LED candles: simple, easy, cheap

Created by Sam Clippinger

https://learn.adafruit.com/led-candles-simple-easy-cheap

Last updated on 2023-08-29 02:57:24 PM EDT

Table of Contents

Overview

LED candles are very common these days. It's easy to see the attraction -- there's no fire danger and they never need to be replaced (just batteries). But there's such a wide variation in price and quality that it's hard to know what to purchase. Some store-bought candles flicker very realistically, others just seem to blink. Some use plain white LEDs, some look like real flames. And there's no way to know how good a candle will look without buying it.

Thankfully, with Adafruit's Gemma microcontroller and an RGB LED pixel, making your own LED candle only takes a couple of minutes. Because the LED can create any color and the Gemma can control the flickering very precisely, it will look better than any store-bought candle ever could. Because the parts are so cheap, it will cost much less too!

This guide was written for the Gemma v2 board, but can be done with either the v2 or Gemma M0. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers!

Before you get started, follow the Gemma M0 guide () or the Classic Introducing GEMMA guide ()

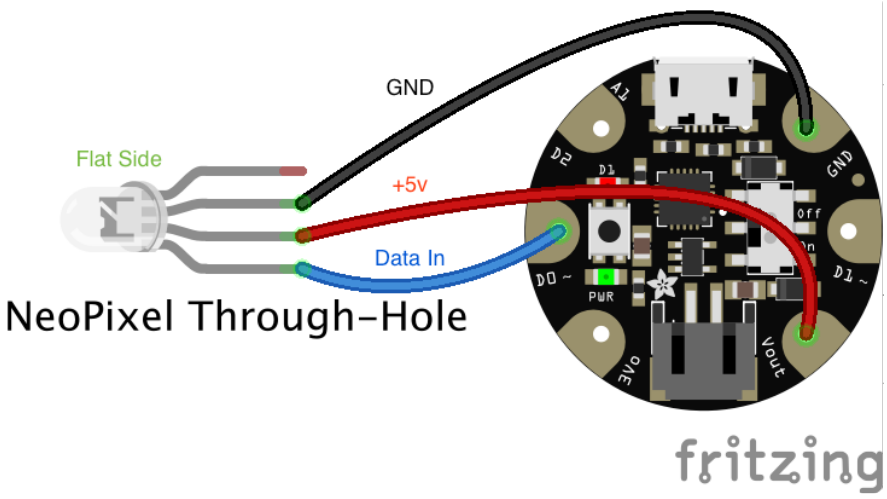
Parts

Each candle needs:

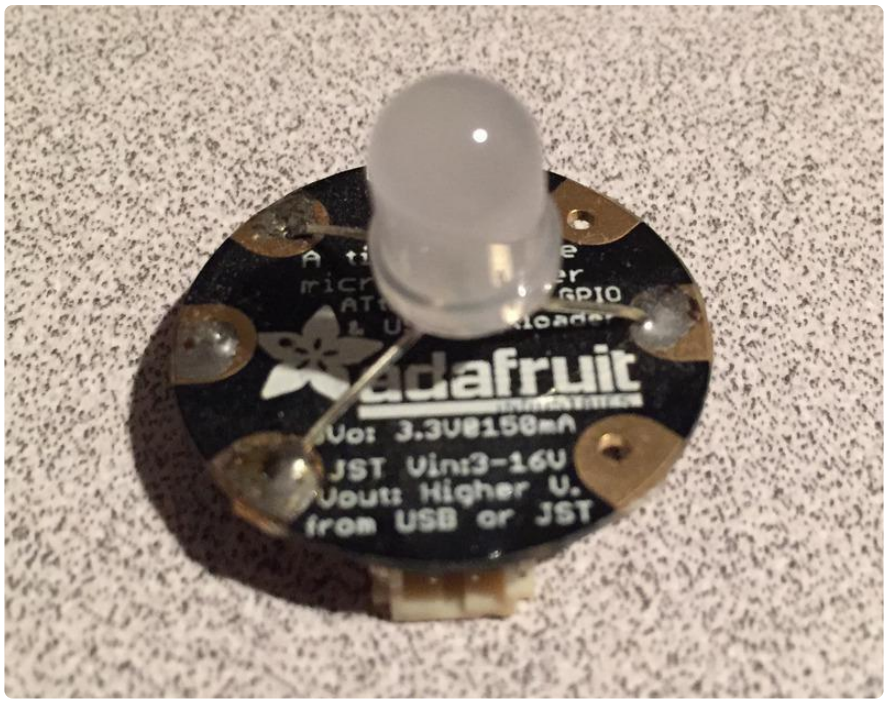
- one Gemma M0 () or GEMMA v2 wearable microcontroller (http://adafru.it/1222)
- one through-hole diffused NeoPixel 5mm () or NeoPixel 8mm ()
- a power source - either a battery pack () or a USB cable.

