below.
Connect the jumper from R-R3 to pin 8 (blue jumper).
Connect the jumper from R-R2 to pin 9 (green jumper).
Connect the jumper from R-R1 to pin 10 (yellow jumper).
Connect the jumper from R-C3 to pin 11 (orange jumper).
Connect the jumper from R-C2 to pin 12 (red jumper).
Connect the jumper from R-C1 to pin 13 (brown jumper).
Connect the jumper from the pushbutton to pin 5 (white jumper).
Connect the jumper from the left potentiometer (pattern speed adjustment) to pin 6 (blue jumper).
Connect the jumper from the right potentiometer (brightness adjustment) to pin 7 (p jumper).



1 / 4

We now have all of the connections that we need from/to our MCU to/from our circuit.

Step 23 : Let's set the MCU up to be programmed again by adding more jumpers:
Connect pin 4 RESET to pin 10 on the Arduino (blue jumper).
Connect pin 7 MOSI to pin 11 on the Arduino (white jumper).
Connect pin 8 MISO to pin 12 on the Arduino (gray jumper).
Connect pin 9 SCK (labeled as "USCK" on the diagram) to pin 13 on the Arduino (p jumper).
Pin VCC is already wired to our positive voltage rail on the breadboard, and pin 14 GND is already wired to the ground rail.
Wire the ground rail to pin GND on the Arduino.



1 / 3

Your Arduino should already have the ArduinoISP sketch loaded on to it from earlier, and the MCU already has the bootloader, so we don't need to repeat those steps.

Step 24 : As in Step 5 from Stage 1C, we need to setup our three code files. Either grab the LED_multiplexer_attiny84 folder from the GitHub repository (https://github.com/nabelekt/LED_Multiplexer), or do the following steps a-c:

a) Duplicate the LED_multiplexer_arduino folder that you have (with its contents). Rename the folder to LED_multiplexer_attiny84.

b) Rename the LED_multiplexer_arduino.ino file to LED_multiplexer_attiny84.ino so that you have:

LED_multiplexer_attiny84
|—— LED_multiplexer.h
|—— LED_multiplexer_attiny84.ino
|—— patterns.c

c) Open LED_multiplexer_attiny84.ino, delete all of its contents, and copy and paste the following into it:

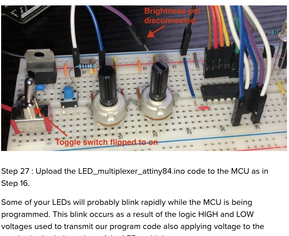
```
// Code partially based on "Transistor Multiplexing 3x3 LEDs" by Marty Peltz (https://www.instructables.com/id/Multiplexing-with-Arduino-Transistors-1-ma de/)
#include "LED_multiplexer.h"
#include "patterns.c"
#include <math.h>
#include <SoftwareSerial.h> // Used for printing to Serial Monitor for debug
#define COL_1_CATHODE 0  // Physical pin 13
#define COL_2_CATHODE 1  // Physical pin 12
#define COL_3_CATHODE 2  // Physical pin 11
#define ROW_1_ANODE 3  // Physical pin 10
#define ROW_2_ANODE 4  // Physical pin 9
#define ROW_3_ANODE 5  // Physical pin 8
#define BUTTON_PIN 8  // Physical pin 5
#define BRIGHTNESS_POT_PIN A6  // Physical pin 7
#define SPEED_POT_PIN A7  // Physical pin 6
#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) =
```

A diff between LED_multiplexer_arduino.ino and LED_multiplexer_attiny84.ino can be found here (http://www.mergely.com/ymRTCRSr/). There, you can look at the files side by side and see what was changed to support the ATtiny84 rather than the Atmega328p on the Arduino. The primary changes were:

- different pin assignments – as mapped out in that pin diagram given towards the beginning of Stage 2
- swapping the `HardwareSerial` object that is already declared for us as `Serial` for a `SoftwareSerial` object that we declare as `Monitor` to allow for printing to the Arduino IDE serial monitor. This is only used for debugging, so I have it commented out.

Step 25 : Temporarily disconnect the jumper leading to the brightness (right) potentiometer. I found that having the potentiometer connected to the MCU at the same pin used for MOSI interferes with programming.

Step 26 : Using the switch on the left, toggle the power to the breadboard circuit ON so that the MCU is powered.

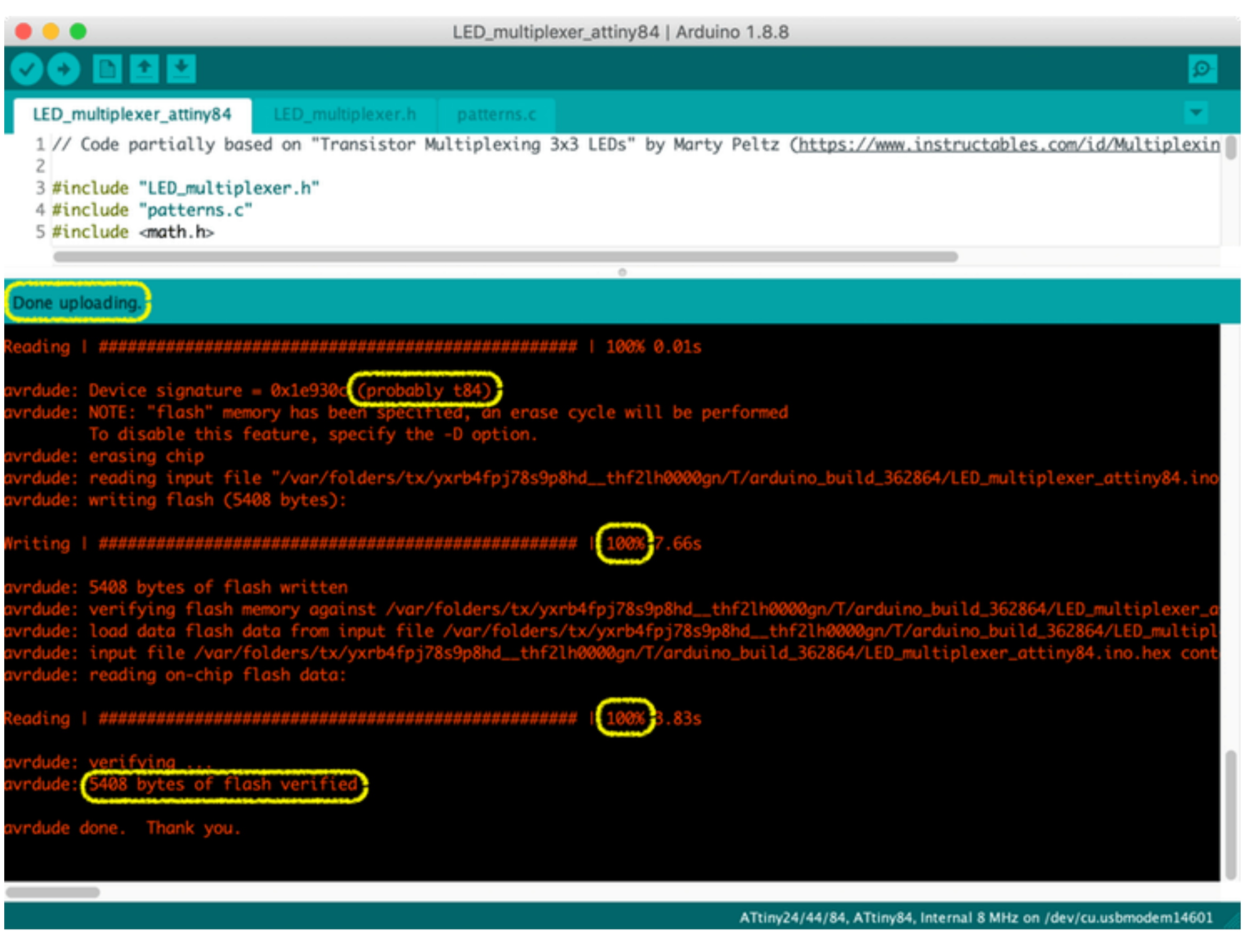


Step 27 : Upload the LED_multiplexer_attiny84.ino code to the MCU as in Step 16.

Some of your LEDs will probably blink rapidly while the MCU is being programmed. This blink occurs as a result of the logic HIGH and LOW voltages used to transmit our program code also applying voltage to the anodes/cathode branches of the LED multiplexer.

Assuming that you have verbose output enabled in the Arduino IDE preferences, make sure that your console looks something like the following.

