Getting Started

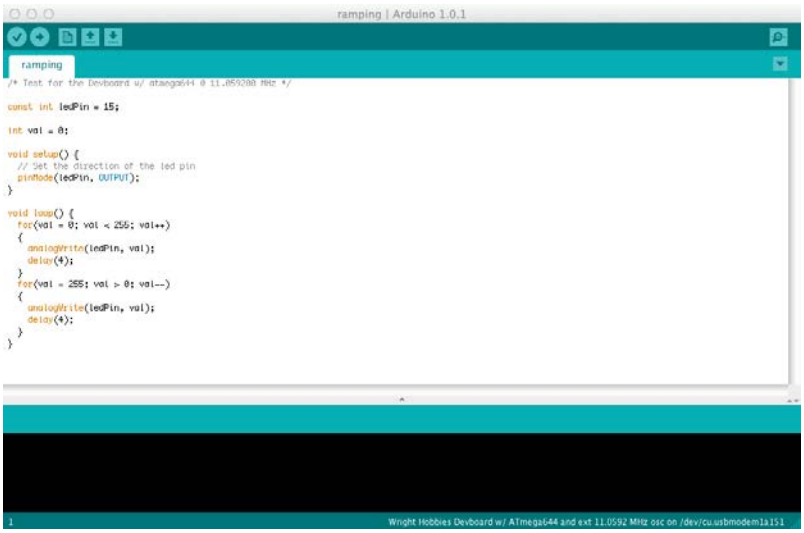
Adafruit's Smart NeoPixels

using Adafruit's Flora and Gemma

By: Anthony Garcia

The Arduino IDE

The Adafruit Flora and Gemma microcontrollers take advantage of the Arduino Integrated Development Environment used for Arduino based boards. This tool is needed to upload your projects (also commonly called sketches) to you microcontroller boards. You could start by downloading the IDE here, but experience has shown this can be a bit more challenging for the first time user, so we will follow an easier installation method.



Preconfigured Arduino IDE

Because the Adafruit Flora and Gemma are not Arduino boards, the IDE does not know how to communicate with them. We will need to tell the IDE about the boards and how to talk to them and program them. Adafruit has a detailed tutorial on this found here if you already have the Arduino IDE installed on your system, but we will go through the easy installation method provided by Adafruit found here.

On the page provided by the link above, scroll down to the “Easy Installation” section. Select the appropriate preconfigured Arduino IDE. Once you have downloaded and installed the preconfigured Arduino IDE follow the steps below for your platform to finish the install.

Windows Setup

Before plugging in the Flora or Gemma (or any other microcontroller), you will need to install the windows driver found here. The default option during installation will be all you will need.

Mas OSX Setup

If you're using Mac OS Mavericks you will need to update the setting to permit running Arduino IDE

1. Click the Lock Icon and Login
2. Change "Allow Apps Downloaded From": to "Anywhere"
3. Open the downloaded IDE.
4. Go back to the Security preferences and change the selection back to "Mac App Store and identified developers"
5. You only need to go through this procedure once. Mavericks will remember that it's OK to run the app.

Linux Setup

To use Adafruit's boards with most modern Linux distributions you'll want to ensure a few udev rules are applied. These rules apply special configuration for USB devices like Adafruit's boards to workaround or fix common issues. Specifically these rules allow Trinket and other boards to be programmed by the Arduino IDE that's run as a normal non-root user. The rules also fix an issue with ModemManager hanging on to /dev/ttyACM devices when using Flora or Bluefruit Micro boards.

To install the rules you'll want download them and copy to the location of udev rules on your system. For most Linux systems like Ubuntu, etc. udev rules are stored in /etc/udev/rules.d/ (check your distribution's documentation / help forums if you don't see this folder exists). Run the following commands:

```
wget https://github.com/adafruit/Trinket_Arduino_Linux/raw/master/99-adafruit-boards.rules

sudo cp 99-adafruit-boards.rules /etc/udev/rules.d/
```

Note that you might need to change the rule if you're using a distribution other than Ubuntu/Debian. In particular the first rule in the file applies the 'dialout' group to Trinket, etc. boards so they can be programmed without running the Arduino IDE as root. Some distributions use a different group for this instead of 'dialout'. Check your distribution docs or help forums to see what group should apply to devices that can be accessed by non-root users.