You'll also need a soldering iron and a computer.

Build It!



This diagram uses the original Gemma but you can also use the Gemma M0 with the exact same wiring!



Note: We will be soldering to the backside of the Gemma.

To assemble the candle, bend the pins so they reach the correct connectors on the Gemma.

- LED data-in to Gemma D0
- LED 5V to Gemma Vout
  LED Ground to Gemma GND

The LED's data-out pin is not needed and located closest to the flat side of the LED. You can cut it off or just bend it out of the way.

Put a drop of solder on each connection to hold it.

Arduino Code

The Arduino code presented below works equally well on all versions of GEMMA: v2 and M0. But if you have an M0 board, consider using the CircuitPython code on the next page of this guide, no Arduino IDE required! Click to Download the NeoPixel Library

Installing Arduino libraries is a frequent stumbling block. If this is your first time, or simply needing a refresher, please read the All About Arduino Libraries () tutorial. ()If the library is correctly installed (and the Arduino IDE is restarted), you should be able

to navigate through the “File” rollover menus as follows:

File→Sketchbook→Libraries→Adafruit_NeoPixel→strandtest

```
// SPDX-FileCopyrightText: 2017 Mikey Sklar for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>

// The onboard red LED's pin
#define REDLED_PIN              1
// The data-in pin of the NeoPixel
#define WICK_PIN                0
// Any unconnected pin, to try to generate a random seed
#define UNCONNECTED_PIN         2

// The LED can be in only one of these states at any given time
#define BRIGHT                  0
#define UP                      1
#define DOWN                    2
#define DIM                     3

#define BRIGHT_HOLD             4
#define DIM_HOLD                5

// Percent chance the LED will suddenly fall to minimum brightness
#define INDEX_BOTTOM_PERCENT    10
// Absolute minimum red value (green value is a function of red's value)
#define INDEX_BOTTOM            128
// Minimum red value during "normal" flickering (not a dramatic change)
#define INDEX_MIN               192
// Maximum red value
#define INDEX_MAX               255

// Decreasing brightness will take place over a number of milliseconds in this range
#define DOWN_MIN_MSECS          20
#define DOWN_MAX_MSECS          250
// Increasing brightness will take place over a number of milliseconds in this range
#define UP_MIN_MSECS            20
#define UP_MAX_MSECS            250
// Percent chance the color will hold unchanged after brightening
#define BRIGHT_HOLD_PERCENT     20
// When holding after brightening, hold for a number of milliseconds in this range
#define BRIGHT_HOLD_MIN_MSECS   0
#define BRIGHT_HOLD_MAX_MSECS   100
// Percent chance the color will hold unchanged after dimming
#define DIM_HOLD_PERCENT        5
// When holding after dimming, hold for a number of milliseconds in this range
#define DIM_HOLD_MIN_MSECS      0
#define DIM_HOLD_MAX_MSECS      50

#define MINVAL(A,B)             (((A) < (B)) ? (A) : (B))
#define MAXVAL(A,B)             (((A) > (B)) ? (A) : (B))

Adafruit_NeoPixel *wick;
byte state;
unsigned long flicker_msecs;