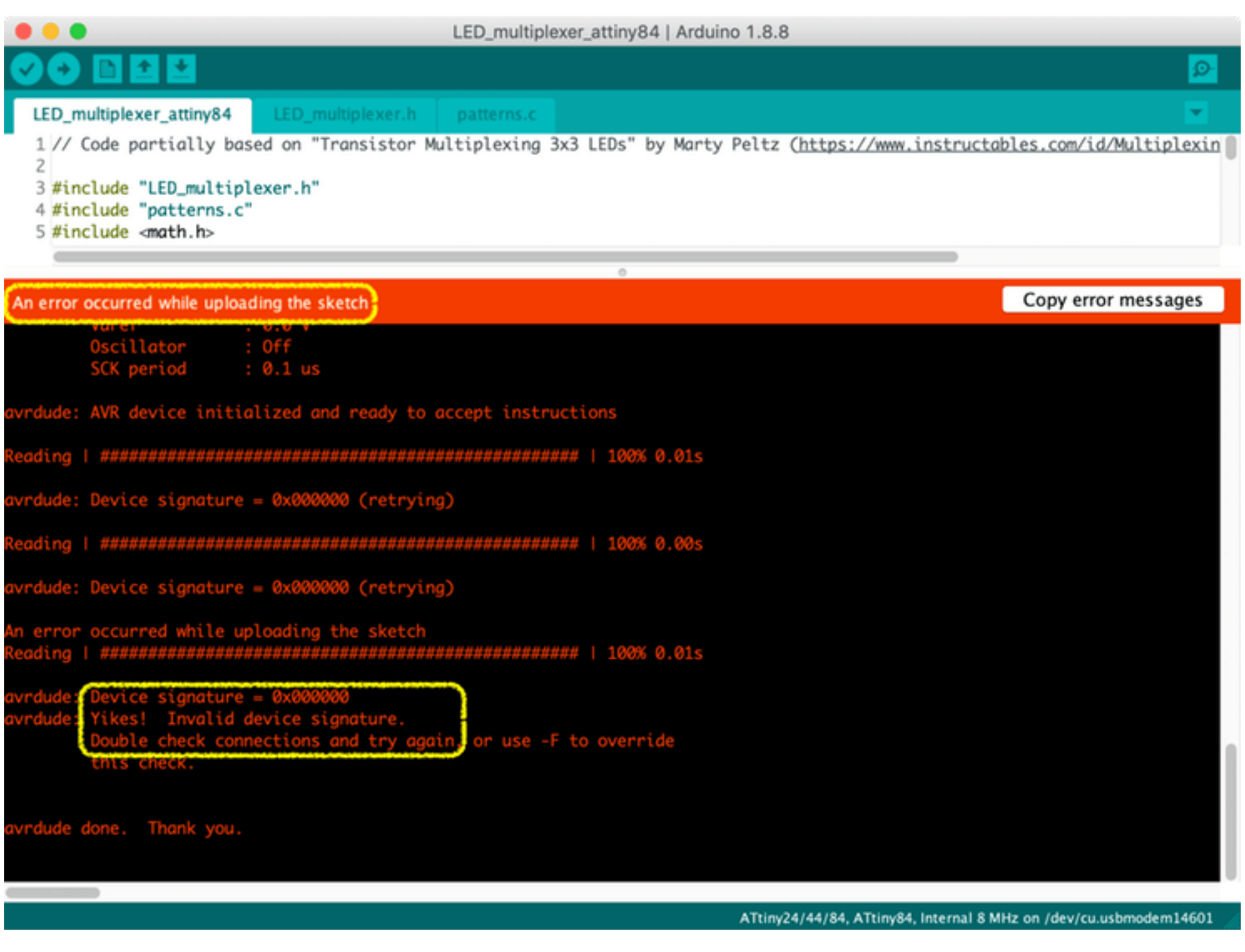
There are a number of things here that will indicate success, I have highlighted some of them:



Successful programming of ATtiny84 MCU

If what you have looks like this screenshot, great! Go to step 28.

If instead you have something that looks like the following, go through the Troubleshooting List below:



Unsuccessful programming of ATtiny84 MCU

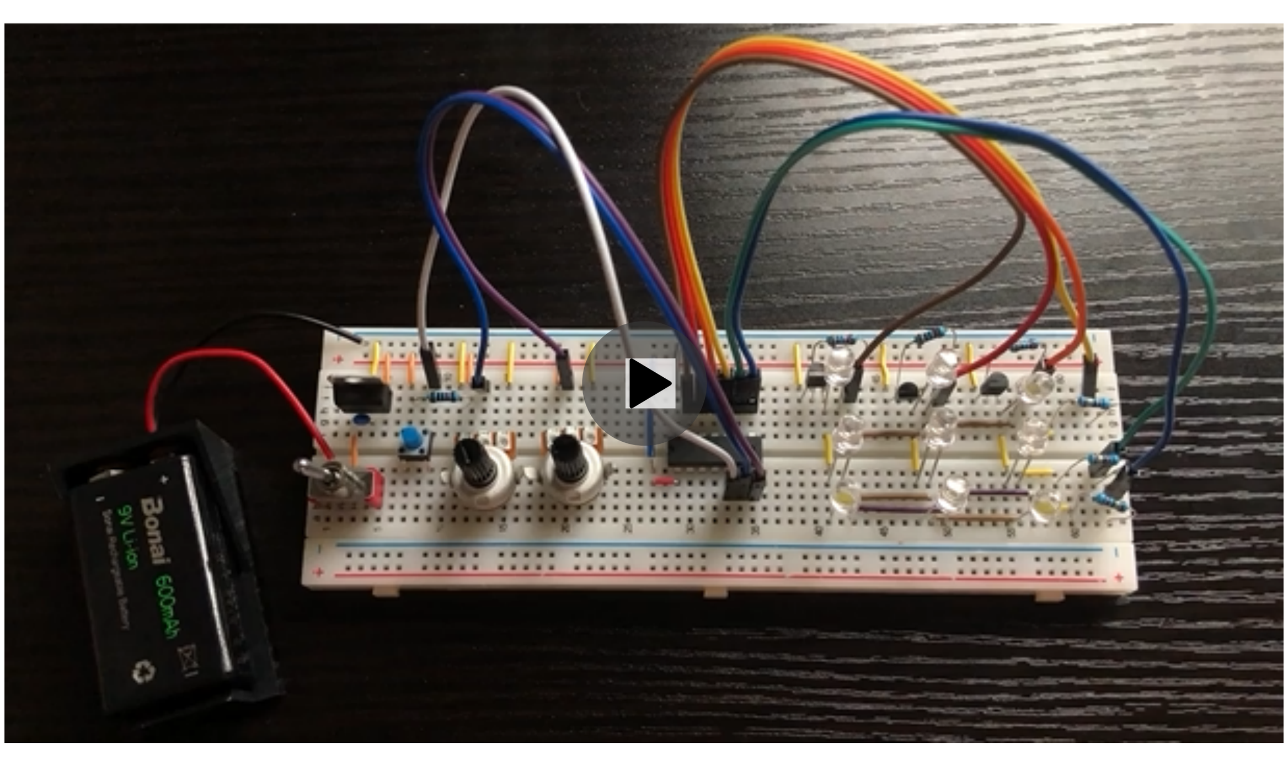
Troubleshooting List

- ATtiny84 MCU and Arduino are wired as shown in Step 23 pictures
- Brightness potentiometer signal jumper is disconnected from MCU
- Ground rail is connected to GND on Arduino
- Toggle switch is flipped to ON position (closed circuit)
- You are using the same MCU used earlier with the bootloader already on it
- The Arduino has the ArduinoISP sketch successfully uploaded to it
- The 10 µF capacitor between RESET and GND is in place, as described in Step 7
- In the Arduino IDE, you have selected "ATtiny24/44/84", "ATtiny84", and "Internal 8 MHz" for Board, Processor, and Clock, respectively
- You have "Arduino as ISP" selected under Tools > Programmer and NOT "ArduinoISP," "ArduinoISP.org," or "Arduino as ISP (ATmega32U4)"
- Arduino is plugged in to computer and selected under Tools > Port

Step 28 : Toggle the power back to OFF. Reconnect the brightness potentiometer jumper wire to its previous position.

Step 29 : Unplug all jumper wires leading to the Arduino from your breadboard.