Next you'll need to reload udev's rules so that they are properly applied. You can restart your machine, or run a command like the following:

```
sudo reload udev
```

If the command above fails, try instead running:

```
sudo udevadm control --reload-rules

sudo udevadm trigger
```

And if that still fails reboot your system as it will ensure udev picks up the new configuration.

Building Your First Project

Now you are ready to start working on your first project! Before starting to write code or copying code into your project, we must ensure our settings are setup for the board we are going to use.

Windows

-Flora

After opening the Arduino IDE navigate to Tools>Board and select “Adafruit Flora”, then navigate to Tool>Serial Port and select the COM option (it should not be COM1/2, but COM3/4).

-Gemma

After opening the Arduino IDE navigate to Tools>Board: and select “Adafruit Gemma 8MHz”, then navigate to Tools>Programmer and select “USBtinyISP”

Mac

-Flora

After opening the Arduino IDE navigate to Tools>Board and select “Adafruit Flora”, then navigate to Tool>Serial Port and select the option with the phrase “usbmodem”.

-Gemma

After opening the Arduino IDE navigate to Tools>Board: and select “Adafruit Gemma 8MHz”, then navigate to Tools>Programmer and select “USBtinyISP”

Sdf

Now we have to check to make sure we have the appropriate libraries available to use for using the Adafruit NeoPixels. Navigate to Sketch>Include Library>Manage Libraries. In the search bar found at the top right, type “neopixel” and hit ENTER/RETURN. If not already installed, you will find a library called “Adafruit NeoPixel by Adafruit”. Click on that and then click the “Install” button and click “Close” when finished.

Programming

You are now ready to start coding! To get you started, we will blink the onboard NeoPixel for the Flora or blink the LED for the Gemma. Select all pre-existing code in the IDE and paste the following in its place:

Flora:

```
#include <Adafruit_NeoPixel.h>     //includes necessary function/methods for using the NeoPixel

#define PIN 8      //Assigns the D8 pin on the Flora as the pin we hook the data wire up to

Adafruit_NeoPixel strip = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);//Creates a NeoPixel object

void setup()
{
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}

void loop()
{
  // Some example procedures showing how to display to the pixels:
  colorWipe(strip.Color(255, 0, 0), 500); // Red
  colorWipe(strip.Color(0, 255, 0), 500); // Green
  colorWipe(strip.Color(0, 0, 255), 500); // Blue
  rainbowCycle(20);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait)
{
  for(uint16_t i=0; i<strip.numPixels(); i++)
{
      strip.setPixelColor(i, c);
      strip.show();
      delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait)
{
  uint16_t i, j;

  for(j=0; j<256*5; j++) // 5 cycles of all colors on wheel
  {
    for(i=0; i< strip.numPixels(); i++)
    {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos)
{
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85)
  {
```

```
   return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
 else if(WheelPos < 170)
 {
    WheelPos -= 85;
   return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
 else
 {
  WheelPos -= 170;
   return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
}
```

Gemma:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.

  To upload to your  or Trinket:
  1) Select the proper board from the Tools->Board Menu (Arduino Gemma if
     teal, Adafruit Gemma if black)
  2) Select the uploader from the Tools->Programmer ("Arduino Gemma" if teal,
            "USBtinyISP" if black Gemma)
  3) Plug in the Gemma into USB, make sure you see the green LED lit
  4) For windows, make sure you install the right Gemma drivers
  5) Press the button on the Gemma/Trinket - verify you see
     the red LED pulse. This means it is ready to receive data
  6) Click the upload button above within 10 seconds
*/

int led = 1; // blink 'digital' pin 1 - AKA the built in red LED

// the setup routine runs once when you press reset:
void setup()
{
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);

}

// the loop routine runs over and over again forever:
void loop()
{
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

Now we can upload our sketch to the Flora/Gemma!