unsigned long flicker_start;
byte index_start;
byte index_end;

void set_color(byte index)
  {
  index = MAXVAL(MINVAL(index, INDEX_MAX), INDEX_BOTTOM);
  if (index >= INDEX_MIN)
```

```
    wick->setPixelColor(0, index, (index * 3) / 8, 0);
  else if (index < INDEX_MIN)
    wick->setPixelColor(0, index, (index * 3.25) / 8, 0);

  wick->show();
  return;
  }

void setup()
  {
  // There is no good source of entropy to seed the random number generator,
  // so we'll just read the analog value of an unconnected pin.  This won't be
  // very random either, but there's really nothing else we can do.
  //
  // True randomness isn't strictly necessary, we just don't want a whole
  // string of these things to do exactly the same thing at the same time if
  // they're all powered on simultaneously.
  randomSeed(analogRead(UNCONNECTED_PIN));

  // Turn off the onboard red LED
  pinMode(REDLED_PIN, OUTPUT);
  digitalWrite(REDLED_PIN, LOW);

  wick = new Adafruit_NeoPixel(1, WICK_PIN, NEO_RGB + NEO_KHZ800);
  // wick = new Adafruit_NeoPixel(1, WICK_PIN); // for RGBW, if you see green
uncomment this line

  wick->begin();
  wick->show();

  set_color(255);
  index_start = 255;
  index_end = 255;
  state = BRIGHT;

  return;
  }

void loop()
  {
  unsigned long current_time;

  current_time = millis();

  switch (state)
    {
    case BRIGHT:
      flicker_msecs = random(DOWN_MAX_MSECS - DOWN_MIN_MSECS) + DOWN_MIN_MSECS;
      flicker_start = current_time;
      index_start = index_end;
      if ((index_start > INDEX_BOTTOM) &&
          (random(100) < INDEX_BOTTOM_PERCENT))
        index_end = random(index_start - INDEX_BOTTOM) + INDEX_BOTTOM;
      else
        index_end = random(index_start - INDEX_MIN) + INDEX_MIN;

      state = DOWN;
      break;
    case DIM:
      flicker_msecs = random(UP_MAX_MSECS - UP_MIN_MSECS) + UP_MIN_MSECS;
      flicker_start = current_time;
      index_start = index_end;
      index_end = random(INDEX_MAX - index_start) + INDEX_MIN;
      state = UP;
      break;
    case BRIGHT_HOLD:
    case DIM_HOLD:
      if (current_time >= (flicker_start + flicker_msecs))
        state = (state == BRIGHT_HOLD) ? BRIGHT : DIM;
```

```
      break;
    case UP:
    case DOWN:
      if (current_time < (flicker_start + flicker_msecs))
        set_color(index_start + ((index_end - index_start) * (((current_time -
flicker_start) * 1.0) / flicker_msecs)));
      else
        {
        set_color(index_end);

        if (state == DOWN)
          {
          if (random(100) < DIM_HOLD_PERCENT)
            {
            flicker_start = current_time;
            flicker_msecs = random(DIM_HOLD_MAX_MSECS - DIM_HOLD_MIN_MSECS) +
DIM_HOLD_MIN_MSECS;
            state = DIM_HOLD;
            }
          else
            state = DIM;
          }
        else
          {
          if (random(100) < BRIGHT_HOLD_PERCENT)
            {
            flicker_start = current_time;
            flicker_msecs = random(BRIGHT_HOLD_MAX_MSECS - BRIGHT_HOLD_MIN_MSECS) +
BRIGHT_HOLD_MIN_MSECS;
            state = BRIGHT_HOLD;
            }
          else
            state = BRIGHT;
          }
        }

      break;
    }

  return;
  }
```