Step 30 : Toggle the power switch on the left side of the board to ON! Hopefully you see something like this:



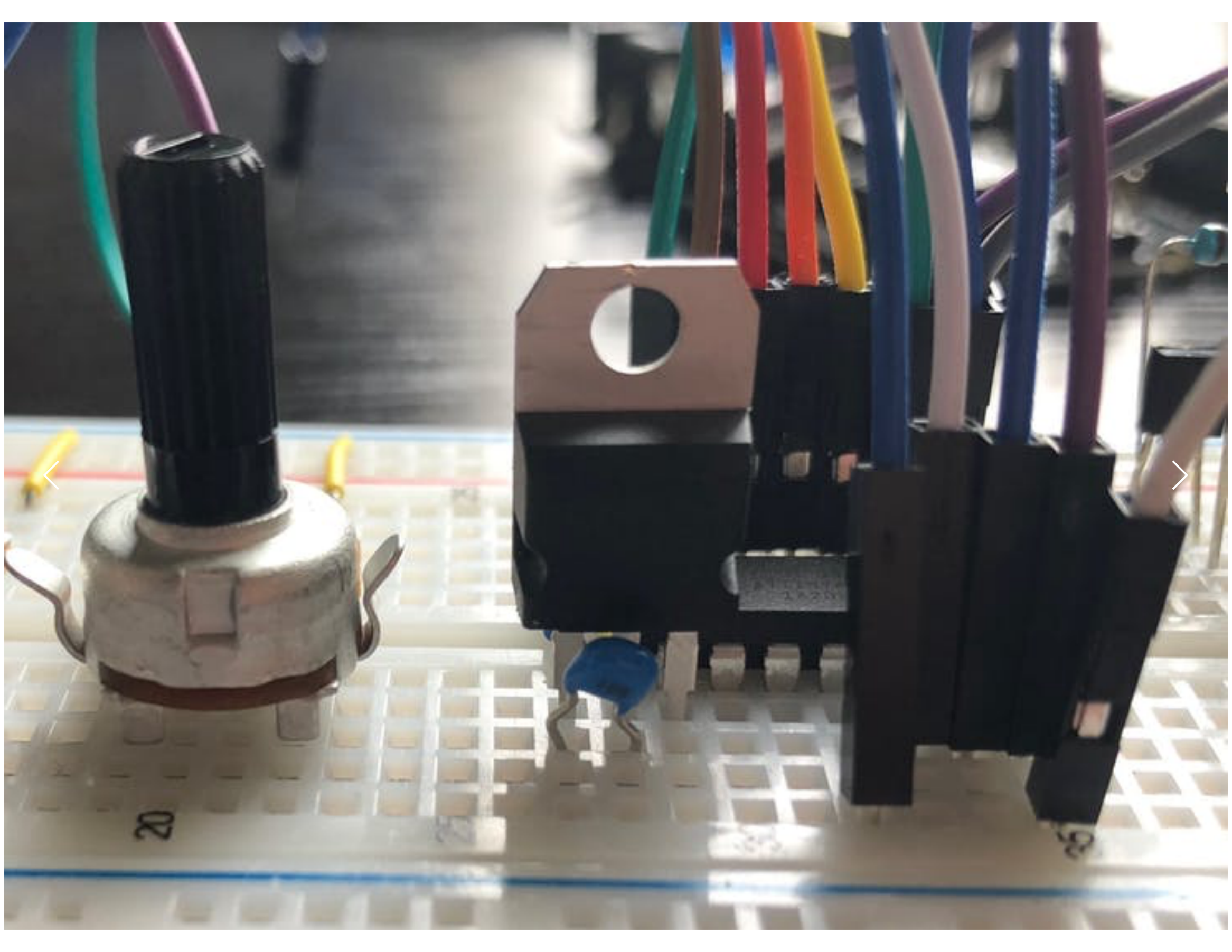
If you do, great! You now have an Arduino-intependent protype circuit. This step is a big one in the Arduino prototype to manufacturable product process. You are done with Stage 2!

Side note: Notice that when you flip the power switch on the Stage 2 prototype, the LED pattern beings immediately. However, when you flip the switch on the Stage 1 prototype, there is an almost two second delay before the LED pattern begins. I believe that this has to do with how the bootloader works.

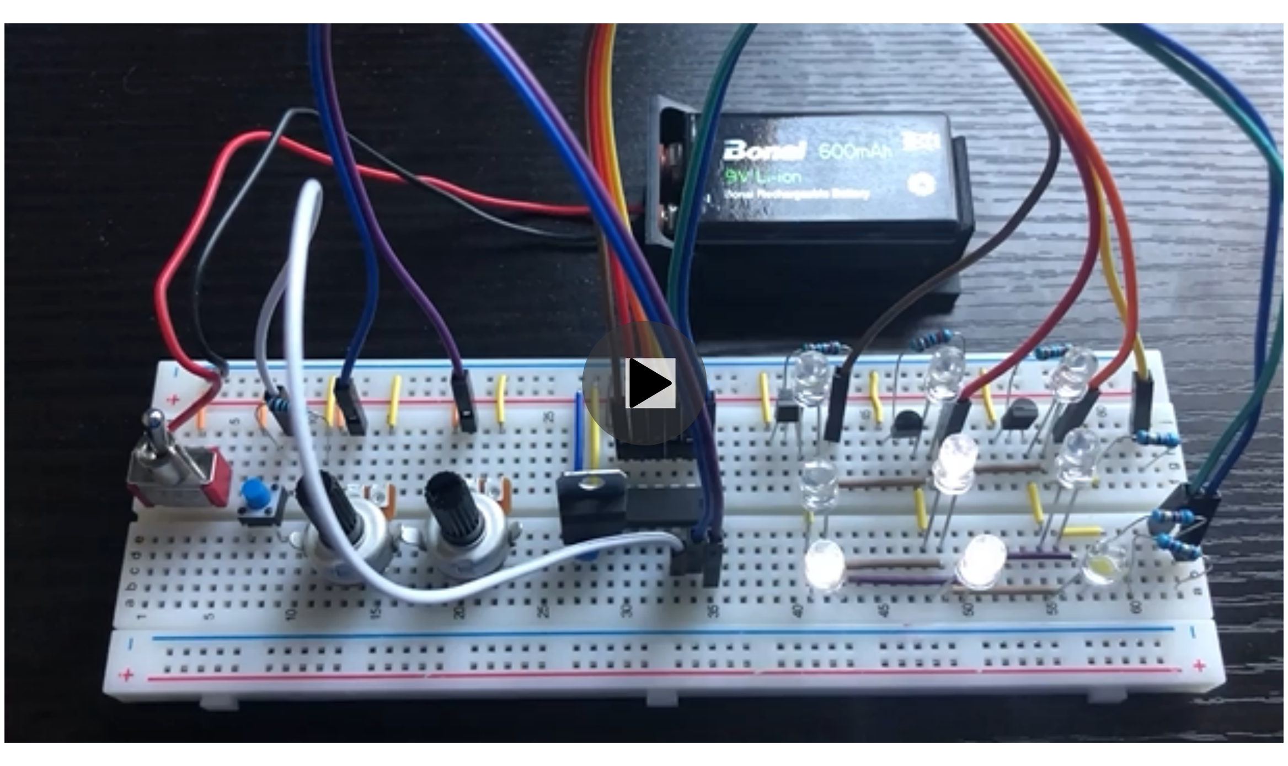
If you do not have a working protype like that shown in the video, but the MCU was successfully programmed as shown in Step 27, and you think that printing messages from the MCU to the Arduino IDE Serial Monitor might be helpful, see the Debug Note below.

Debug Note

Initially, I made the mistake of regulating the voltage only for the MCU:


1 / 2 • Previous setup regulating only the voltage to the MCU

This caused the input voltage on every button press to be unregulated ~8 V at the button pin on the MCU. This too-high voltage would reset the MCU so that I got the following behavior:



Problem with previous setup regulating only the voltage to the MCU

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.

To do that debugging, it was helpful to be able to print things out to the Arduino IDE Serial Monitor despite the MCU not having a serial port. This is where the SoftwareSerial Monitor came from in the LED_multiplexer_attiny84.ino code. I am not going to go into detail here about how to set that up, but if you need to do the same, checkout Tom Donnelly's helpful video (https://www.youtube.com/watch?v=9CX4i6rMXS8). Note that the jumper wires that he connects are just those wired in Step 23 above, in addition to the RX/TX jumper(s). If you get stuck on this, feel free to ask for help in the comments at the end.