Uploading Sketch

The Arduino IDE makes uploading your sketch to the Flora and Gemma and any other compatible board very easy. Towards to the top left of the IDE you will see a dark green bar running across. On there you will find a check mark, a right-pointing arrow, a file icon, an up-pointing arrow and a down-pointing arrow. The check mark button is the Compile button. This will “compile” your code and check it for typos/syntax errors (along with other errors if any). The right-pointing arrow is the “Upload” button. This will recompile your sketch and then send it to the Flora or Gemma. The file button will open a new project/sketch. Then the last two buttons are open and save.

Generally it is a good idea to try to compile your sketch before trying to upload. But you can just click the upload button if you have written your whole program and are ready to test it out.

The results of the compilation/upload of your sketch will then be presented in the black dialog box at the bottom of the IDE. If all you see is white text, you’re good to do! Red text is very bad and will need some debugging, but that is beyond the scope of this document and can vary too much to discuss. But, if you receive errors, Adafruit provides SOME guidance on some of the errors which can be found here and here. The white text includes information as to how much memory space your sketch/program takes.

So once you have copied the corresponding example code into your IDE and your Flora/Gemma is plugged into your computer, you can now upload the sketch!

NOTE: If using the Gemma, you must use a USB 2.0 port for the system to reliably detect the Gemma bootloader (sort of its operating system). Also, the Gemma needs to be in what is called “Bootloader Mode” before you can upload a sketch to it. You can tell if your Gemma is in bootloader mode if the green light is on and the red light is pulsing. If this is the case, you’re good to go for now! If your Gemma is not pulsing a red light, before you can upload this sketch or any other sketch following, you will need to activate the “Bootloader Mode” on the Gemma by pressing and releasing the small push button found on the board. DO NOT press and hold this button! Very bad things will happen, so don’t do it.

Blink and Beyond

You’re done! If everything went smoothly without error your Flora or Gemma should be blinking the LED or NeoPixel! You could even unplug the board from your computer and power it using a power supply with the JST connecter and watch it run by itself! I would recommend taking a quick look at the code. You don’t have to necessarily understand it, but you can follow the logic and see how things work to some degree.

These sample sketches are great for testing the board and ensuring the IDE is properly set up and the boards work correctly. Now that you have your first project working, you’re ready to move on to the big one. The project you’ve had in your mind for months and are now ready to jump right in! There are some notes and key concepts that must be noted and kept in mind when designing a project. The rest of the document briefly goes over these concepts to ensure your first wearable tech project goes as smoothly as possible.

Connecting your NeoPixel(s)

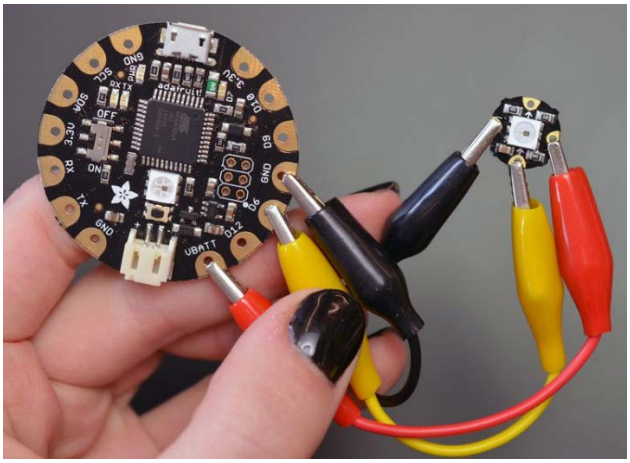
Connecting your first Smart NeoPixel to your Flora or Gemma will be very exciting, so exciting you may not realize you've connected it wrong! This will be very disappointing when you try to run you sketch and it doesn't work! So here is what to look for:

Power

Each Smart NeoPixel has 4 contact points, power, ground, input and output. The power contact is indicated by the white plus sign (+). The ground contact is indicated by the white minus sign (-). The input is indicated by the white arrow pointing towards the white spuare(the actual NeoPixel), and the output is indicated by the white arrow pointing away from the white square.