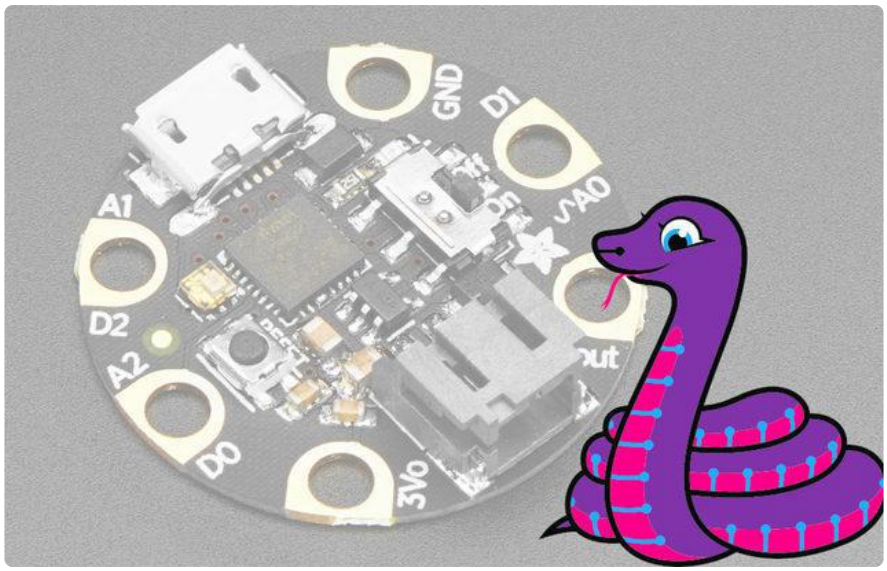
From the Tools→Board menu, select the device you are using:

- Adafruit Gemma M0
- Adafruit Gemma 8 MHz

Connect the USB cable between the computer and your device. The original Gemma (8 MHz) need the reset button pressed on the board, then click the upload button (right arrow icon) in the Arduino IDE. You do not need to press the reset on the newer Gemma M0.

When the battery is connected, you should get a light show from the LEDs. All your pixels working? Great! You can take apart this prototype and get ready to put the pixels in the collar. Refer to the NeoPixel Uberguide () for more info.

CircuitPython Code



GEMMA M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on GEMMA M0. If you've overwritten it with an Arduino sketch, or just want to learn the basics of setting up and using CircuitPython, this is explained in the Adafruit GEMMA M0 guide ().

These directions are specific to the "M0" GEMMA board. The original GEMMA with an 8-bit AVR microcontroller doesn't run CircuitPython...for those boards, use the Arduino sketch on the "Arduino code" page of this guide.

Below is CircuitPython code that works similarly (though not exactly the same) as the Arduino sketch shown on a prior page. To use this, plug the GEMMA M0 into USB...it should show up on your computer as a small flash drive...then edit the file "main.py" with your text editor of choice. Select and copy the code below and paste it into that file, entirely replacing its contents (don't mix it in with lingering bits of old code). When you save the file, the code should start running almost immediately (if not, see notes at the bottom of this page).