Stage 3

Moving on to Stage 3, we want to move on from our solderless breadboard to something a bit more condensed and permanent. Optionally, this stage could certainly be skipped in the prototyping process if you as a developer/engineer wanted to move straight to designing a PCB (what we'll do in Stage 4) based off of the a Stage 2-equivalent circuit. In this stage we will use stripboard (https://en.wikipedia.org/wiki/Stripboard) and solder to build our prototype. Completing this process requires a lot of materials and tools:

Component List:

- 22 AWG wire like this (https://www.amazon.com/gp/product/B01180QKJ0) to solder to the board (I often prefer solid core wire over stranded wire, but that is up to your preference)
- wire to program the MCU (jumper wires like these, (https://www.digikey.com/product-detail/en/PRT-12795/1568-1512-ND)and these o (https://www.amazon.com/gp/product/B079FKJ4S3)r 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) - whatever you have used in the previous stages
- A smaller solderless breadboard, (https://www.digikey.com/product-detail/en/dfrobot/FIT0096/1738-1326-ND/7597069) or just use the one from Stage 2
- Stripboard like this (https://www.amazon.com/gp/product/B00C9NXP94) – be sure that it it is at least 32 holes wide and 17 holes high so that you can fit your circuit (see images below)
- 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 (bipolar junction transistors) (I used these (https://www.amazon.com/gp/product/B06XRBLKDR))
- ATtiny84 MCU (https://www.digikey.com/product-detail/en/microchip-technology/ATTINY84A-PU/ATTINY84A-PU-ND/2774082)
- 10 kΩ resistor
- 6x header pins (https://www.digikey.com/product-detail/en/adam-tech/PH1-02-UA/2057-PH1-02-UA-ND/9830266) - you need 6 total: 2x1, 2x1, 1x1, and 1x1; see images below. The 2x1 pins can be broken/cut apart to have 2 1x1 pins, but it may just be easier to buy separate 1x1 pins.
- 1x 2-position jumper shunt/connector (https://www.digikey.com/product-detail/en/SPC02SYAN/S9001-ND/76375)
- 9x Cree C512A-WNN-CZ0B0152 LEDs (https://www.digikey.com/product-detail/en/cree-inc/C512A-WNN-CZ0B0152/C512A-WNN-CZ0B0152CT-ND/6138557)

,

- 2x 10 µF capacitor like this (https://www.digikey.com/product-detail/en/tdk-corporation/FG24X7R1A106KRT00/445-181259-ND/) - 1 of these is to use in the programming process. You can use the same one used in Stage 2.
- 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)
- SPDT Toggle switch - this time I used this one (https://www.digikey.com/product-detail/en/nidec-copal-electronics/ATE1D-2M3-10-Z/563-1157-ND/1792018) because I don't think I could get the one used in the previous stages through the stripboard holes.
- Pushbutton (https://www.digikey.com/product-detail/en/te-connectivity-alcoswitch-switches/1-1825910-0/450-1652-ND/1632538)
- 2x 10 kΩ potentiometer (https://www.digikey.com/product-detail/en/tt-electronics-bi/P120PK-Y25BR10K/987-1710-ND/5957454)
- L7805C 5V Linear Voltage Regulator (https://www.digikey.com/product-detail/en/stmicroelectronics/L7805CV/497-1443-5-ND/)
- 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)
- Copper tape (https://www.digikey.com/product-detail/en/adafruit-industries-llc/3483/1528-2748-ND/9745247)

You will be soldering in this stage. You should either already know what materials and tools you prefer to use for soldering, or you should take advantage of the abundant resources on the interwebs giving an introduction to soldering. Here are a few videos you could start with:

- Soldering Tutorial for Beginners: Five Easy Steps (https://www.youtube.com/watch?v=Qps9woUGkvI)
- SparkFun Advanced PTH Soldering (https://www.youtube.com/watch?v=t9LOtOBOTb0)
- How to Solder properly ‖ Through-hole (THT) & Surface-mount (SMD) (https://www.youtube.com/watch?v=VxMV6wGS3NY)

You will at least need:

- solder
- soldering iron and stand

you may also want:

- flux (https://www.digikey.com/product-detail/en/chip-quik-inc/CQ4LF/CQ4LF-ND/6227077)
- solder wick (https://www.digikey.com/product-detail/en/aven-tools/17542/243-1186-ND/2815651)

You should already have:

- 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) – use what you used to program the MCU in Stage 2

Other recommended tools for this stage are:

- Drill with 3/8" drill bit, maybe start smaller
- X-Acto knife or similar
- A circuit board holder like this one (https://www.amazon.com/gp/product/B00Q2TTQEE)
- A wire cutting tool like this one (https://www.amazon.com/Hakko-CHP-170-Stand-off-Maximum-Capacity/dp/B076M3ZHBV)
- Plastic cutter like this one (https://www.amazon.com/Hyde-Tools-45730-Knife/dp/B000C027ZE/)
- A cutting mat like this one (https://www.amazon.com/gp/product/B0025II5P4)

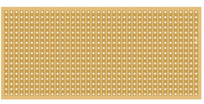
The same circuit diagram as that introduced in Stage 2 Step 16 also applies to our Stage 3 circuit.

We are going to program the MCU (still an ATtiny84) in the same way that we did in Steps 23-30 of Stage 2. I suggest swapping the unprogrammed MCU that you have for this stage with the MCU already programmed and integrated with the Stage 2 circuit. After making this swap, you can follow Steps 23-30 of Stage 2 to program the new MCU (you probably don't have to repeat Step 24) and ensure that it is programmed correctly with an already-proven circuit. Our Stage 3 circuit will provide MCU-programming capability, but it may be easier to address programming issues now than when everything is soldered in place.

Step 1 : Program the MCU as described in the previous paragraph.

Now it's time for us to prepare our stripboard. We need to get the circuit node layout on the stripboard – which currently looks like what is shown in the first image below – to look like what is shown in the second image so that we can solder our components and wires as shown in the third image.

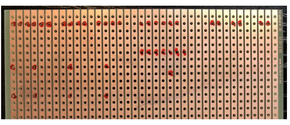
Note that these images are somewhat misleading because they show the copper strips on the same side of the board as the components. We will achieve our desired node configuration by breaking the conductive stripboard paths where necessary and using copper tape to create a GND rail/node across the top.



1 / 3

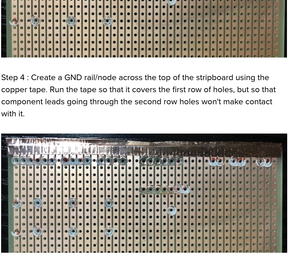
Because the images show the components on the same side of the board as the copper strips, we need to reflect the node layout shown, with the mirror line/axis of reflection running vertically down the center of the board.

Step 2 : Mark the board with a permanent marker as shown in the image below. This gives us our reflected node layout and will tell us where we need to break the conductive copper strips. Refer to the second image given above, but remember that you must reflect the layout.



Step 3 : Break the conductive paths where you marked them in the previous step. There are different thoughts on how to go about doing this and tools that people use. (https://electronics.stackexchange.com/questions/94659/how-to-cut-the-tracks-of-a-stripboard) I used a drill. I believe I started with a 1/4" drill bit and moved up to a 3/8" drill bit. You should start smaller as well, and go slow. You don't want to drill any further into the board than it takes to break the conductive path.

You should use your multimeter/ohmmeter to ensure that there is no conductivity (https://www.ifixit.com/Guide/How+To+Use+A+Multimeter/25632#s64987)across the path/strip breaks.

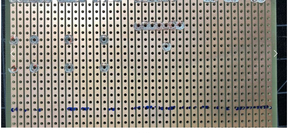


Step 4 : Create a GND rail/node across the top of the stripboard using the copper tape. Run the tape so that it covers the first row of holes, but so that component leads going through the second row holes won't make contact with it.

Flip the board over, place it on a cutting mat, and use an X-Acto knife or similar to trim off the excess copper tape.

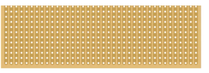
Step 5 : Cut the excess stripboard off to minimize the board's size. We need 16 rows of holes excluding those that we covered with the copper tape. Starting with the row of holes that we covered, count to find the 18th row and mark it with a permanent marker.

Use a tool like the plastic cutter listed above to score the board where you have marked it enough so that you can break the board without cracking it where you don't want to – I think I went almost all the way through the board with the cutter. This can be dangerous – be sure to score so that if the cutter slips, it will not make contact with your body.



1 / 2

Now that our stripboard is prepped, we'll solder our components to the board in these phases:

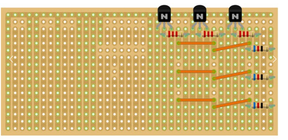


Hopefully this is obvious to you, but be sure to place the components on the side without the copper strips. The above animation and the graphics below are misleading in that they show the components and copper strips on the same side of the board.

Make use of a circuit board holder like I listed above to make this process much easier. I used lightly-placed scotch tape to hold the components in place so that I could flip the board over and solder away.

Step 6 : Solder the transistors, 62 Ω resistors, 2.2 kΩ resistors, and wires that will connect the LED anodes in place as shown below.

Because our copper tape is reliably conductive on only one side, in this step I also put solder to ensure good connections between the strip board strips and the copper tape.



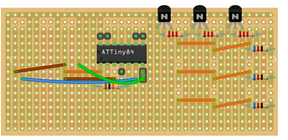
1 / 3

Step 7 : Add the MCU, header pins, 10 kΩ resistor, and wires shown.

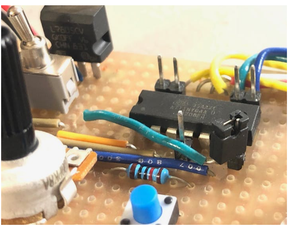
ERRATUM NOTE: In the graphics shown, and in most of the photos below, the green wire is wired to a strip/node with nothing else. This green wire must be soldered one strip to the left – to the same strip/node containing the variable/output/wiper/middle lead of the right potentiometer (brightness adjustment).