Both the Flora and Gemma also have power and ground contacts, but are labeled differently. Their power contact(s) are indicated by their voltage output. The Flora has two 3.3V power contacts, while the Gemma has one 3Vo power contact. Each board also has an unregulated raw battery voltage contact, VBATT or Vout respectively. Then their ground contacts are indicated by "GND". The Flora/Gemma also has I/O contacts (input and output on the same contact). These are indicated by the letter D followed by a number (ex. D6). Also, both Flora and Gemma are only rated for power supplies ranging from 3V-9V. Any more than 9V will put you at risk of damaging the board.

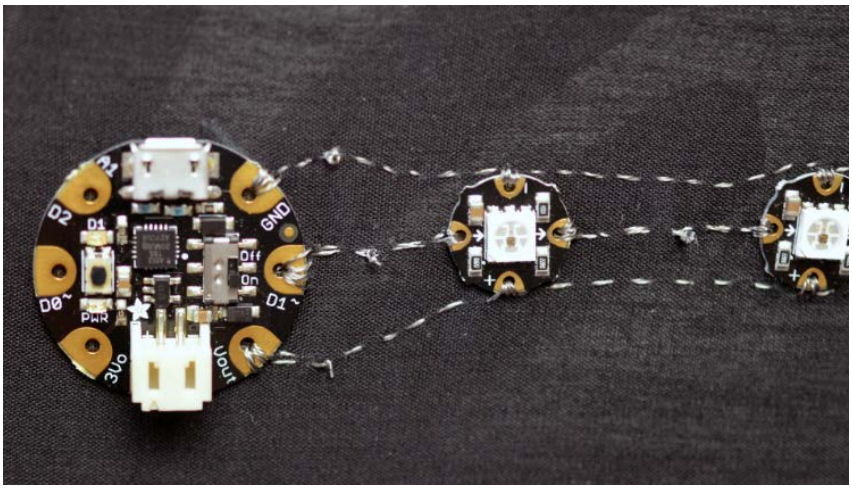
The minus on the first NeoPixel in your chain or strip must be connected to the GND on the Flora/Gemma, and the plus on the NeoPixel must be connected to either 3.3V/3Vo or VBATT/Vout. Then the input must be connected to one of the output pins on the Flora/Gemma. Additional NeoPixels can be added by connecting it to the last connected NeoPixel. Match the plus with plus, minus with minus, and each output from the previous NeoPixel must go into the input of the next NeoPixel. For testing code or NeoPixels, I recommend using Alligator Clips to temporarily hookup your NeoPixel(s) to your Flora/Gemma.

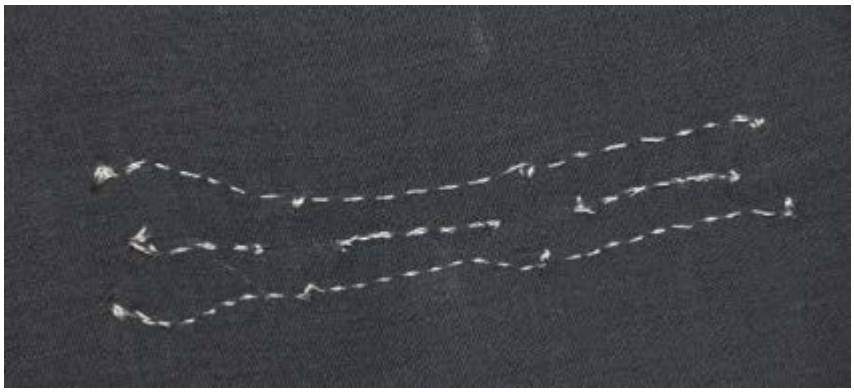


And that's it! That is the basic connection configuration of your NeoPixels. Of course different functions or patters must be connected to different I/O contacts. The number of different patterns will determine which board to use or how many boards to use (Flora has 8 I/O pins while Gemma has 3 I/O pins).