If GEMMA M0 doesn't show up as a drive, follow the GEMMA M0 guide link above to prepare the board for CircuitPython.

```
# SPDX-FileCopyrightText: 2017 Mikey Sklar for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time

import board
import neopixel
```

```
from analogio import AnalogIn

try:
    import urandom as random
except ImportError:
    import random

wick_pin = board.D0  # The data-in pin of the NeoPixel
unconnected_pin = board.A0  # Any unconnected pin, to generate random seed

# The LED can be in only one of these states at any given time
bright = 0
up = 1
down = 2
dim = 3
bright_hold = 4
dim_hold = 5

# Percent chance the LED will suddenly fall to minimum brightness
index_bottom_percent = 10
# Absolute minimum red value (green value is a function of red's value)
index_bottom = 128
# Minimum red value during "normal" flickering (not a dramatic change)
index_min = 192
index_max = 255  # Maximum red value

# Decreasing brightness will take place over a number of milliseconds
down_min_msecs = 20
down_max_msecs = 250

# Increasing brightness will take place over a number of milliseconds
up_min_msecs = 20
up_max_msecs = 250

# Percent chance the color will hold unchanged after brightening
bright_hold_percent = 20

# When holding after brightening, hold for a number of milliseconds
bright_hold_min_msecs = 0
bright_hold_max_msecs = 100

# Percent chance the color will hold unchanged after dimming
dim_hold_percent = 5

# When holding after dimming, hold for a number of milliseconds
dim_hold_min_msecs = 0
dim_hold_max_msecs = 50

numpix = 1  # Number of NeoPixels
pixpin = board.D0  # Pin where NeoPixels are connected
strip = neopixel.NeoPixel(pixpin, numpix, brightness=1,
                          auto_write=True)  # initialize strip

# Random number generator is seeded from an unused 'floating'
# analog input - this helps ensure the random color choices
# aren't always the same order.
pin = AnalogIn(unconnected_pin)
random.seed(pin.value)
pin.deinit()

index_start = 255
index_start = 255
index_end = 255
state = bright


def set_color(index):
    index = max(min(index, index_max), index_bottom)
    if index >= index_min:
```

```
        strip[0] = [index, int((index * 3) / 8), 0]
    elif index < index_min:
        strip[0] = [index, int((index * 3.25) / 8), 0]


set_color(255)

while True:

    current_time = time.monotonic()

    # BRIGHT
    if state == bright:
        flicker_msecs = random.randint(
            0, down_max_msecs - down_min_msecs) + down_min_msecs
        flicker_start = current_time
        index_start = index_end

        is_index_in_range = index_start > index_bottom
        is_random_in_range = random.randint(0, 100) < index_bottom_percent
        if is_index_in_range and is_random_in_range:
            index_end = random.randint(
                0, index_start - index_bottom) + index_bottom
        else:
            index_end = random.randint(0, index_start - index_min) + index_min

        state = down

    # DIM
    elif state == dim:
        flicker_msecs = random.randint(
            0, up_max_msecs - up_min_msecs) + up_min_msecs
        flicker_start = current_time
        index_start = index_end
        index_end = random.randint(0, (index_max - index_start)) + index_min
        state = down

    # DIM_HOLD
    elif state == dim_hold:
        # dividing flicker_msecs by 1000 to convert to milliseconds
        if current_time >= (flicker_start + (flicker_msecs / 1000)):
            if state == bright_hold:
                state = bright
            else:
                state = dim

    # DOWN
    elif state == down:
        # dividing flicker_msecs by 1000 to convert to milliseconds
        if current_time < (flicker_start + (flicker_msecs / 1000)):
            index_range = index_end - index_start
            time_range = (current_time - flicker_start) * 1.0

            set_color(index_start + int(
                (index_range * (time_range / flicker_msecs))))
        else:
            set_color(index_end)

            if state == down:
                if random.randint(0, 100) < dim_hold_percent:
                    flicker_start = current_time

                    dim_max = dim_hold_max_msecs - dim_hold_min_msecs
                    flicker_msecs = random.randint(
                        0, dim_max
                    ) + dim_hold_min_msecs
                    state = dim_hold
                else:
                    state = dim
```

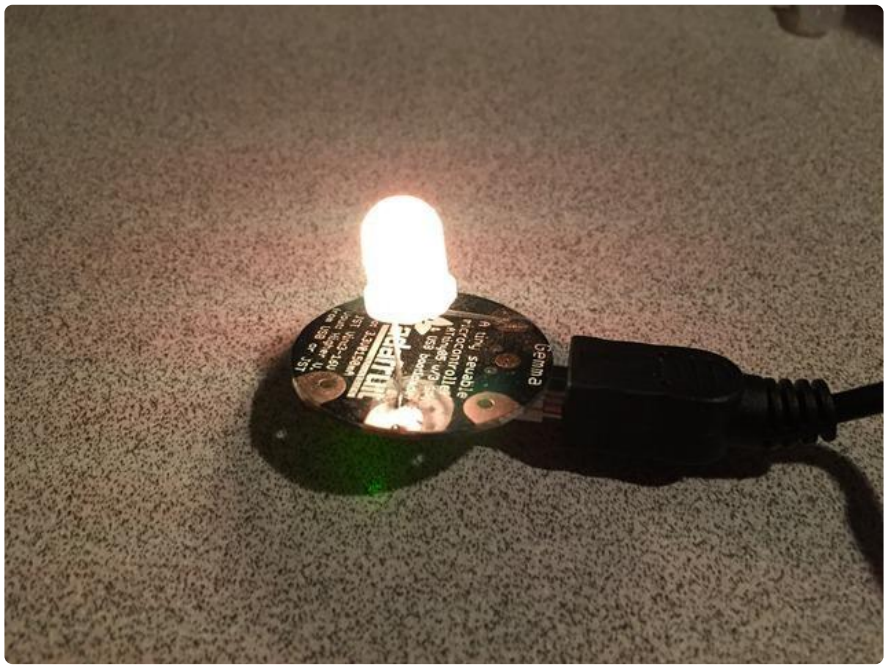
```
            else:
                if random.randint(0, 100) < bright_hold_percent:
                    flicker_start = current_time

                    max_flicker = bright_hold_max_msecs - bright_hold_min_msecs
                    flicker_msecs = random.randint(
                        0, max_flicker) + bright_hold_min_msecs
                    state = bright_hold
                else:
                    state = bright
```

This code requires the neopixel.py library. A factory-fresh board will have this already installed. If you've just reloaded the board with CircuitPython, create the "lib" directory and then download neopixel.py from Github ().