This same wire should not be soldered on its MCU end. Rather, we will connect the potentiometer terminal to the MCU by using a shunt to bridge a header pin at Pin 7 on the MCU to the wire.
ERRATUM NOTE: The shunt is missing in some of the photos below.) Also, the header pin is missing from the second and third images in the series directly below. In the first image, the header pin is covered by the shunt. The photo below the series of three shows what you should end up with.

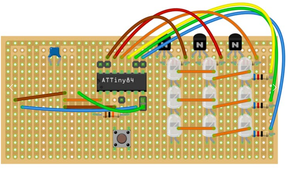
The whole purpose of this shunt business is the same as that of Step 25 in Stage 2 – having the potentiomeer connected to the MCU at the same pin used for MOSI interferes with programming. When we want to program the MCU again, we must be able to temporarily disconnect the potentiometer from that pin. Using the shunt instead of soldering the connection allows us to do that.



1 / 3

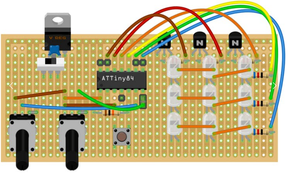


Step 8 : Solder into place the capacitor, pushbutton, and LED multiplexer control signal wires.



1 / 3

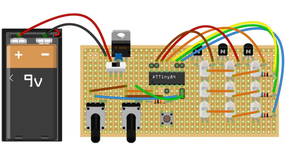
Step 9 : Add the voltage regulator, toggle switch, and potentiometers. Note that the leads of the potentiometers must be bent and placed in the stripboard like they were placed in the breadboards in Stages 1 and 2. Don't let the first image below mislead you – the leads should span four strips with the Vcc lead through the first, the variable/output/wiper lead through the second, and the GND lead through the *fourth*, with nothing through the third.



1 / 3

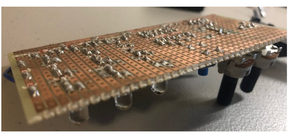
In this layout, the strip with the output pin of the voltage regulator functions as a
5 V rail.

Step 10 : Finally, connect the battery terminals.

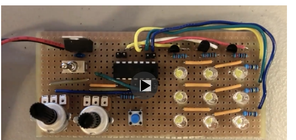


1 / 2

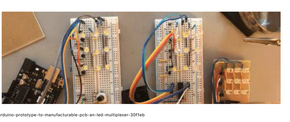
Step 11 : Using something like the wire cutting tool listed above, trim the component and wire leads off the underside of the board to clean it up a bit and help it to sit a bit flatter. I suggest disconnecting the battery for this so that you don't short any components, and wearing safety glasses is a great idea here as well – the pieces that you cut off are likely to want to fly.



Reconnect the battery and there you have it! Because we programmed the MCU previously, we don't need to do so now. You should now have a fully functioning Stage 3 circuit:



We want our LED Multiplexer circuit to be reprogrammable, so we built in the ability to write new code to the MCU by using the header pins to connect to the Arduino. If you need to do that, just follow Steps 23-29 of Stage 2, but this time making use of the header pins.



Stage 4

Stage 4 introduces our first printed circuit board (PCB) version of the circuit. If you are new to PCB design, know that there is a learning curve to it and that I will not give the details of how to go about it here. It is possible to complete the Stage 4 and 5 circuits without understanding PCB design by using the files that I have provided, but if you do that, you will not learn and be able to use the skill. To learn PCB design, I encourage you to look at other resources online including courses like this quick one on Udacity (https://www.udemy.com/course/pcb-design-a-tiny-arduino-in-altium-circuitmaker/) (usually costs $9.99, you just have to be sure to catch it durning one of the frequent "sale" periods) and the free tutorials in this YouTube playlist (https://www.youtube.com/watch?v=x79IiHZZBV0&list=PL_NjiOJOuXg3dDFFe9jmv0mgmol3o5pBb). I used the same Altium CircuitMaker software that those two resources show, and my files are available to you.

Before moving to the Stage 5 PCB which will focus on using surface-mount device/technology (SMD/SMT) (https://en.wikipedia.org/wiki/Surface-mount_technology) components, we will develop a PCB using all through hole (TH) components. We will assemble by hand as you might do for early prototypes of a device. Assembling a PCB by hand is generally much simpler with TH components than with SMD components.

Here is my basic process for completing Stage 4:

Step 1 : Recreate schematic with appropriate components.

In Stages 1-3, we never particularly cared about the footprint of our components. We needed the pins of some of the components – like the voltage regulator, switch, potentiometer, MCU, etc. – to have a pitch with a multiple of 0.1" so that they would fit into the breadboard while other components – like the resistors, transistors, LEDs, etc. – had long leads that we could bend as necessary. Otherwise, we were able to arrange the board so that everything would fit. With PCB circuits, placement of the vias (https://en.wikipedia.org/wiki/Via_(electronics)) needs to be precise so that our components can be placed and soldered to ensure conductivity (by traces (https://en.wikipedia.org/wiki/Signal_trace)) where appropriate. To do this, we must ensure that the components used in our schematic have footprints attached to them to enable our PCB design. Parts/components to be used in our EDA design software (https://en.wikipedia.org/wiki/Electronic_design_automation) can be downloaded from a variety of sources. Some of my favorites are:

- SnapEDA (https://www.snapeda.com)
- Octopart (https://octopart.com)
- UltraLibarian (https://www.ultralibrarian.com)

though there are many others. Sometimes you can download components from the manufacturer. When you download a component, it should come with three pieces:

- Symbol
- Footprint
- 3D Model
- (and occasionally a SPICE simulation model)

though the 3D model will often not be included. This is only an issue if you want to show off the 3D model of your PCB or have mechanical concerns. Checkout the part files that can be downloaded for our MCU – the ATTINY84A-PU – here (https://app.ultralibrarian.com/details/8915b507-109b-11e9-ab3a-0a3560a4cccc/Microchip/ATTINY84A-PU?uid=e3282e80494a3cef). Of course basically every different EDA software suite uses different file types for parts with limited compatibility between them, so be sure to keep that in mind.

It is now important to be specific with manufacturer part numbers. One component – e.g. the ATtiny84 – can have many different variants and configurations. Searching "attiny84" on Digi-Key gets you many different results (https://www.digikey.com/products/en/integrated-circuits-ics/embedded-microcontrollers/685?k=attiny84), but they are all ATtiny84s. I encourage you to go and look at those and to think about the different uses cases for each. Additionally, components with similar or even matching specifications, such as 62 Ω SMD resistors, can come in all sorts of different packages (https://www.digikey.com/short/p15hvw) (take a look at the different values in the (Package/Case) field.

Now, to begin: Following along will be much easier if you use Altium CircuitMaker as I did. CircuitMaker is essential if you want to make use of my shared project files. So, the first step, is probably to get that installed and running.

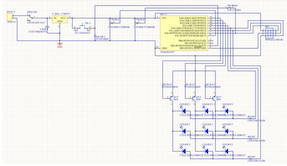
CircuitMaker does let one import a .DSN file from OrCAD PSpice Designer Lite – the software that I used in Stage 1 to create the circuit schematic and simulate the circuit. However, when I put together the circuit in OrCAD, I didn't pay attention to make sure that I was using component parts that have PCB footprints associated with them and that are readily available from sources like DigiKey. So, I decided to recreate the circuit from scratch in CircuitMaker, this time choosing specific SMD (surface-mount device) (https://en.wikipedia.org/wiki/Surface-mount_technology) components that were available from DigiKey in low quantities; that had specifications that most closely matched the through-hole components I used in Stages 1, 2, and 3; and that had PCB footprints available.

In CircuitMaker, one cannot import parts downloaded from any site. Altium designed the software so that it uses parts from the Octopart database only. The good thing is that you can edit or create symbols and footprints, and attach 3D STEP models, for any of those parts. I did this for several of the parts I used, so if you go through and build the circuit in CircuitMaker yourself, and use the same parts, you will probably see my name attached to various revisions.

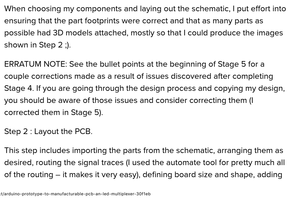
After getting the circuit re-designed in CircuitMaker, I found that I was able to use most of the same components that I used in Stage 3 and previous stages. Below is the BOM (https://en.wikipedia.org/wiki/Bill_of_materials) that I ended up with, and here (https://www.digikey.com/short/pvh4wp) is a Digi-Key cart containing all of these parts (minus the 9 V battery)! (If you see "an improved value is available for one or more of your selections. Press Submit to continue with no changes," click "Submit.")