Sewing with Conductive Thread (less than 10 NeoPixels)

When you are done testing and are ready to sew your NeoPixels to your wearable tech project, There are some things to watch out for that you wouldn't have to with wire or alligator clips. Because the conductive thread is bare and exposed, there is a possibility of a short. Special care should be taken to the way the thread is sewn through the fabric to ensure no two threads ever touch, EVER. Your threading should be neat, tight and never cross. You should never have your Flora/Gemma and NeoPixels powered while sewing to prevent the risk of a short. The power and ground threads should be continuous, while your data thread should have a break between the input and output contact on each NeoPixel. All loose thread should be tightened or trimmed appropriately. Not following these rules and concepts will greatly decrease the easy and fun of your project. The examples below show the correct way for sewing your NeoPixels.





Connecting NeoPixels with Stranded Core Wire or Copper Braid (more than 10 NeoPixels)

If you will be using more than 10 NeoPixels chained together, the easy conductive thread just simply will not do. Its conductivity is not large enough to carry the necessary current to all of the NeoPixels to function properly. To avoid this issue you have two options:

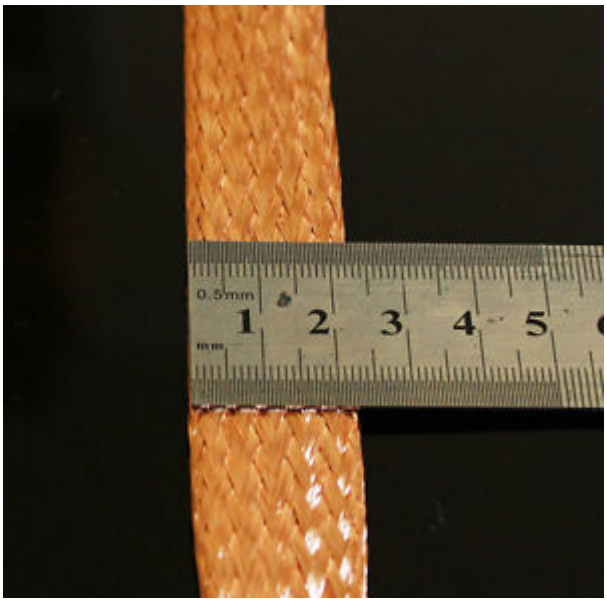
1. Stranded Core Wire
2. Copper Braid Wire

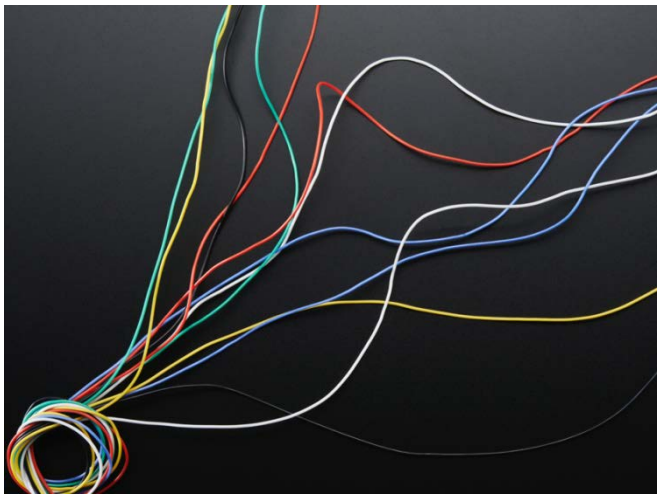
These two type of wire are able to carry the necessary power needed for 10+ NeoPixels. Your project may dictate which of these two wire you use.

It is generally easier to stitch the copper braid to the cloth along the way but it, like the conductive thread, is bare and prone to shorting if two of the braids ever touch.

The stranded core wire is insulated so you don't have to worry about shorting you project and come in various colors, but is harder to stitch to fabric neatly. But, if you so desire, you can get a braid sleeve to cover all the wire between the NeoPixels, but this adds more work and time and money.

Both options are very viable and doable and is up to you. The stranded wire usually comes in 25ft spools while the copper braid come in 50ft spools for about the same price.





Sample Code

For people new to programming or not concerned about it but would like to run some simple tests or see the functionality of their NeoPixels, Adafruit has made available same projects and methods for use! It shows the code to change the color of your NeoPixels, make them blink, pulsate, make a rainbow, spin different colors inside each individual NeoPixel and many more! These sample codes can be found here.