Download neopixel.py from GitHub

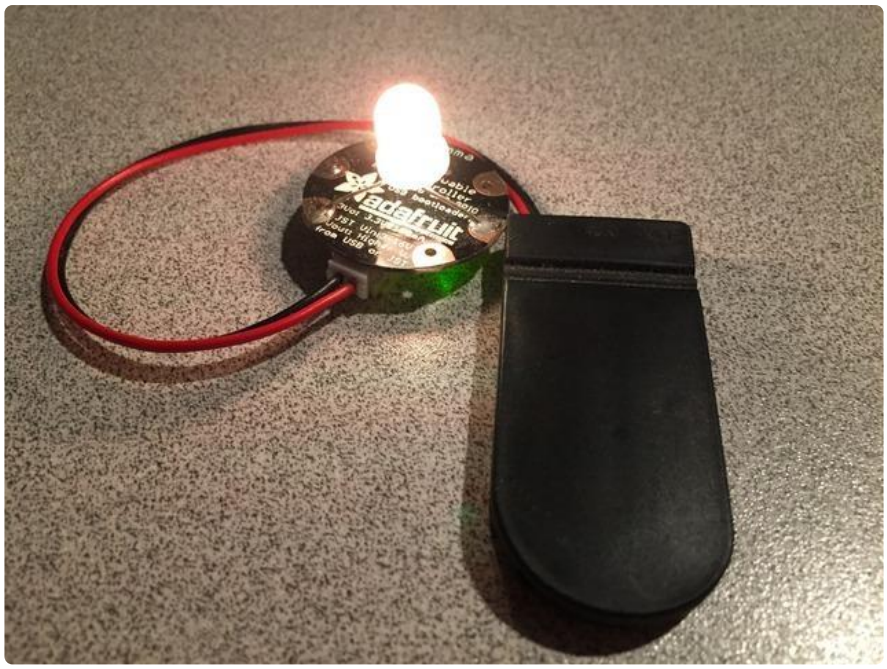
Light it up!



Power

To power the candle, you can either use a USB cable or a battery pack. Two CR2032 batteries are enough to light it up, thanks to the Gemma's voltage regulator dropping the power to 3V. I haven't done any testing to see how long they'll last; you'll know the batteries are getting low when you start seeing green flickering as the candle dims.

Since I've built several of these and I want them side-by-side, I found a cheap powered USB hub that works very nicely.



Diffuser

To make the candle look real, it will need to be covered by a semi-opaque chimney or globe to diffuse the LED's light. I found some frosted glass chimneys at my local hardware store. They're supposed to be replacements for broken fixtures, but they look great here. Almost anything is possible -- consider a structure covered in tissue paper or a white paper bag or even an old wax candle that's been hollowed out.

Final thoughts

In this project, I used one Gemma for each candle, which makes it very easy to run them from batteries and place them anywhere. But there's no reason why one Gemma couldn't run multiple LEDs if they were wired together. This could be done to create a string of candles or even a multi-wick LED candle.

This LED candle only flickers a realistic yellow/orange color, but there's no reason it couldn't have different a colored "flame", or even randomly rotate through colors.

One small caveat: because of the way NeoPixels are designed, their three internal LEDs (red, green and blue) constantly flicker at 400 Hz to hold a color. This is far too fast to be seen with the human eye, so they appear to have a steady, smooth light. But cameras can see the refresh flicker. In still photos, the effect creates "banding" in the photo (visible in the photos above). In videos, the effect creates constantly moving horizontal bands that are very noticable. DotStar LEDs don't have this problem; hopefully Adafruit will come out with through-hole versions of those soon!