- battery connection terminal "BATT-T": TE Connectivity 282834-2 (https://www.digikey.com/product-detail/en/te-connectivity-amp-connectors/282834-2/A98333-ND/1150135) (x1)
- toggle power switch "SPDT-SW": Nidec Copal Electronics 563-1157-ND (https://www.digikey.com/product-detail/en/nidec-copal-electronics/ATE1D-2M3-10-Z/563-1157-ND/7792018) (x1)
- 10 µF capacitor "C10uF-1": KEMET C322C106K3R5TA (https://www.digikey.com/product-detail/en/kemet/C322C106K3R5TA/399-13968-ND/6562379) (x1)
- voltage regulator "V_REG": STMicroelectronics L7805CV (https://www.digikey.com/product-detail/en/stmicroelectronics/L7805CV/497-1443-5-ND/585964) (x1)
- pushbutton "SW-1": TE Connectivity 1-1825910-0 (https://www.digikey.com/short/pvh4bn) (x1)
- 1 kΩ resistor "CF14JT1K00": Stackpole Electronics Inc CF14JT1K00 (https://www.digikey.com/product-detail/en/stackpole-electronics-inc/CF14JT1K00/CF14JT1K00CT-ND/1830350) (x1)
- 10 kΩ potentiometers "VR10k-S" and "VR10k-B": TT Electronics/BI P120PK-Y25BR10K (https://www.digikey.com/short/pvh4bm) (x2)
- MCU "MCU-1": Microchip Technology ATtiny84A-PU (https://www.digikey.com/product-detail/en/ATTINY84A-PU/ATTINY84A-PU-ND/2774082) (x1)
- SPST switch to disconnect VR10k-B when programming "SW-PROG": Würth Elektronik 418117270901 (https://www.digikey.com/short/pvh4b3) (x1)
- SPI programming header "SPI": Sullins PPPC032LFBN-RC (https://www.digikey.com/short/pvh491) (x1)
- 2.2 kΩ resistors "R2.2k-CX": Stackpole Electronics CFM12JT2K20 (https://www.digikey.com/product-detail/en/stackpole-electronics-inc/CFM12JT2K20/S2.2KHCT-ND/2617587) (x3)
- 62 Ω resistors "R62-RX": Yageo CFR-25JB-52-62R (https://www.digikey.com/short/pvh494) (x3)
- NPN BJTs "Q-CX": ON Semiconductor 2N3904BU (https://www.digikey.com/short/pvh459) (x3)
- LEDs "LED-RXCX": Cree C512A-WNS-CZ0B0152 (https://www.digikey.com/short/pvh4r8) (x9)
- 9 V battery (not part of PCB design but important!)
- 9 V connector (https://www.digikey.com/product-detail/en/mpd-memory-protection-devices/BH9VW/BH9VW-ND/221547) like that used in previous stages (also not part of PCB design)

You can access my CircuitMaker project (https://circuitmaker.com/Projects/Details/nabelekt/Adjustable-and-Programmable-ATtiny-Based-LED-Multiplexer-TH) for this Stage 4 board by launching CircuitMaker and typing my name "Thomas Nabelek" in the Projects search box:



Hover over the truncated names of the different LED Multiplexer projects so that you find the one with "(TH)" at the end. The "(SMD)" version is the Stage 5 circuit. Once you open the project, open the schematic and you should see a familiar design:



Stage 4 CircuitMaker Schematic

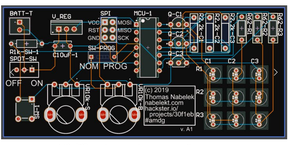
When choosing my components and laying out the schematic, I put effort into ensuring that the part footprints were correct and that as many parts as possible had 3D models attached, mostly so that I could produce the images shown in Step 2 :).

ERRATUM NOTE: See the bullet points at the beginning of Stage 5 for a couple corrections made as a result of issues discovered after completing Stage 4. If you are going through the design process and copying my design, you should be aware of those issues and consider correcting them (I corrected them in Stage 5).

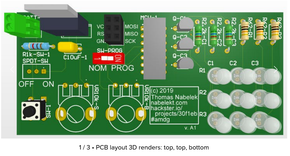
Step 2 : Layout the PCB.

This step includes importing the parts from the schematic, arranging them as desired, routing the signal traces (I used the automate tool for pretty much all of the routing – it makes it very easy), defining board size and shape, adding

text to the silkscreen, and making sure that your manufacturer's fabrication rules (like these (https://www.pcbway.com/pcb_prototype/PCB_Manufacturing_tolerances.html)) are adhered to. Again, I am not going to go into the details of doing all of that here and encourage you to look at the resources I listed under Step 1 to learn about doing all of this. Here is what my completed layout looked like:



1 / 3 • PCB layout: all layers, top layer only, bottom layer only



1 / 3 • PCB layout 3D renders: top, top, bottom

Step 3 : Export Gerber (https://en.wikipedia.org/wiki/Gerber_format) and NC drill (https://en.wikipedia.org/wiki/PCB_NC_formats) files.

Make sure to be aware of your manufacturer's file requirements when submitting a fabrication order. Here (https://www.pcbway.com/blog/PCB_Layout_Software/How_to_generate_Gerber_files_from_CircuitMaker.html) are PCBWay's instructions specific to CircuitMaker.

Step 4 : Order fabricated boards from your manufacturer and components from your supplier.

The components for the Stage 4 circuit can be ordered using the DigiKey cart that I put together and linked above in Step 1.

When I ordered by boards from PCBWay, I got five boards (quantity five was the minimum) for just $5 plus shipping from China. With shipping, the total fabrication cost was $19. Here are the details from my order:

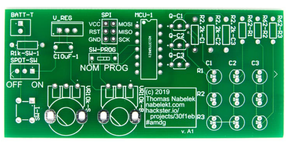
Product Detail

Product No.:	W0107764A21	Gerber File :	PCB_Combined_Gerber_and_Drill.zip		
Board type :	Single PCB	Panel Way :			
Size :	43 x 93 mm	Quantity :	5	Layers :	2 Layers
Material :	FR-4 TG130	Thickness :	1.6 mm	Min Track/Spacing :	6/6mil
Min Hole Size :	0.3	Solder Mask :	Green	Silkscreen :	White
Gold fingers :	No	Surface Finish :	HASL with lead	Via Process :	Tenting vias
Finished Copper :	1 oz Cu	Additional Options :	UL Marking:None,		
Create Time :	2019/10/4 14:26:16	Build Time :	24hours	Estimated Finish Time :	2019-10-06 China Time Zone(GMT+8)
Manufacturing :					

Total: **US $5**

PCB fabrication order details

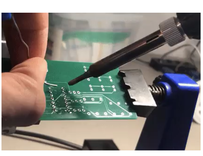
I like PCBWay because they look over and review your design for manufacturing issues and communicate with you to get them resolved. See a full PCBWay review My order was shipped one day after I submitted it and I received it four days after that. Not too shabby. Here is what my boards looked like when I received them:



1 / 4

Step 5 : Assemble the PCB.

As in Stage 4, you will need to get our your solder, soldering iron, safety glasses, and other materials and get to work.



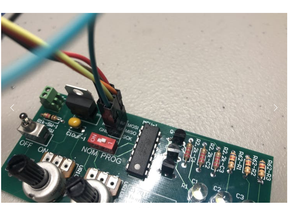
Once you're done soldering, attach the battery holder.

1 / 3

Step 6 : Program the MCU, now much easier with our SPI header!

You know the drill! Same MCU SPI pins (from the header on the board) to same Arduino pins.

VCC -> 5V
RST -> pin 10
GND -> GND
SCK -> pin 13
MISO -> pin 12
MOSI -> pin 11



1 / 4

Don't forget the 10 µF capacitor on the Arduino and make sure that the board's dip switch is set to PROG, not NORM.

It is important to note here that in the Stage 4 design, the labels for ON and OFF are flipped at the toggle power switch. Or rather, with the orientation of our switch (I oriented it saw that the manufacturer's printed text would not face the front of the board), pin 3 should have been connected to the node that pin 1 is connected to, and pin 1 to nothing as pin 3 is. I corrected this in Stage 5. What this issue means for Stage 4 is that we need to have the switch flipped to "ON" when we want the circuit to be off and vice versa. Should have just used a SPST switch!

Once you've got the MCU programmed (and verified by the Arduino IDE), disconnect the SPI wires, flip the programming switch back to NOM, connect the 9 V battery to the battery holder, and toggle the power switch if necessary.

Wahla! Hopefully your circuit works just the same as those in the previous stages:

Stage 5

Stage 5 involves transitioning from a PCB using primarily TH components to a PCB using primarily SMD components. Stage 5 is split up into Stage 5a and Stage 5b. In 5a, we will assemble our redesigned PCB ourselves. In 5b, the assembly will be down by the PCB manufacturer. Steps 1, 2, and 4 are shared by 5a and 5b while Step 3 (self-assembly) is unique to 5a. That is to say: the design and board fabrication are shared by 5a and 5b, the self-assembly is unique to 5a, and the SPI programming will be the same for 5a and 5b.

Step 1 : Modify schematic to use SMD components where appropriate and update PCB layout.

The biggest hurdle of moving from a TH-component-based circuit to a SMD-based circuit is finding SMD components for which the specs most closely match the previously chosen TH components. Note that any components we normally interact with while using our prototype device – e.g. power switch, potentiometers, SPI header, and button – will remain TH components. You may find this Reddit post (https://www.reddit.com/r/PrintedCircuitBoard/comments/d9v8uq/integrating_panel_mount_components_into_a_pcb/)helpful in understanding this choice and in learning a bit about panel mount components and considerations that should be made for mechanical stresses and housings. Here is one comment that stood out to me, from datenwolf: "As a general rule of thumb: *Any* component that comes into mechanical contact with a [human] (switches, connectors) should always be done THT or edge launched."

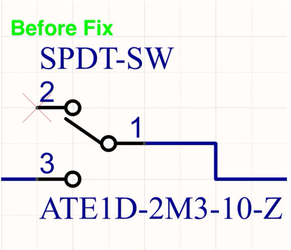
As in Stage 4, you can find my CircuitMaker project (https://circuitmaker.com/Projects/Details/nabelekt/Adjustable-and-Programmable-ATtiny-Based-LED-Multiplexer-SMD) by searching my name and then finding the LED Multiplexer projects. This time choose the one with " (SMD)" at the end of the name. OR, maybe more preferably, find your Stage 4

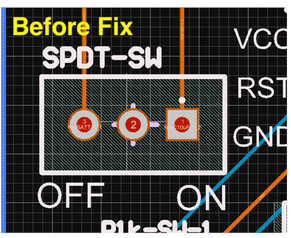
TH CircuitMaker project and fork it to create the Stage 5 SMD project (that's what I did).

To see which SMD components I chose, take a look at the BOM exported from CircuitMaker (https://www.dropbox.com/s/i8x0o3znfnhjtwl/stage_5_bom.xlsx?dl=0), which I then used to easily create this DigiKey cart (https://www.digikey.com/short/pvdfnj) containing those parts.

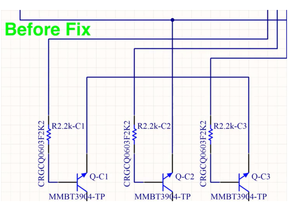
A couple other changes made for the Stage 5 design after completing Stage 4 include:

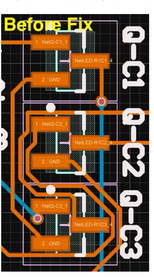
- I added + and – markers at the battery terminal so that I would know which connection was positive and which was negative without going back and looking at the design.
- In the Stage 4 circuit, the OFF and ON makers were flipped. I wanted to keep OFF on the left and ON on the right, so instead of switching the markers, I connected the positive battery voltage node to pin 3 instead of pin 2 of the SPDT-SW switch. As you will see below, this was a mistake and actually made the switch completely nonfunctional, and that was caused by me creating/editing the symbol incorrectly so that the labels for pins 1 and 2 were flipped. Really I should have just used a SPST switch here. After completing this stage, I fixed this issue in my CircuitMaker project so that you won't run into the same problem:





The other big mistake that I made in Stage 5 was to not connect the GND of Q-C1 and Q-C3 to the GND node for the rest of the circuit. This made the C1 and C3 LEDs completely nonfunctional. This issue has also been fixed in my CircuitMaker project:





If you look closely at the photos of my assembled boards below, you will see that my solutions to get my circuits working were to create a solder bridge between pins 1 and 2 of the power toggle switch and to add a jumper wire between the Q-C1/Q-C3 GND node and the GND node for the rest of the circuit.

Step 2 : Place your PCB fabrication and assembly order, and order your components for self-assembly.

Once you have completed your design in CircuitMaker (or your preferred EDA software), you will need to export and upload Gerber (https://en.wikipedia.org/wiki/Gerber_format)and NC drill (https://en.wikipedia.org/wiki/PCB_NC_formats) files (https://en.wikipedia.org/wiki/PCB_NC_formats)to PCBWay (or you preferred manufacturer) just as in Stage 4 Step 3. This time, you will also need to include Pick and Place/Centroid files and a BOM. Here (https://documentation.circuitmaker.com/display/CMAK/WorkspaceManager_Dlg-PickPlaceSetup_Form)(Pick+and+Place+Setup)_CM) is how to export the Pick and Place/Centroid files from CircuitMaker. For the BOM, you can make use of mine linked above in Step 1. Here (https://youtu.be/aEbiyIjYS6g) is a short video on how to export your own from CircuitMaker. To get mine to include the manufacturer, DigiKey part number, DigiKey URL, package, and type, I went through and added those parameters and values to all of the components in my design:

Once I had my files, I placed my order with PCBWay and ordered my components from DigiKey. When selecting turnkey assembly for the Stage 5b circuit, PCBWay will order the components themselves. For the stage 5a circuit, you can use the DigiKey cart linked above in Step 1. That cart does not include battery holders like those used in the previous stages, so you may want to add two of those – one for Stage 5a and one for Stage 5b.

When ordering from PCBWay, I suggest that you request a solder stencil (see details under the PCBWay Review at the end of this stage). Here are the details from my PCBWay order:

PCB fabrication order details

PCB assembly order details

Here is what the PCBs I received looked like:

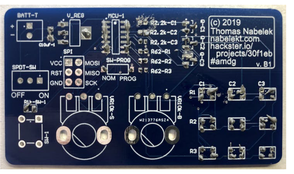
1 / 4

Step 3 : For Stage 5a only, assemble the PCB!

Self-assembly of SMD components onto a PCB can be quite different from TH soldering using a soldering iron as we have previously done. There are different methods of going about it. You should watch this video (https://youtu.be/ak1uv_6JEvs) and/or read this article (https://www.instructables.com/id/How-to-Solder-SMD-Parts/) to learn a few of those methods. My first attempt in doing this self-assembly was following their Method #2. I had only a cheap hot air gun with no flow control and had the problem of my components blowing away, as they mention. So, I settled on a variation of Method #3 – using an oven. I used a standard countertop toaster oven. DO NOT do this in an oven that you used to cook food. Don't think you want to be eating lead or flux. This is a variation on reflow soldering (https://en.wikipedia.org/wiki/Reflow_soldering), which you should take some time to learn about. Here (https://youtu.be/NY2bPGIHSeE) is another video demonstrating this method, and here are the steps that I followed:

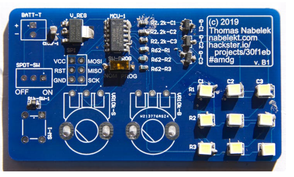
Step 3a : Lay out your components so that you can easily do a manual pick and place.

Step 3b : Apply the solder paste to the SMD pads on the PCB. This is best done using a solder mask. However, if you don't have one (I did not), simply dab some solder paste onto each one of your pads. Don't worry too much about paste getting off of the pads. The solder mask (https://en.wikipedia.org/wiki/Solder_mask) will help to prevent solder bridges between pads and help to restrict the solder to the pads once the solder paste is melted.



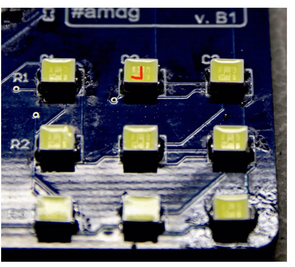
Step 3c : Pick and place your components.

Don't worry about perfectionism when it comes to aligning your component pads with your solder pads. Once the solder paste melts, the solder mask will help guide and orient your components so that the pins are directly above the solder pads.



Note that my SW-PROG orientation is flipped here! I had to fix that in Step 3e.

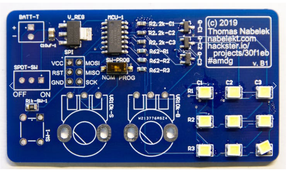
When it comes to the correct LED orientation, it isn't particularly obvious which pin is which. Looking at the LED's datasheet (https://www.cree.com/led-components/media/documents/CLA1BWKWMKW.pdf), the image on the first page shows that the corner above pin 1 (see page 12) is a bit different than the rest of the corners, but that corner is much harder to discern when looking at one of these LEDs in person. However, you can also see a backwards blocky L shape in the middle of the LED both in the datasheet image and in person, and you can use that to help you orient the LEDs:



Step 3d : "Bake" the PCB to melt the solder paste and create solder joints between your components and PCB solder pads.

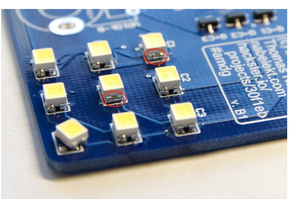
My solder paste melts at 137°C (278°F). I set my oven to about 165°C (330°F) and let it run for four minutes including the period during which the oven warmed up. This worked quite well.

Step 3e : Examine the soldered PCB to ensure conductivity where appropriate, no unintentional solder bridges, and correct pin-pad alignment.



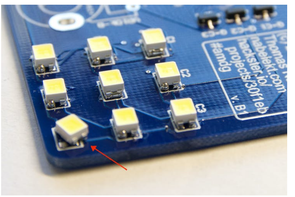
1 / 7

It might be a good idea at this point to get out your multimeter and check for conductivity (see link in Stage 3 Step 3) between the pin of one component and the pin of another component in the same node. I did this for the nodes/solder joints that I wasn't too sure about. I also checked that there was no conductivity where there shouldn't be, e.g. across the capacitors pins. Then, check for solder bridges (https://en.wikipedia.org/wiki/Solder_mask), especially at the MCU pins. The only place I found solder bridges was at some of the LEDs:



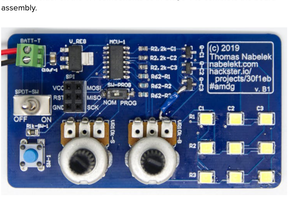
I wasn't concerned about these solder bridges because they are between pins 3 and 4 of the LEDs, and pin 4's only purpose is to help secure the LED to the PCB – it's not part of an electrical node.

I also had an obviously misaligned LED:



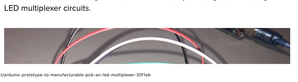
I also reversed the placement of the SW-PROG switch. I got my soldering iron, solder wick, and flux pen out to fix these issues.

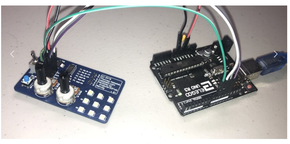
Step 3f : Solder on the TH components as in Stage 4 to complete the board assembly.



Step 4 : Program the MCU using the SPI header!

Same as Step 6 from Stage 4! But now with both your Stage 5a and Stage 5b LED multiplexer circuits.





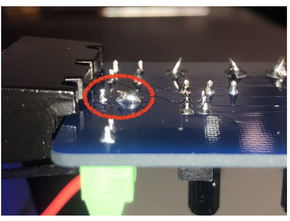
1 / 2

Connect the battery holders and the 9 V batteries, flip the power switch, and hopefully you will see that you now have two working Stage 5 circuits and are done with this project! Yay!

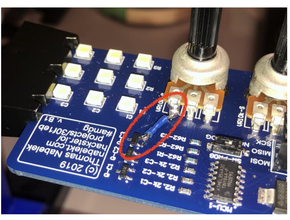
Troubleshooting

If, however, one or both of your Stage 5 circuits is not working as expected, apply Step 3e again. Step 3e should be applied to both Stage 5a and 5b. Examine the soldered PCB to ensure conductivity where appropriate, no unintentional solder bridges, and correct pin-pad alignment. If you find a solder bridge or misaligned components, you'll probably want to use your soldering iron, solder wick, and flux to redo/fix the solder joints. If you find a lack of conductivity between points that should have a connecting conductive pathway, you might need to add some solder to or redo a joint. If you didn't just copy my design and made your own, you could have a design issue. Here is how I addressed those design issues that I discussed above in Step 1.

Created a solder bridge across SPDT-SW pins 1 and 2:



Wired the GND node of Q-C1 and Q-C3 to the GND node of the rest of the PCB:



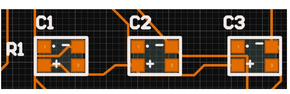
Another option there might have been to use a thin strip of that copper tape that we used in Stage 3 and somehow insulated most of it.

Of course, because these boards were fabricated in the same batch using the same design, I had to apply these fixes to both the Stage 5a and Stage 5b boards. When prototyping, even boards assembled by a vendor often have to be reworked.

PCBWay Review

PCBWay (pcbway.com) (https://www.pcbway.com) was kind enough to sponsor the order for my Stage 5 PCBs, fabrication and assembly, on the condition that I write a review here, so here are some details on my experience working with PCBWay.

One of the best things about working with PCBWay is the customer support and communication. In my experience, their support representatives have been very communicative throughout the whole process from ordering to inspection post-delivery. When you place an order, their first step is to review your order and let you know about any problems they notice. You may have to go back into your EDA software to adjust your design and resubmit your files. It's nice to have another set of eyes look at things from a production perspective. When PCBWay was getting ready to assemble by Stage 5b circuit, they weren't sure about the orientation of the LEDs – they knew which pins were the anode and cathode, but I didn't have polarity marked on my board, so they sent me an image by email to mark. (I corrected this in the CircuitMaker project for your sake):



When ordering components from digikey.cn (http://digikey.cn) (Digikey's Chineese site), the SPDT-SW switch was not in stock. From this I learned that the availability for a particular part from DigiKey in the US does not necessarily reflect availability from DigiKey in China. PCBWay asked if I wanted to ship the part to them (and thus probably delay production), or solder it on myself. I choose the latter.

When assembly was complete, they sent me images for review:

1 / 3 • PCBWay's images sent to me for correct-assembly verification

I noted three things: (1) LED-R3C3 was not properly placed. I let them know about the issue. They verified that they understood the problem correctly and then corrected it. (2) In the second image, there appears to be some imperfection in the solder mask, maybe caused by a soldering iron. This wasn't present when I inspected the boards on delivery. (3) As shown in the third image, they made an educated guess on the orientation of SW-PROG and got it right.

While board fabrication and shipping is quick (Stage 4 took 4.5 days from placing the order to delivery), assembly takes quite a bit longer than I would expect. As you can see from my order details in Step 2 above, build time was estimated at 18-20 days. Part of this time is the time it takes to ship the components from the supplier (DigiKey) to the production house (PCBWay). However, I would still expect quicker turnaround time. My order took 26 days from the time I placed it online to delivery in Colorado. In the future, for low quantity prototype production, I plan to assemble the boards myself as we did for Stage 5a.
Note : After reading this, PCBWay mentioned to me that when it comes to assembly, "if you want to get it earlier, you can note your order as an urgent one."

I would also like to see lower assembly costs for single prototype boards. $30 to solder fewer than 30 (SMD and TH) parts to a board seems like a lot. I am guessing that they might assemble a quantity of one by hand and anything more than that my machine. For my same Stage 5 design, I can get 20 boards assembled for the same $30, but then the price jumps before coming back down:

Assembly Service Price

	Per Piece	Qty	Total
✓	30.0/pcs	1	$30
	1.5/pcs	20	$30
	4.6/pcs	50	$228
	2.2/pcs	200	$443

When I ordered my boards, I expected to receive a solder stencil, but I did not. A solder stencil could have made Step 3 of Stage 5 a bit clearner and easier. When I asked PCBWay about this, I was told: "when the assembly service completed, we usually do not send the solder stencil to customer unless you have special needs... if you really want solder stencil, for your next order, we can send it to you with your special need."

As for quality, I was very pleased. The fabrication of both the Stage 4 and Stage 5 boards left me nothing to complain about. Assembly for Stage 5b was equally satisfying.

All in all, I was pleased with my PCBWay experience.

Conclusion

Now that you have walked through multiple stages of prototyping an electronic device and gone from an Arduino-based breadboard circuit to something produced in a factory, I hope that you have learned some skills that you will value and apply practically. Please feel free to leave questions and

comments below, and thanks for following along!

Schematics

Project Repository

See the various project stages for schematics. The OrCAD PSpice Designer files can be found in the project repository

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

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

Code, circuit schematic files, pictures, and screenshots related to the "Arduino Prototype to Producible PCB: An LED Multiplexer" project on hackster.io: — Read More (https://github.com/nabelekt/LED_Multiplexer#readme) **https://www.hackster.io/nabelekt/arduino-prototype-to-producible-pcb-an-led-multiplexer-301eb (https://www.hackster.io/nabelekt/arduino-prototype-to-producible-pcb-an-led-multiplexer-301eb)**

Latest commit to the master branch on 2019-12-15

Download as zip (https://github.com/nabelekt/LED_Multiplexer/archive/master.zip)

Code

Project Repository

All code and other files used in the project

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

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

Code, circuit schematic files, pictures, and screenshots related to the "Arduino Prototype to Producible PCB: An LED Multiplexer" project on hackster.io: — Read More (https://github.com/nabelekt/LED_Multiplexer#readme) **https://www.hackster.io/nabelekt/arduino-prototype-to-producible-pcb-an-led-multiplexer-301eb (https://www.hackster.io/nabelekt/arduino-prototype-to-producible-pcb-an-led-multiplexer-301eb)**

Latest commit to the master branch on 2019-12-15

Download as zip (https://github.com/nabelekt/LED_Multiplexer/archive/master.zip)

Credits

(/nabelekt)

Thomas Nabelek (/nabelekt)

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

Start the conversation!

Similar projects you might like

There are no projects.